

# Contents

[Документация по пакету SDK Bot Framework](#)

[Обзор](#)

[Что такое пакет SDK Bot Framework](#)

[Новые возможности](#)

[Краткое руководство](#)

[Создание бота с помощью .NET](#)

[Создание бота с помощью JavaScript](#)

[Создание бота с помощью Python](#)

[Создание бота с помощью службы Azure Bot](#)

[Учебники](#)

[1. Создание базового бота](#)

[2. Развёртывание базового бота](#)

[3. Добавление QnA Maker и повторное развертывание бота](#)

[Примеры](#)

[Репозиторий примеров для Bot Framework на сайте GitHub](#)

[Безопасность](#)

[Базовые средства безопасности](#)

[Шифрование в Службе Bot](#)

[Основные сведения об аутентификации](#)

[Типы проверки подлинности](#)

[Аутентификация пользователей](#)

[Основные сведения об аутентификации пользователя](#)

[Поставщики удостоверений](#)

[Единый вход](#)

[Добавление проверки подлинности в бот](#)

[Добавление функции единого входа в бот](#)

[Расширенная аутентификация](#)

[Рекомендации по безопасности](#)

[Основные понятия](#)

## Бот

Принципы работы бота

Управление состоянием

Управляемые событиями беседы

Библиотека диалогов

Каскадные диалоги

ПО промежуточного слоя

Описание структуры эхо-бота

## Адаптивные диалоги

Общие сведения об адаптивных диалогах

События и триггеры в адаптивных диалогах.

Действия в адаптивных диалогах

Запрос ввода данных пользователем с помощью адаптивных диалогов

Распознаватели в адаптивных диалогах

Создание текста в адаптивных диалогах

Области памяти и управление состоянием в адаптивных диалогах

Управление прерываниями в адаптивных диалогах

Использование декларативных ресурсов в адаптивных диалогах

Перекрестное обучение моделей LUIS и QnA Maker

## Управление ресурсами бота

### Принцип работы ботов для Microsoft Teams

## Навыки

Сведения о навыках

Сведения о ботах навыка

Сведения о потребителях навыков

## Адаптивные выражения

## Создание текста

## Инструкции

## Конструирование

Принципы разработки ботов

Первое взаимодействие

Разработка элементов бота

- Проектирование потока беседы и управление им
- Разработка навигации для бота
- Проектирование взаимодействия с пользователем
- Шаблоны
  - База знаний
  - Передача ведения диалога человеку
  - Использование ботов в приложениях
  - Использование ботов на веб-сайтах
- Разработка
  - Отправка и получение текстовых сообщений
  - Добавление мультимедиа в сообщения
  - Добавление кнопок для управления действиями пользователя
  - Сохранение данных пользователя и диалога
  - Запрос пользователям на ввод данных
  - Отправка приветственного сообщения пользователям
  - Отправка упреждающих уведомлений пользователям
  - Управление долгосрочными операциями
  - Реализация процесса общения
  - Управление сложностью диалогов
  - Создание сложного потока беседы с использованием ветвления и циклов
  - Обработка прерываний со стороны пользователя
  - Завершение срока действия диалога
  - распознавание языка;
- LUIS
  - Добавление возможности распознавания естественного языка в функционал бота
  - Развертывание ресурсов LUIS с помощью команд CLI для LUIS
  - Обновление ресурсов LUIS с помощью команд CLI для LUIS
- QnA Maker
  - Использование QnA Maker для ответов на вопросы пользователя
  - Развертывание базы знаний QnA Maker с помощью команд CLI
  - Настраиваемые списки синонимов в QnA Maker
  - Использование нескольких моделей LUIS и QnA

## Адаптивные диалоги

Создание проекта бота с использованием адаптивных диалогов

Создание бота с использованием адаптивных диалогов

Создание бота с использованием адаптивных, компонентных, каскадных и настраиваемых диалогов

Управление пользовательскими прерываниями в адаптивных диалогах

Создание бота с использованием декларативных адаптивных диалогов

Перекрестное обучение бота для использования LUIS и QnA Maker

Запись данных напрямую в хранилище

Реализация пользовательского хранилища для бота

## Создание текста

Использование шаблонов создания речи в боте

Использование настраиваемых функций при создании текста

## Навыки

Реализация навыка

Создание манифеста навыков на основе схемы версии 2.1

Создание манифеста навыков на основе схемы версии 2.0

Использование диалогов в навыке

Реализация потребителя навыка

Использование навыка с помощью диалога

Добавление телеметрии в бот

Добавление данных телеметрии в код чат-бота для ответов на вопросы

Анализ данных телеметрии бота

Использование канала Direct Line Speech в боте

## Тест

Модульное тестирование ботов

Добавление действий трассировки в бот

## Отладка

Отладка бота с помощью IDE

Отладка ботов с помощью Bot Framework Emulator

Отладка ботов из любого канала с помощью ngrok

Отладка навыка или потребителя навыка

Отладка бота с помощью проверяющего ПО промежуточного слоя

[Отладка бота с помощью файлов записей разговоров](#)

[Отладка бота с помощью адаптивных средств](#)

## [Развертывание](#)

[Развертывание бота в Azure](#)

[Непрерывное развертывание с использованием GIT в службе приложений Azure](#)

## [Управление](#)

[Управление ботом](#)

[Регистрация каналов бота](#)

[Аналитика бота](#)

[Настройка параметров бота](#)

## [Каналы](#)

[Подключение бота к каналам](#)

[Реализация возможностей для определенных каналов](#)

[Кортана \(поддержка прекращена\)](#)

[Direct Line](#)

[Сведения о Direct Line](#)

[Подключение к Direct Line](#)

[Подключение к каналу "Речь Direct Line"](#)

[Расширение Службы приложений Direct Line](#)

[Настройка бота .NET для использования расширения](#)

[Настройка бота Node.js для работы с расширением](#)

[Создание клиента .NET с использованием расширения](#)

[Использование WebChat с расширением](#)

[Использование расширения в виртуальной сети](#)

[Alexa](#)

[Email](#)

[Facebook](#)

[GroupMe;](#)

[Kik](#)

[LINE](#)

[Microsoft Teams](#)

[Skype](#)

[Skype для бизнеса](#)

[Slack](#)

[Telegram](#)

[Telephony \(Телефония\)](#)

[Twilio](#)

[WeChat](#)

[Веб-чат.](#)

[Webex](#)

[Дополнительные каналы](#)

## [Миграция](#)

[Общие сведения о переносе](#)

[Перенос ботов .NET](#)

[Различия между версиями 3 и 4 пакета SDK для .NET](#)

[Краткий справочник по миграции для .NET](#)

[Перенос бота .NET версии 3 в бот .NET Framework версии 4](#)

[Перенос бота .NET версии 3 в бот .NET Core версии 4](#)

[Использование данных .NET о состоянии пользователя версии 3 в боте версии 4](#)

[Перенос ботов JavaScript](#)

[Различия между версиями 3 и 4 пакета SDK для JavaScript](#)

[Краткий справочник по миграции для JavaScript](#)

[Перенос бота JavaScript версии 3 в бот версии 4](#)

[Использование данных JavaScript о состоянии пользователя версии 3 в боте версии 4](#)

[Преобразование бота версии 3 в навык](#)

[Общие сведения о преобразовании в навыки](#)

[Преобразование бота .NET версии 3 в навык](#)

[Преобразование бота JavaScript версии 3 в навык](#)

## [Справочник](#)

[Справочные материалы по пакету SDK версии 4](#)

[Пакет SDK версии 4 для .NET](#)

[Пакет SDK версии 4 для JavaScript](#)

[Пакет SDK версии 4 для Java \(предварительная версия\)](#)

# Пакет SDK для Python версии 4

## Справочник по REST API

### REST API для Bot Framework

[Обзор](#)

[Основные понятия](#)

[Создание бота с помощью REST](#)

[Справочник по API](#)

#### API соединителя

[Аутентификация](#)

[Общие сведения о действиях](#)

[Создание сообщений](#)

[Отправка и получение сообщений](#)

[Добавление мультимедийных вложений в сообщения](#)

[Добавление вложений в виде форматированных карточек в сообщения](#)

[Добавление речи в сообщения](#)

[Добавление подсказок для ввода в сообщения](#)

[Добавление предлагаемых действий к сообщениям](#)

[Реализация возможностей для определенных каналов](#)

[Управление данными состояния](#)

#### API 3.0 для Direct Line

[Основные понятия](#)

[Аутентификация](#)

[Начало общения](#)

[Повторное подключение к диалогу](#)

[Отправка действия боту](#)

[Получение действий от бота](#)

[Завершение диалога](#)

[Справочник по API](#)

[Файл Swagger](#)

#### API 1.1 для Direct Line

[Основные понятия](#)

[Аутентификация](#)

[Начало общения](#)  
[Отправка сообщения боту](#)  
[Получение сообщений от бота](#)  
[Справочник по API](#)  
[Файл Swagger](#)

[Программа командной строки Bot Framework](#)

[Адаптивные диалоги](#)

[События и триггеры](#)

[Действия](#)

[Входные данные](#)

[Распознаватели](#)

[Области памяти](#)

[Адаптивные выражения](#)

[Предварительно созданные функции](#)

[Справочник по интерфейсам API](#)

[LG](#)

[Справочник по интерфейсам API](#)

[Формат файлов .lg](#)

[Структурированный шаблон ответа](#)

[Функции, внедренные из библиотеки LG](#)

[Форматы файлов](#)

[Формат LU-файла](#)

[Формат QNA-файла](#)

[Сущности и типы действий](#)

[Встроенные компоненты Политики Azure](#)

[Ресурсы](#)

[Техническая поддержка](#)

[Ключи для App Insights](#)

[Соответствие требованиям в службе Azure Bot](#)

[Рекомендации по проверке бота](#)

[Справочник по использованию каналов](#)

[Руководство по идентификаторам](#)

[Реализация навыка в Power Virtual Agents](#)

[Запросы между агентом и пользователем](#)

## [ВОПРОСЫ И ОТВЕТЫ](#)

[Индекс](#)

[Доступность](#)

[Общие сведения](#)

[Экосистема](#)

[Безопасность](#)

[Azure](#)

[Диагностика](#)

[Индекс](#)

[Общие сведения](#)

[Конфигурация](#)

[Ошибки HTTP 500](#)

[Аутентификация](#)

[Схемы для Bot Framework](#)

[Схема действий](#)

[Схема карточки](#)

[Схема расшифровок](#)

[Виртуальный помощник](#)

[WebChat](#)

[Обзор](#)

[Настройка](#)

[Добавление функции единого входа в Web Chat](#)



# Что такое пакет SDK Bot Framework

27.03.2021 • 12 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

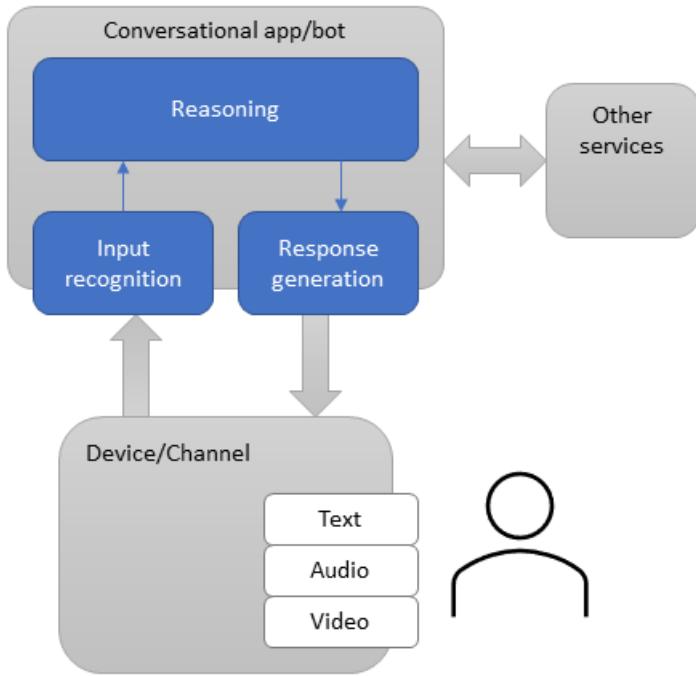
Платформа Bot, а также служба Azure Bot предоставляют средства для создания, тестирования, развертывания и управления интеллектуальными программы-роботами в одном месте. The Bot Framework включает модульный и расширяемый пакет SDK для создания программы-роботы, а также инструменты, шаблоны и связанные службы искусственного интеллекта. С помощью этой платформы разработчики могут создавать программы-роботы, использующие речь, знания естественного языка, обработки вопросов и ответов и многое другое.

## Что такое бот?

Боты обеспечивают взаимодействие, больше похожее не на работу с компьютером, а на общение с живым человеком, ну или хотя бы с очень умным роботом. Они помогут вам перенести на автоматизированные системы простые и повторяющиеся задачи, такие как резервирование столов в ресторане или сбор сведений для профиля без необходимости прямого участия человека. Пользователи взаимодействуют с ботом, используя текстовые сообщения, интерактивные карты и речь. Взаимодействие с ботом может ограничиваться простыми вопросами и ответами или представлять собой сложное интеллектуальное общение с предоставлением доступа к службам.

Робот можно рассматривать как веб-приложение с интерфейсом для общения. Пользователь подключается к роботу, например к Facebook, временному резерву или Microsoft Teams.

- Причины входа и выполнения соответствующих задач. Это может включать запрос дополнительных сведений о пользователе или доступ к службам от имени пользователя.
- Bot выполняет распознавание входных данных пользователя, чтобы интерпретировать, что пользователь запрашивает или говорят.
- Bot создает ответы для отправки пользователю, чтобы сообщить, что делает программа-робот.
- В зависимости от того, как настроена программа-робот и как она зарегистрирована в канале, пользователи могут взаимодействовать с программой-роботом через текст или речь, а диалог может включать изображения и видео.



Программы-роботы — это очень похоже на современные веб-приложения, живущие в Интернете и использующие API-интерфейсы для отправки и получения сообщений. Содержимое бота может быть самым разным в зависимости от его типа и назначения. Программное обеспечение современных ботов опирается на сложный набор технологий и средств, позволяющих предоставлять все более сложные возможности на широком спектре платформ. Но могут существовать и простейшие боты, которые умеют лишь получать текстовое сообщение и возвращать его пользователю. Для таких достаточно лишь пары строк кода.

Боты могут выполнять все те же действия, что и другие виды программного обеспечения: читать и сохранять файлы, использовать интерфейсы API и базы данных, производить вычисления. Уникальность ботов заключается в том, что кроме этого они используют механизмы, традиционно задействованные для обмена данными между людьми.

Служба Azure Bot и Bot Framework предлагают:

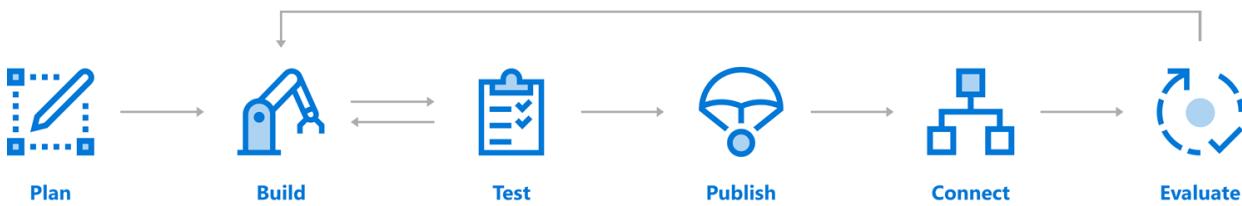
- Пакет SDK для Bot Framework для разработки программы-роботы
- средства Bot Framework для поддержки всех этапов разработки ботов;
- службу Bot Framework для отправки и получения сообщений и событий между ботами и каналами;
- возможность настройки развертывания и канала в Azure.

Кроме того, боты могут использовать другие службы Azure, такие как:

- Azure Cognitive Services для создания интеллектуальных приложений;
- служба хранилища Azure в качестве облачного хранилища.

## Сборка бота

Службы Azure Bot и Bot Framework предлагают интегрированный набор средств и служб для ускорения этого процесса. Выберите любую среду разработки или средства командной строки для создания бота. Доступны пакеты SDK для C#, JavaScript, Typescript и Python (пакет SDK для Java находится в разработке). Мы предоставляем средства, помогающие на разных этапах проектирования и разработки ботов.



## План

Как и с любым другим типом программного обеспечения, для успешного создания бота важно хорошее понимание всех целей, процессов и потребностей пользователей. Прежде чем писать код, просмотрите [руководство по разработке](#) бота, чтобы ознакомиться с рекомендациями и определить потребности своего бота. Вы можете создать бота как с простыми, так и более сложными возможностями. Это может быть распознавание речи и естественного языка или возможность отвечать на вопросы.

## Сборка

Бот — это веб-служба, которая реализует интерфейс для общения, а также взаимодействует со службой Azure Bot для отправки и получения сообщений и событий. Служба Bot Framework — это один из компонентов службы Azure Bot. Боты можно создавать в большом количестве языков и сред. Вы можете начать разработку бота на [портале Azure](#) или применить шаблоны для [C# | JavaScript | Python] для локальной разработки.

В составе служб Azure Bot и Bot Framework мы предлагаем дополнительные компоненты, которые расширяют функции бота.

КОМПОНЕНТ	ОПИСАНИЕ	ССЫЛКА
Добавление возможности обработки естественного языка	Включите в боте возможность распознавать естественные языки, распознавать орфографические ошибки, использовать обработку речи и распознавать намерения пользователя.	Использование <a href="#">LUIS</a>
Ответы на вопросы	Добавьте базу знаний, чтобы отвечать на вопросы в более естественной разговорной форме.	Использование <a href="#">QnA Maker</a>
Объединение нескольких моделей	При использовании нескольких моделей, например для LUIS и QnA Maker, приложение само определяет, когда и какую модель следует использовать во время диалога.	Средство <a href="#">Dispatch</a>
Добавление карточек и кнопок	Расширьте возможности взаимодействия с пользователем, чтобы кроме текста можно было использовать элементы мультимедиа, например изображения, меню и карточки.	<a href="#">Добавление карточек</a>

### NOTE

В таблице выше приведен неполный список доступных компонентов. Для дополнительных сведений о функциональности бота просмотрите статьи слева, начиная с [отправки сообщений](#).

Кроме того, мы предоставляем средства командной строки для создания, администрирования и проверки ресурсов ботов. Эти средства настраивать приложения LUIS, создавать базы знаний QnA, создавать модели отправки между компонентами, имитировать беседу и многое другое. Дополнительные сведения см. в файле [readme](#) для средств командной строки.

Также у вас есть доступ к различным [примерам](#), которые демонстрируют многие возможности, доступные через пакет SDK. Они отлично подходят для разработчиков, которые ищут более широкие возможности отправной точки.

## Test

Программы-роботы — это сложные приложения с множеством различных частей, работающих вместе. Как и в любом другом сложном приложении, это может привести к некоторым нестандартным ошибкам или бот может вести себя иначе, чем ожидалось. Поэтому перед публикацией протестируйте бота. Мы предоставляем несколько способов проверить боты перед выводом в общий доступ.

- Тестирование в локальной среде с помощью [эмулятора](#). Bot Framework Emulator — это автономное приложение с интерфейсом для чата, которое содержит дополнительные средства отладки и опроса, позволяющие обнаружить проблемы с ботом и выявить их причины. Эмулятор может быть запущен локально вместе с разрабатываемым приложением-роботом.
- Проверка бота в [Интернете](#). Завершив настройку бота через портал Azure, вы сможете обратиться к нему через веб-интерфейс чата. Веб-интерфейс чата — это отличный способ предоставить доступ к боту инженерам-испытателям и другим заинтересованным лицам, у которых нет прямого доступа к исполняемому коду бота.
- [Модульное тестирование](#) программы Bot с помощью текущего пакета SDK для Bot Framework.

## Публикация

Когда все будет готово для публикации бота, передайте его содержимое в [Azure](#) или в собственной веб-службе или центре обработки данных. Постоянный адрес в общедоступном сегменте Интернета будет первым шагом к активному использованию бота на вашем сайте или в специализированных каналах для чата.

## Подключение

Подключите бот к таким каналам, как Facebook, Messenger, Kik, Slack, Microsoft Teams, Telegram, каналам текстовых сообщений, SMS, Twilio и Cortana. Платформа Bot Framework берет на себя основную часть работы по отправке и получению сообщений на всех этих платформах. Ваше приложение всегда получит унифицированный и нормализованный поток сообщений независимо от количества и типа каналов, к которым оно подключено. Сведения о добавлении каналов, см. в разделе [о каналах](#).

## Evaluate

Используйте данные, собранные на портале Azure, чтобы определить возможности для повышения производительности бота. Вы можете получить данные уровня службы и инструментирования, например трафик, задержку и интеграцию. Аналитика также поддерживает отчеты уровня общения для данных пользователя, сообщений и каналов. Дополнительные сведения см. в разделе [о сборе аналитики](#).

## Дальнейшие действия

Ознакомьтесь с [примерами внедрения](#) ботов или щелкните приведенную ниже ссылку, чтобы сразу приступить к созданию своего бота.

### Создание бота

# Новые возможности за март 2021

27.03.2021 • 2 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Пакет SDK для Bot Framework v4 — это [пакет SDK с открытым исходным кодом](#), позволяющий разработчикам моделировать и создавать сложные беседы с помощью предпочтительного языка программирования.

В этой статье перечислены новые ключевые возможности и улучшения Bot Framework и службы Azure Bot.

ЭЛЕМЕНТ	C#	JS	PYTHON	JAVA
Release	<a href="#">4.12 (GA)</a>	<a href="#">4.12 (GA)</a>	<a href="#">4.12 (GA)</a>	<a href="#">4.6 (Предварительная версия 8)</a>
Примеры	<a href="#">.NET Core, WebAPI</a>	<a href="#">Node.js, TypeScript, ES6</a>	<a href="#">Python</a>	

Добро пожаловать в выпуск пакета SDK для Bot-файла в марте 2021. Ниже перечислены некоторые ключевые особенности.

- Канал телефонии теперь доступен для примеров на раннем этапе предварительной версии.
- Microsoft Teams — новые и улучшенные примеры, адаптивные вкладки карт, Action.Exemilie (Предварительная версия, C#) и поддержка composer (Предварительная версия).
- Облачный адаптер (Предварительная версия 2, C#) повысил поддержку платформы благодаря расширенным функциональным возможностям.
- Orchestrator (Предварительная версия 3) теперь поддерживает больше языков, и документация была улучшена.
- Средства интерфейса командной строки Bot. LUIS приложения поддержка технологии нейронных технологий и многое другое.
- Робот службы работоспособности Azure. Служба Microsoft здравоохранения Bot перемещается в Azure и предоставляет организациям преимущества улучшенных средств, безопасности и соответствия требованиям Azure.
- Виртуальные агенты для Power-Bot — создание, редактирование и публикация, которые были сделаны простыми!

Дополнительные сведения об изменениях, внесенных в пакет SDK в выпуске 4.12, см. в [записках о выпуске](#) пакета SDK для Bot Framework.

**Инсайдеры!** Вы хотите как можно быстрее испытать новые возможности в деле? Вы можете скачивать сборку предварительной оценки, которая создается каждую ночь [\[C#\]](#) [\[JS\]](#) [\[Python\]](#) [\[CLI\]](#), чтобы получать все обновления сразу, как только они станут доступны. Также следите за нашими новостями, обновлениями Bot Framework и интересными материалами в Twitter @msbotframework!

## Дополнительные сведения

Предыдущие объявления можно просмотреть в [архивных данных](#).

# Создание бота с помощью пакета SDK Bot Framework для .NET

27.03.2021 • 7 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье показано, как создать робот с помощью шаблона C#, а затем проверить его с помощью эмулятора Bot Framework.

Создание бота с помощью службы Azure Bot и локальных средств — это независимые друг от друга параллельные процессы.

## Предварительные условия

- [Среда выполнения ASP.NET Core 3.1](#)
- [Bot Framework Emulator](#).
- Знание [ASP.NET Core](#) и [асинхронного программирования в C#](#)

### Шаблоны

- [Visual Studio](#)
- [Visual Studio Code/Командная строка](#)
- [Visual Studio 2019](#) или более поздней версии.
- [Шаблоны пакета SDK для Bot Framework версии 4 для Visual Studio](#)

Чтобы добавить шаблоны Bot в Visual Studio, скачайте и установите [Шаблоны пакета SDK для Bot Framework версии 4 для Visual Studio](#) VSIX.

### NOTE

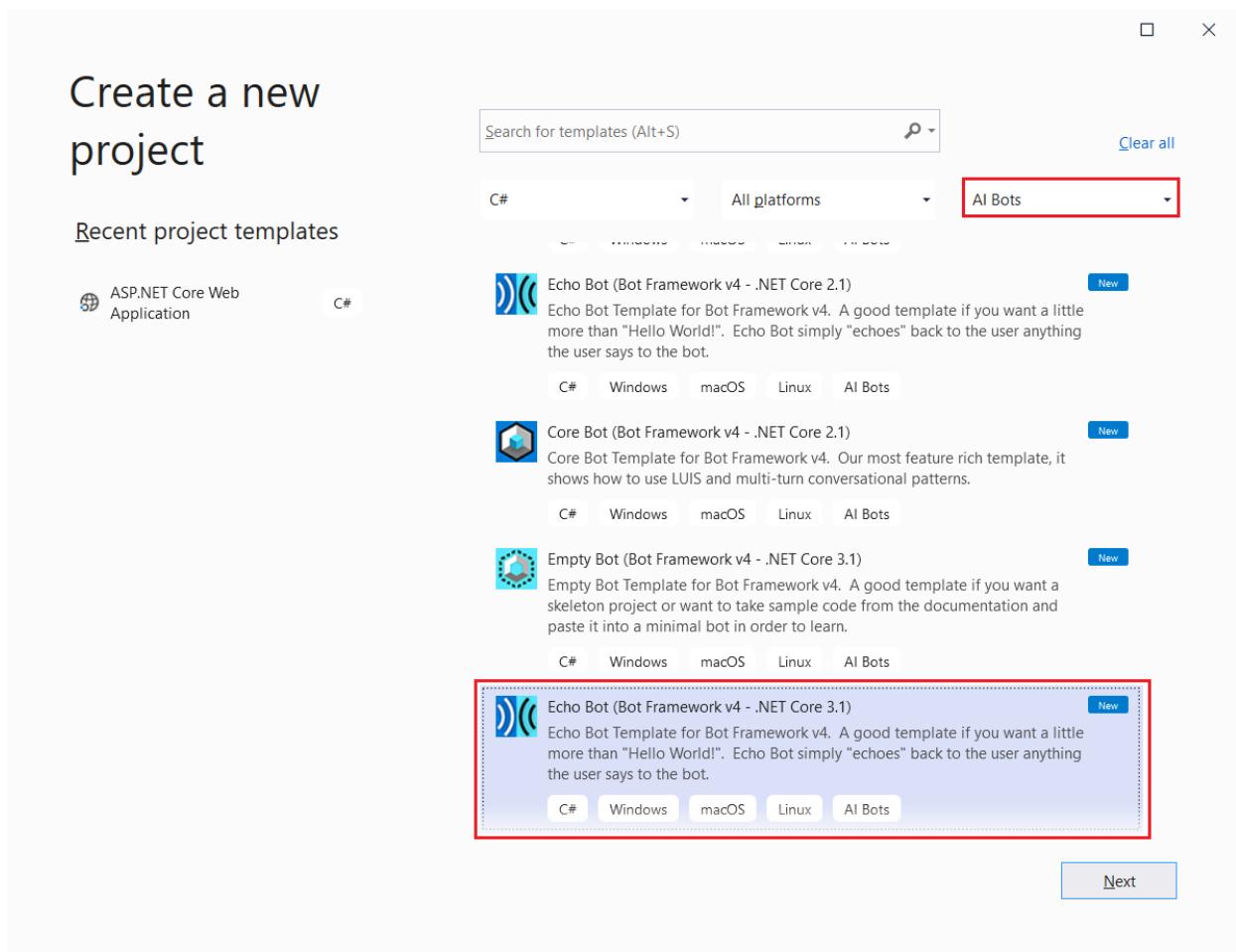
Пакет [VSIX](#) включает версии .net Core 2.1 и .net Core 3.1 шаблонов C#. При создании ботов в Visual Studio 2019 следует использовать шаблоны .NET Core 3.1. В текущих примерах ботов используются шаблоны .NET Core 3.1. Примеры, использующие шаблоны .NET Core 2.1, можно найти в ветви [4.7-archive](#) репозитория BotBuilder-Samples. Сведения о развертывании .NET Core 3.1 программы-роботы в Azure см. в статье [развертывание программы Bot в Azure](#).

## Создание бота

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Командная строка](#)

### Сборка с помощью Visual Studio

В Visual Studio создайте проект бота с использованием шаблона **эхо-бота на основе Bot Framework версии 4 и .NET Core 3.1**. Выберите тип проекта **AI Bots** (Боты (ИИ)), чтобы отображались только шаблоны ботов.



Благодаря шаблону проект содержит весь код, необходимый для создания бота в рамках этого краткого руководства. Для тестирования бота не требуется дополнительный код.

#### NOTE

При создании **основного** робота требуется модель языка Luis. Вы можете создать языковую модель по адресу [Luis.AI](#). Создав модель, обновите файл конфигурации.

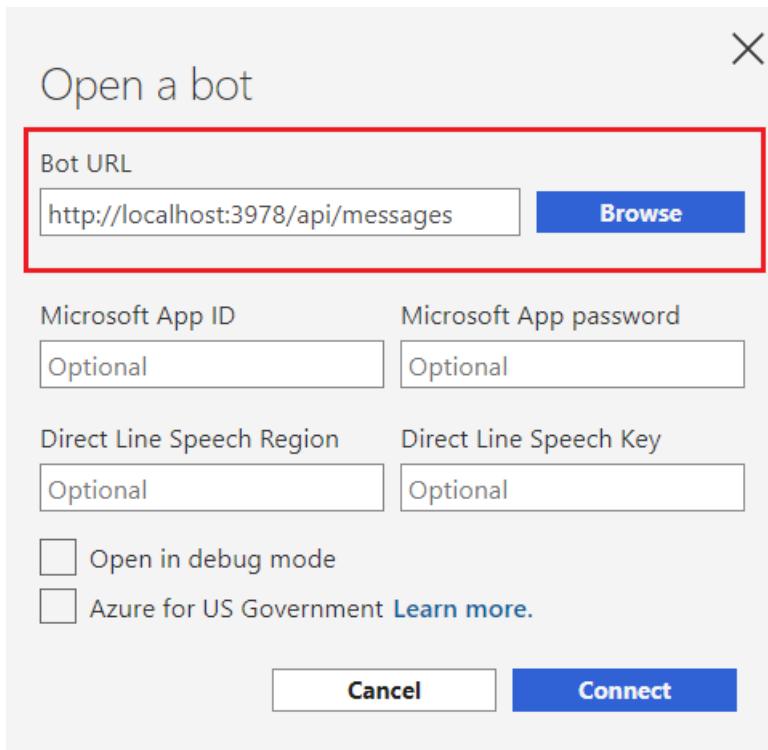
## Запуск бота

- [Visual Studio](#)
- [Visual Studio Code](#)
- [Командная строка](#)

В Visual Studio запустите проект. Visual Studio создаст приложение, развернет его на узле localhost и запустит веб-браузер для отображения страницы приложения `default.htm`. На этом этапе бот выполняется локально, используя порт 3978.

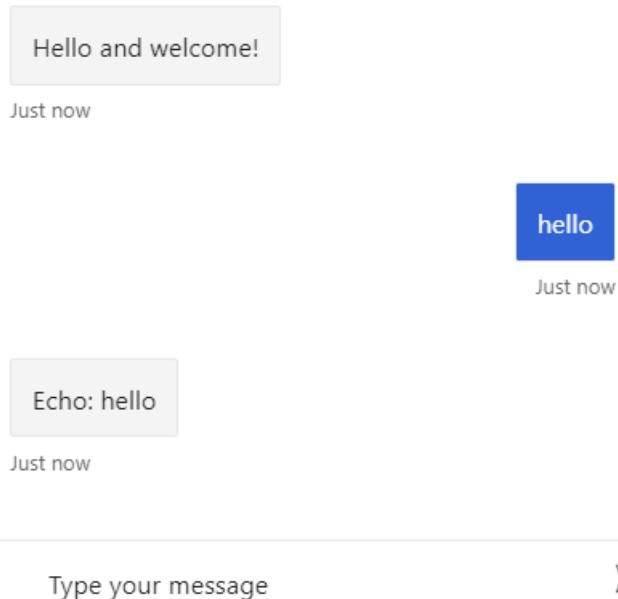
## Запуск эмулятора и подключение к боту

1. Установите Bot Framework Emulator.
2. Выберите **Открыть Bot** на вкладке **приветствия** эмулятора.
3. Введите локальный URL-адрес бота с указанием порта и пути /api/messages. Обычно этот адрес выглядит так: `http://localhost:3978/api/messages`.



4. В этом случае выберите **Подключиться**.

Отправьте сообщение в Bot, и Bot ответит обратно.



## Дополнительные ресурсы

- Дополнительные сведения о шаблонах .NET Core см. в файле сведений о [шаблонах пакет SDK для .NET Core](#).
- См. раздел [Отладка программы-робота](#) для отладки с помощью Visual Studio или Visual Studio Code и эмулятора Bot Framework.
- Сведения о том, как установить ngrok, см. в статье [туннелирование \(ngrok\)](#).

## Дальнейшие действия

[Развертывание бота в Azure](#)

# Создание бота с помощью пакета SDK Bot Framework для JavaScript

27.03.2021 • 4 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье показано, как создать робот с помощью генератора Bot Builder Yeoman и пакета SDK для Bot Framework для JavaScript, а затем проверить его с помощью эмулятора Bot Framework.

Создание бота с помощью службы Azure Bot и локальных средств — это независимые друг от друга параллельные процессы.

## Предварительные требования

- [Node.js](#)
- [Bot Framework Emulator](#).
- навыки разработки для [Restify](#) и асинхронного программирования в JavaScript.
- [Visual Studio Code](#) или любой привычный редактор кода, если вы намерены изменять код бота.

### Шаблоны

Чтобы установить Yeoman и генератор Yeoman для Bot Framework V4, сделайте следующее:

1. Откройте терминал или командную строку с повышенными привилегиями.
2. Перейдите в каталог, где вы намерены разместить боты JavaScript. Если такого каталога еще нет, создайте его.

```
mkdir myJsBots  
cd myJsBots
```

3. Убедитесь, что у вас установлены последние версии NPM и Yeoman.

```
npm install -g npm  
npm install -g yo
```

4. Установите генератор Yeoman. Yeoman — это инструмент для создания приложений.

Дополнительные сведения см. в разделе [Yeoman.IO](#).

```
npm install -g generator-botbuilder
```

### NOTE

Установка Microsoft Build Tools, указанная ниже, требуется только в том случае, если Windows используется как операционная система для разработки. Для некоторых конфигураций при установке Restify возникает ошибка, связанная с node-gyp. В таком случае попробуйте выполнить следующую команду с дополнительными разрешениями. Этот вызов также может зависнуть без завершения, если на компьютере уже установлена среда Python.

```
# only run this command if you are on Windows. Read the above note.  
npm install -g windows-build-tools
```

## Создание бота

- Используйте генератор для создания эхо-робота.

```
yo botbuilder
```

Yeoman запросит некоторые сведения для создания бота. Для задач в этом руководстве используйте значение по умолчанию.

```
? What's the name of your bot? my-chat-bot  
? What will your bot do? Demonstrate the core capabilities of the Microsoft Bot Framework  
? What programming language do you want to use? JavaScript  
? Which template would you like to start with? Echo Bot - https://aka.ms/bot-template-echo  
? Looking good. Shall I go ahead and create your new bot? Yes
```

Благодаря шаблону проект содержит весь код, необходимый для создания бота в рамках этого краткого руководства. Для тестирования бота не требуется дополнительный код.

### NOTE

Чтобы создать бот `Core`, вам потребуется языковая модель LUIS. (Языковую модель можно создать на сайте [luis.ai](#).) Создав модель, обновите файл конфигурации.

## Запуск бота

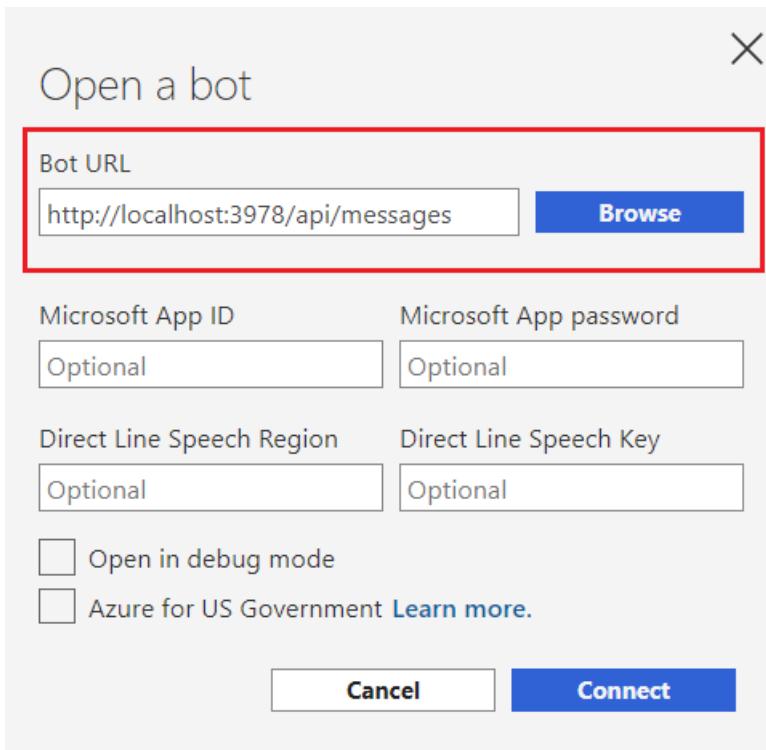
В терминале или командной строке перейдите к каталогу, созданному для бота, и запустите бот с помощью `npm start`.

```
cd my-chat-bot  
npm start
```

На этом этапе бот выполняется локально и использует порт 3978.

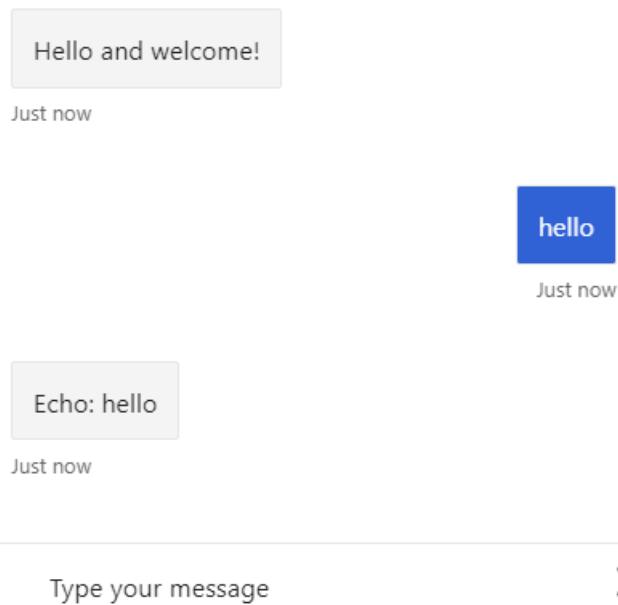
## Запуск эмулятора и подключение к боту

- Установите Bot Framework Emulator.
- Выберите **Открыть Bot** на вкладке **приветствия** эмулятора.
- Введите локальный URL-адрес бота с указанием порта и пути /api/messages. Обычно этот адрес выглядит так: `http://localhost:3978/api/messages`.



4. В этом случае выберите **Подключиться**.

Отправьте сообщение в Bot, и Bot ответит обратно.



## Дополнительные ресурсы

Подключение к удаленно размещенном боту описано в документации по [туннелированию \(ngrok\)](#).

## Дальнейшие действия

[Развертывание бота в Azure](#)

# Создание бота с помощью пакета SDK Bot Framework для Python

27.03.2021 • 3 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье показано, как создать робот с помощью шаблона Python Echo Bot, а затем проверить его с помощью эмулятора Bot Framework.

Создание бота с помощью службы Azure Bot и локальных средств — это независимые друг от друга параллельные процессы.

## Предварительные условия

- Python [3.6](#), [3.7](#) или [3.8](#)
- [Bot Framework Emulator](#).
- Знание асинхронного программирования в Python

### TIP

Некоторые разработчики могут оказаться полезными при создании программы-роботы Python в [виртуальной среде](#). Приведенные ниже инструкции подходят для разработки и в виртуальной среде, и на локальном компьютере.

## Шаблоны

Установите требуемые пакеты, выполнив следующие команды:

```
pip install botbuilder-core  
pip install asyncio  
pip install aiohttp  
pip install cookiecutter==1.7.0
```

Последний пакет, `cookiecutter`, будет использоваться для создания программы-робота. Убедитесь, что она правильно установлена, выполнив `cookiecutter --help`.

## Создание бота

Чтобы создать программу-робот, перейдите в каталог, в котором вы хотите создать Bot, а затем выполните команду:

```
cookiecutter https://github.com/microsoft/BotBuilder-Samples/releases/download/Templates/echo.zip
```

Эта команда копирует все необходимые файлы из GitHub, чтобы создать эхо-робот на основе [шаблона Python Echo](#). Вам будет предложено ввести имя и описание бота. Присвойте имя Bot echo-Bot и задайте для него описание, **которое будет выводить ответ обратного пользователя**. как показано ниже:

```
bot_name [my_chat_bot]: echo-bot  
bot_description [Demonstrate the core capabilities of the Microsoft Bot Framework]:  
A bot that echoes back user response.
```

## Запуск бота

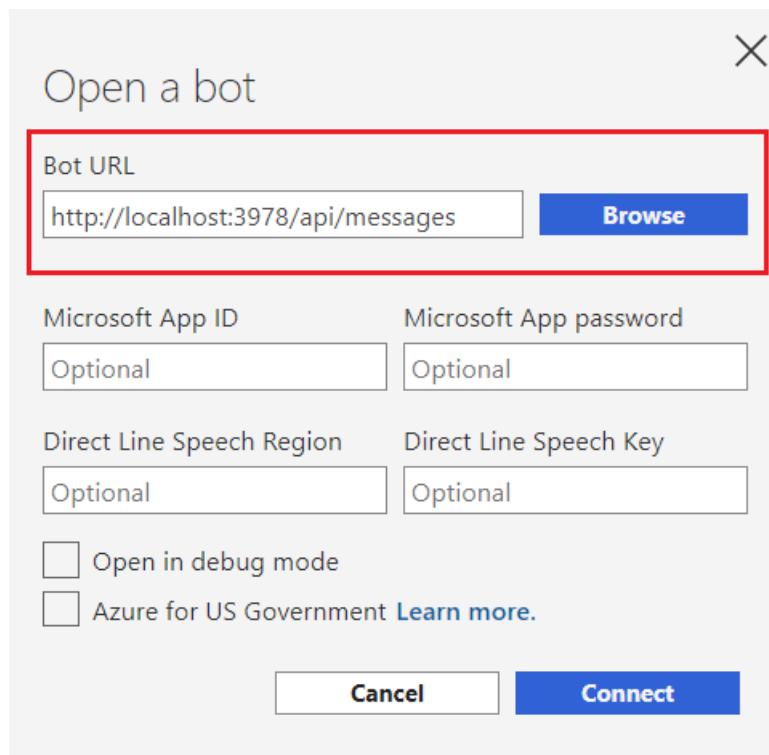
1. В окне терминала перейдите в папку echo-Bot , в которую вы сохранили робот. Выполните команду `pip install -r requirements.txt` , чтобы установить пакеты, требуемые для запуска бота.
2. После установки пакетов выполните `python app.py` для запуска бота. Вы поймете, что бот готов к тестированию, когда появится последняя строка, показанная на снимке экрана ниже.

```
D:\MyEchoBot>python app.py
===== [Running on http://localhost:3978] =====
(Press CTRL+C to quit)
```

Скопируйте последние четыре цифры в адресе в последней строке (обычно это 3978) для использования на следующем шаге. Теперь все готово для запуска эмулятора.

## Запуск эмулятора и подключение к боту

1. Установите Bot Framework Emulator.
2. Выберите **Открыть** Bot на вкладке **приветствия** эмулятора.
3. Введите локальный URL-адрес бота с указанием порта и пути /api/messages . Обычно этот адрес выглядит так: `http://localhost:3978/api/messages` .



4. В этом случае выберите **Подключиться**.

Отправьте сообщение в Bot, и Bot ответит обратно.

Hello and welcome!

Just now

hello

Just now

Echo: hello

Just now



Type your message



## Дополнительные ресурсы

Подключение к удаленно размещенному боту описано в документации по [туннелированию \(ngrok\)](#).

## Дальнейшие действия

[Развертывание бота в Azure](#)

# Создание бота с помощью службы Azure Bot

27.03.2021 • 8 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Служба Azure Bot предоставляет основные компоненты для создания ботов, включая пакет SDK Bot Framework для разработки ботов и службу ботов для подключения ботов к каналам. В этой статье показано, как создать робот с помощью пакета SDK для Bot Framework v4 и выбрать шаблон .NET или Node.js.

Существует два подхода к созданию программы-робота с помощью Azure.

1. **Веб-приложение.** Создайте робот и зарегистрируйте его в Azure с помощью веб-приложения, как показано в этой статье. Этот подход используется при разработке и размещении программы-робота в Azure.
2. **Регистрация каналов Bot.** Создавайте и разрабатывайте программу Bot локально и размещайте ее на платформе, отличной от Azure. При регистрации робота вы предоставляете веб-адрес, на котором размещена программа Bot. Вы по-прежнему можете разместить его в Azure. Выполните действия, описанные в статье о [регистрации каналов Bot](#).

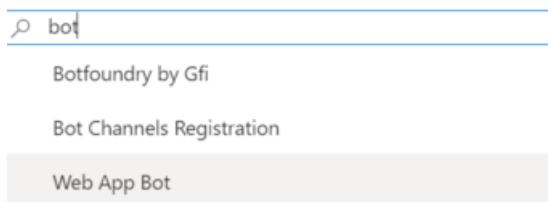
Создание бота с помощью службы Azure Bot и локальных средств — это независимые друг от друга параллельные процессы.

## Предварительные требования

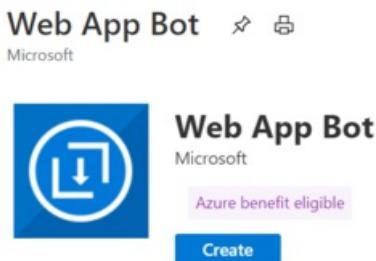
- Учетная запись [Azure](#)

### Создание службы ботов

1. Войдите на [портал Azure](#).
2. Щелкните ссылку **создать ресурс** в верхнем левом углу портал Azure.
3. В поле поиска введите *Bot* и в раскрывающемся списке выберите **веб-приложение Bot**.



4. На странице программы **веб-приложения "робот"** нажмите кнопку **создать**.



5. В форме Bot **веб-приложения** Укажите запрашиваемые сведения о программе Bot, как указано в таблице под изображением.

**Web App Bot**

Bot Service

\* Bot name [?](#)

\* Subscription

\* Resource group  
 [▼](#)  
[Create new](#)

\* Location [?](#)  
 [▼](#)

Pricing tier ([View full pricing details](#))  
 [▼](#)

\* App name [?](#)  
 [.azurewebsites.net](#)

\* Bot template [?](#) [>](#)  
**Basic Bot (C#)**

---

\* LUIS App location  
 [▼](#)

---

\* App service plan/Location [>](#)

---

Application Insights [?](#)  
 On  Off

\* Application Insights Location [?](#)  
 [▼](#)

---

Microsoft App ID and password [?](#) [>](#)  
[Auto create App ID and passw...](#)

---

[Create](#) [Automation options](#)

ПАРАМЕТР	РЕКОМЕНДУЕМОЕ ЗНАЧЕНИЕ	ОПИСАНИЕ
<b>Имя бота</b>	Отображаемое имя вашего бота	Отображаемое имя бота, которое показывается на каналах и в каталогах. Это имя можно изменить в любое время.
<b>Подписка</b>	Ваша подписка	Выберите подписку Azure, которую нужно использовать.
<b>Группа ресурсов</b>	myResourceGroup	Вы можете создать <a href="#">группу ресурсов</a> или выбрать имеющуюся.

ПАРАМЕТР	РЕКОМЕНДУЕМОЕ ЗНАЧЕНИЕ	ОПИСАНИЕ
<b>Расположение</b>	Расположение по умолчанию	Выберите географическое расположение для группы ресурсов. Вы можете выбрать любое расположение в списке, но лучше всего выбрать расположение, ближайшее к вашему клиенту. Расположение нельзя изменить после создания бота.
<b>Ценовая категория</b>	F0	Выберите ценовую категорию. Вы можете обновить ценовую категорию в любое время. Дополнительные сведения см. на странице <a href="#">цен на службу Bot</a> .
<b>Имя приложения</b>	Уникальное имя	Уникальный URL-адрес бота. Например, если вы используете для бота имя <i>myawesomebot</i> , его URL-адресом будет <a href="http://myawesomebot.azurewebsites.net">http://myawesomebot.azurewebsites.net</a> . В имени нужно использовать только буквы, цифры и символ подчеркивания. Допустимое число символов для этого поля: 35. Имя приложения нельзя изменить после создания бота.
<b>Bot template (Шаблон бота)</b>	Бот Echo	Выберите <b>SDK v4</b> (Пакет SDK версии 4). Выберите C# или Node.js для работы с этим кратким руководством, затем нажмите кнопку <b>Выбрать</b> .
<b>Расположение или план службы приложений</b>	План службы приложений	Выберите расположение <a href="#">плана службы приложений</a> . Вы можете выбрать любое расположение в списке, но обычно рекомендуется выбирать расположение, в котором находится служба бота.
<b>Учетные записи LUIS доступны только для шаблона Базового бота</b>	Имя ресурса LUIS в Azure	После <a href="#">переноса ресурсов LUIS в ресурс Azure</a> введите имя ресурса Azure, чтобы связать приложение LUIS с этим ресурсом Azure.

ПАРАМЕТР	РЕКОМЕНДУЕМОЕ ЗНАЧЕНИЕ	ОПИСАНИЕ
Application Insights	C	Решите, хотите ли вы <b>включить</b> или <b>выключить</b> Application Insights. Если вы выберете параметр <b>Вкл.</b> , необходимо также указать региональное расположение. Вы можете выбрать любое расположение в списке, но обычно рекомендуется выбирать расположение, в котором находится служба бота.
Microsoft App ID and password (Идентификатор и пароль приложения Майкрософт)	Auto create App ID and password (Автоматическое создание идентификатора и пароля приложения)	Используйте этот параметр, если вам нужно вручную ввести идентификатор и пароль приложения Майкрософт. В противном случае они будут созданы при создании бота. При создании регистрации приложения вручную для службы Bot убедитесь, что для поддерживаемых типов учетных записей заданы <b>учетные записи в любом каталоге Организации или учетных записях в любом каталоге Организации и личных учетных записях Майкрософт (например, Outlook.com, Xbox и т. д.)</b> .

6. Нажмите кнопку **Создать** для создания службы и развертывания бота в облаке. Это может занять несколько минут.

Убедитесь, что бот был развернут, проверив раздел **Уведомления**. Уведомление изменится с **Развертывание выполняется...** на **Развертывание выполнено**. Щелкните ссылку **Перейти к ресурсу**, чтобы открыть страницу ресурсов Bot.

После создания бота проверьте его в компоненте "Веб-чат".

## Тестирование бота

В разделе **Параметры** щелкните **проверить в веб-чате**. Служба Azure Bot загрузит элемент управления "Веб-чат" и подключится к боту.

The screenshot shows the Microsoft Bot Framework interface. On the left, there's a sidebar with various navigation options: Overview, Activity log, Access control (IAM), Tags, Settings (with sub-options like Bot profile, Configuration, Channels, Pricing, and Test in Web Chat), Encryption, Properties, Locks, Monitoring (with sub-options like Conversational analytics, Alerts, Metrics, Diagnostic settings, and Logs), Automation, and Tasks (preview). The 'Test in Web Chat' option is highlighted with a red box. The main area is titled 'Test' and features a large blue circular logo with a white 'L' shape and two small circles. Below the logo, the text 'Welcome to Bot Framework!' is displayed, followed by a message: 'Now that you have successfully run your bot, follow the links in this Adaptive Card to expand your knowledge of Bot Framework.' Three buttons are shown: 'Get an overview', 'Ask a question', and 'Learn how to deploy'. At the bottom, a message input field says 'What can I help you with today? Say something like "Book a flight from Paris to Berlin on March 31, 2021"' and shows a timestamp '2 minutes ago'. A message box at the bottom right contains the placeholder 'Type your message'.

Введите сообщение. Бот должен ответить.

## Скачать код

Можно скачать код, чтобы работать с ним локально.

- Перейдите в колонку **Обзор**.
- Выберите **загрузить исходный код** Bot на верхней панели инструментов или в нижней части панели.
- Следуйте инструкциям на экране, чтобы скачать код, а затем распакуйте папку.

При скачивании бота вы сможете включить в файл все параметры (с ключами и секретами) своего бота, без которых, возможно, он не будет работать. Если вы выберете **Да**, ключи будут содержаться в файле `appsettings.json` ИЛИ `.env`.

Search (Ctrl+ /) <> Download bot source code Refresh Move Delete

Overview Activity log Access control (IAM) Tags

Settings Bot profile Configuration Channels Speech priming Pricing Test in Web Chat Encryption Properties Locks

Monitoring Conversational analytics

Automation Tasks (preview) Export template

Essentials

Resource group (change)  
echo-test

Subscription (change)

Subscription ID

Tags (change)  
Click here to add tags

Bot Service pricing tier F0

Messaging endpoint https://echo-test.000.azurewebsites.net

Linked app service echo-test

Build enterprise-grade conversational AI experiences with the Bot Service

Create bots with sophisticated abilities to speak, listen, and understand, connect to popular channels, and maintain ownership and control of your data. [Learn more](#)

Get set up with local development

Download your bot's source code, start building locally using your preferred dev environment, and add intelligence using Microsoft's services.

Test and refine your bot

Refined and debug locally with [Visual Studio](#), or [test your bot online in Web Chat](#). Learn more about [testing](#) and [debugging](#).

Download bot source code

Test in Web Chat

## Дополнительные сведения

### Регистрация приложений вручную

Ручная регистрация может потребоваться в следующих случаях:

- Вы не можете выполнить регистрацию в своей организации, и требуется третья сторона, чтобы создать идентификатор приложения для бота, сборка которого выполняется.
- Вам нужно вручную создать свой идентификатор приложения (и пароль).

Подробные сведения см. в разделе [Регистрация приложения](#).

## Дальнейшие действия

После скачивания кода вы можете продолжить разработку бота на своем локальном компьютере. Когда вы протестируете Bot и готовы отправить код Bot в портал Azure, следуйте инструкциям в разделе [Настройка развертывания континуос](#), чтобы автоматически обновлять код после внесения изменений.

# Учебник. Создание базового робота

27.03.2021 • 12 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этом учебнике описывается, как создать базовый робот с пакетом SDK для Bot Framework. Если вы уже создали базовый робот и хотите, чтобы он выполнялся локально, ознакомьтесь с [руководством по развертыванию базовой ленты](#).

Вы научитесь:

- Создание базового бота
- Запуск программы-робота на локальном компьютере
- Тестирование с помощью эмулятора Bot Framework

## Предварительные требования

- [C#](#)
- [JavaScript](#)
- [Python](#)
- [Среда выполнения ASP.NET Core 3,1](#)
- [Bot Framework Emulator.](#)
- Знание [ASP.NET Core](#) и [асинхронного программирования в C#](#)

### Шаблоны

- [Visual Studio](#)
- [Visual Studio Code/Командная строка](#)
- [Visual Studio 2019](#) или более поздней версии.
- [Шаблоны пакета SDK для Bot Framework версии 4 для Visual Studio](#)

Чтобы добавить шаблоны Bot в Visual Studio, скачайте и установите [Шаблоны пакета SDK для Bot Framework версии 4 для Visual Studio VSIX](#).

#### NOTE

Пакет [VSIX](#) включает версии .net Core 2,1 и .net Core 3,1 шаблонов C#. При создании ботов в Visual Studio 2019 следует использовать шаблоны .NET Core 3.1. В текущих примерах ботов используются шаблоны .NET Core 3.1. Примеры, использующие шаблоны .NET Core 2.1, можно найти в ветви [4.7-archive](#) репозитория BotBuilder-Samples. Сведения о развертывании .NET Core 3,1 программы-роботы в Azure см. в статье [развертывание программы Bot в Azure](#).

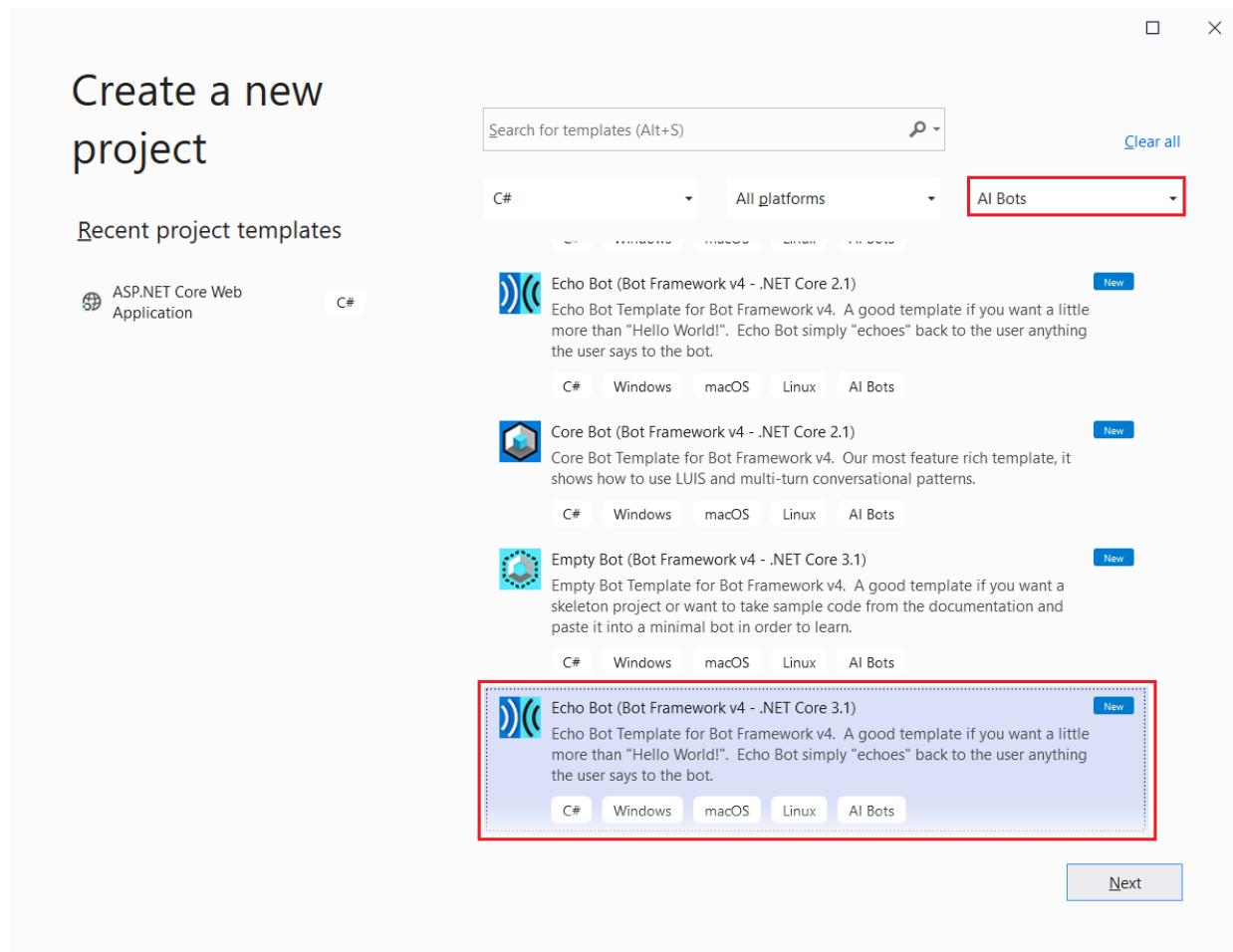
## Создание бота

- [C#](#)
- [JavaScript](#)
- [Python](#)
- [Visual Studio](#)

- [Visual Studio Code](#)
- [Командная строка](#)

## Сборка с помощью Visual Studio

В Visual Studio создайте проект бота с использованием шаблона **эхо-бота на основе Bot Framework версии 4 и .NET Core 3.1**. Выберите тип проекта **AI Bots** (Боты (ИИ)), чтобы отображались только шаблоны ботов.



Благодаря шаблону проект содержит весь код, необходимый для создания бота в рамках этого краткого руководства. Для тестирования бота не требуется дополнительный код.

### NOTE

При создании *основного* робота требуется модель языка Luis. Вы можете создать языковую модель по адресу [Luis.AI](#). Создав модель, обновите файл конфигурации.

## Запуск бота

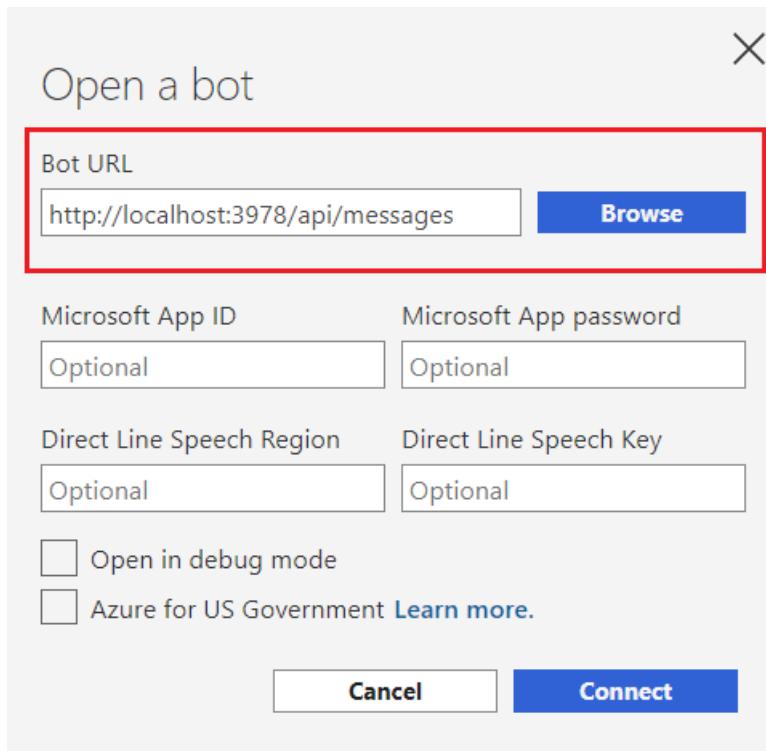
- [C#](#)
- [JavaScript](#)
- [Python](#)
- [Visual Studio](#)
- [Visual Studio Code](#)
- [Командная строка](#)

В Visual Studio запустите проект. Visual Studio создаст приложение, развернет его на узле localhost и запустит веб-браузер для отображения страницы приложения `default.htm`. На этом этапе бот

выполняется локально, используя порт 3978.

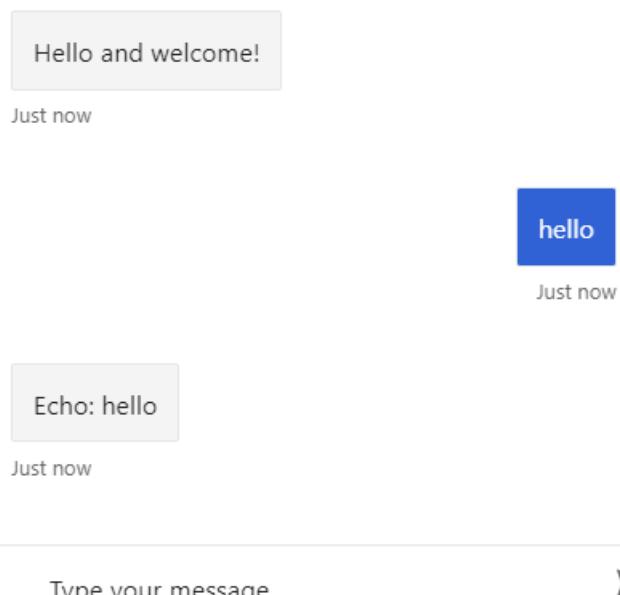
## Запуск эмулятора и подключение к боту

1. Установите Bot Framework Emulator.
2. Выберите **Открыть Bot** на вкладке **приветствия** эмулятора.
3. Введите локальный URL-адрес бота с указанием порта и пути /api/messages. Обычно этот адрес выглядит так: `http://localhost:3978/api/messages`.



4. В этом случае выберите **Подключиться**.

Отправьте сообщение в Bot, и Bot ответит обратно.



## Дальнейшие действия

[Руководство. Развёртывание базового робота](#)

# Руководство. Развертывание базового робота

27.03.2021 • 24 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этом руководстве описывается, как развернуть базовый робот в Azure. В нем объясняется, как подготовить программу Bot к развертыванию, развернуть программу Bot в Azure и протестировать робот с помощью веб-разговора. Это было бы полезно, если вы прочитали эту статью перед выполнением действий, чтобы полностью понять, что участвует в развертывании программы-робота.

- Если вы еще не создали базовый робот, ознакомьтесь с руководством по [созданию простой ленты](#).
- Если у вас еще нет подписки Azure, [создайте бесплатную учетную запись](#), прежде чем начинать работу.

Вы научитесь:

- Подготовка базовой программы-робота к развертыванию
- Разворачивание бота в Azure
- Тестирование с помощью веб-разговора

## IMPORTANT

Убедитесь, что используется последняя версия [Azure CLI](#). Если вы используете версию Azure CLI, предшествующую версии [2.2.0](#), возникнет ошибка из-за нерекомендуемых команд CLI. Кроме того, развертывание с помощью Azure CLI, описанное в этой статье, не стоит сочетать с развертыванием на портале Azure.

## Предварительные требования

- Подписка на [Microsoft Azure](#).
- Бот C#, JavaScript, TypeScript или Python, который разработан на локальном компьютере.
- Последняя версия [Azure CLI](#).
- Навыки работы с [Azure CLI и шаблонами ARM](#).

## Подготовка к развертыванию

В этой статье предполагается, что у вас есть бот, готовый к развертыванию. Сведения о том, как создать простой робот Echo, см. в разделе [руководство. Создание базовой ленты Bot](#). Вы также можете использовать один из примеров, приведенных в соответствующем репозитории [Bot Framework](#).

- [C#](#)
- [JavaScript/TypeScript](#)
- [Python](#)

Если вы развертываете бот на C#, убедитесь, что он [создан в режиме выпуска](#). В Visual Studio для конфигурации решения укажите значение **Release**. Затем для решения выполните повторную сборку с очисткой, прежде чем продолжить. Развертывание может завершиться ошибкой, если для конфигурации решения задано значение **Отладка**.

При создании программы-робота с помощью [шаблона Visual Studio](#) созданный исходный код включает в

себя `DeploymentTemplates` папку, содержащую шаблоны ARM. В описанном здесь процессе развертывания используется один из шаблонов ARM для подготовки необходимых для бота ресурсов Azure с помощью Azure CLI.

С появлением пакета SDK Bot Framework 4.3 *не рекомендуется* использовать файл `.bot`. Вместо этого мы используем `appsettings.json` файл конфигурации для управления ресурсами Bot. Сведения о переносе параметров из файла `.bot` в файл конфигурации см. в разделе [Управление ресурсами Bot](#).

#### NOTE

Пакет [VSIX](#) включает версии .NET Core 2.1 и .NET Core 3.1 шаблонов C#. При создании ботов в Visual Studio 2019 следует использовать шаблоны .NET Core 3.1. В текущих примерах ботов используются шаблоны .NET Core 3.1. Примеры, использующие шаблоны .NET Core 2.1, можно найти в ветви [4.7-archive](#) репозитория BotBuilder-Samples. Сведения о развертывании .NET Core 3.1 программы-роботы в Azure см. в статье [развертывание программы Bot в Azure](#).

## 1. Вход в Azure

Созданный и протестированный локально бот можно развернуть в Azure. Откройте командную строку, чтобы войти на портал Azure.

```
az login
```

Она откроет окно браузера с интерфейсом для входа.

#### NOTE

При развертывании бота в облаке, отличном от Azure, например US Gov, необходимо выполнить `az cloud set --name <name-of-cloud>` перед `az login`, где `<name-of-cloud>` обозначает имя зарегистрированного облака, например `AzureUSGovernment`. Если вы хотите вернуться в общедоступное облако, можно запустить `az cloud set --name AzureCloud`.

## 2. Настройка подписки

Укажите подписку, которая будет использоваться по умолчанию.

```
az account set --subscription "<azure-subscription>"
```

Если вы не уверены, какую подписку выбрать для развертывания бота, просмотрите список подписок в учетной записи с помощью команды `az account list`.

## 3. Регистрация приложения

На этом шаге вы создадите регистрацию приложения Azure, что обеспечит следующие возможности.

- Взаимодействие пользователя с ботом через несколько каналов, например *Web Chat*.
- Определение *параметров подключения OAuth* для проверки подлинности пользователя и создания *маркера*, с помощью которого бот будет обращаться к защищенным ресурсам от имени пользователя.

### 3.1 Создание регистрации приложения Azure

Чтобы создать регистрацию приложения Azure, выполните приведенную ниже команду.

```
az ad app create --display-name "displayName" --password "AtLeastSixteenCharacters_0" --available-to-other-tenants
```

ПАРАМЕТР	ОПИСАНИЕ
display-name	Отображаемое имя приложения. Оно указывается на портале Azure в общем списке ресурсов и в той группе ресурсов, в которую включено это приложение.
password	Пароль для приложения, также известный как <b>секрет клиента</b> . Это тот пароль, который вы создали для этого ресурса. Он должен содержать не менее 16 символов с минимум одной буквой в верхнем или нижнем регистре и минимум одним специальным символом.
available-to-other-tenants	Указывает, что приложение может использоваться из любого клиента Azure AD. Включите этот параметр, чтобы разрешить боту работать с каналами службы Azure Bot.

### 3.2 Запись значений appId и appSecret

После выполнения команды на предыдущем шаге (3.1) необходимо будет записать два значения:

- `password`, созданное на предыдущем шаге;
- `appId`, которое можно найти в выходных данных JSON.

Скопируйте и сохраните значения `appId` и `password`. Они потребуются на шаге развертывания ARM для присвоения значений параметрам `appId` и `appSecret` соответственно.

## 4. Развертывание с использованием шаблона ARM

При создании службы приложения бота можно развернуть бот в новой или существующей группе ресурсов. Для этого можно использовать [шаблон Azure Resource Manager \(ARM\)](#). Шаблон ARM — это JSON-файл, который декларативно определяет один или несколько ресурсов Azure и устанавливает зависимости между развернутыми ресурсами. Убедитесь, что у вас есть правильный путь к каталогу шаблонов развертывания ARM проекта **деплойменттемплатес**. Он необходим для присвоения значения файлу шаблона. Выберите любой вариант, который вам подходит.

- [Развертывание с помощью шаблона ARM в новой группе ресурсов](#).
- [Развертывание с помощью шаблона ARM в существующей группе ресурсов](#).

#### IMPORTANT

Программы-роботы Python невозможно развернуть в группе ресурсов, содержащей службы Windows или программы-роботы. Несколько программы-роботы Python можно развернуть в одной группе ресурсов, но в другой группе ресурсов необходимо создать другие службы (LUIS, QnA и т. д.).

#### Развертывание с помощью шаблона ARM в новой группе ресурсов

На этом шаге вы создадите службу приложения для бота, то есть определите этап развертывания для бота. Вы примените для этого шаблон ARM, создадите план обслуживания и группу ресурсов. Выполните следующую команду Azure CLI, чтобы запустить развертывание в области подписки из локального файла шаблона.

#### TIP

Используйте шаблон ARM для новой группы ресурсов, `template-with-new-rg.json`.

```
az deployment sub create --template-file "<path-to-template-with-new-rg.json>" --location <region-location-name> --parameters appId=<app-id-from-previous-step> appSecret=<password-from-previous-step> botId=<id or bot-app-service-name> botSku=F0 newAppServicePlanName=<new-service-plan-name> newWebAppName=<bot-app-service-name> groupName=<new-group-name> groupLocation=<region-location-name> newAppServicePlanLocation=<region-location-name>" --name "<bot-app-service-name>"
```

#### NOTE

Этот шаг может занять несколько минут.

ПАРАМЕТР	ОПИСАНИЕ
name	Имя развертывания.
template-file	Путь к шаблону ARM. Обычно файл <code>template-with-new-rg.json</code> размещается в папке <code>deploymentTemplates</code> проекта бота. Это путь к существующему файлу шаблона. Можно указать абсолютный или относительный путь к текущему каталогу. Все шаблоны бота создают файлы шаблонов ARM.
location	Расположение. Значения из <code>az account list-locations</code> . Расположение по умолчанию можно настроить с помощью <code>az configure --defaults location=&lt;location&gt;</code> .
параметры	Параметры развертывания в формате списка пар "ключ — значение". Введите следующие значения параметров:

- `appId` — значение идентификатора приложения из выходных данных JSON, созданных на шаге [Создание регистрации приложения](#).
- `appSecret` — пароль, указанный на шаге [Создание регистрации приложения](#).
- `botId` — имя создаваемого ресурса регистрации канала бота. Оно должно быть глобально уникальным. Оно используется как неизменяемый идентификатор бота и сохраняется в качестве отображаемого имени бота, но это значение вы можете изменить.
- `botSku` — ценовая категория, где возможны значения F0 (Бесплатный) или S1 (Стандартный).
- `newAppServicePlanName` — имя нового плана службы приложений.
- `newWebAppName` — имя службы приложений для бота.
- `groupName` — имя новой группы ресурсов.
- `groupLocation` — расположение группы ресурсов Azure.
- `newAppServicePlanLocation` — расположение плана службы приложений.

#### Развертывание с помощью шаблона ARM в существующей группе ресурсов

На этом шаге вы создадите службу приложения для бота, то есть определите этап развертывания для бота. Если вы используете существующую группу ресурсов, можно выбрать существующий план службы приложений или создать новый. Выберите любой вариант, который вам подходит.

- [Вариант 1. Существующий план Службы приложений](#)
- [Вариант 2. Новый план Службы приложений](#)

## NOTE

Этот шаг может занять несколько минут.

### Вариант 1. Существующий план Службы приложений

В этом варианте мы используем существующий план Службы приложений, но при этом создаем новое веб-приложение и новую регистрацию каналов бота.

Эта команда ниже определяет идентификатор и отображаемое имя бота. Параметр `botId` должен быть глобально уникальным. Он используется как неизменяемый идентификатор бота. Отображаемое имя бота является изменяемым.

## TIP

Используйте шаблон ARM для существующей группы ресурсов, [template-with-preexisting-rg.json](#).

```
az deployment group create --resource-group "<name-of-resource-group>" --template-file "<path-to-template-with-preexisting-rg.json>" --parameters appId=<app-id-from-previous-step> appSecret=<password-from-previous-step> botId=<id or bot-app-service-name> newWebAppName=<bot-app-service-name> existingAppServicePlan=<name-of-app-service-plan> appServicePlanLocation=<region-location-name> --name "<bot-app-service-name>"
```

### Вариант 2. Новый план Службы приложений

В этом варианте мы создаем план Службы приложений, веб-приложение и регистрацию каналов бота.

## TIP

Используйте шаблон ARM для существующей группы ресурсов, [template-with-preexisting-rg.json](#).

```
az deployment group create --resource-group "<name-of-resource-group>" --template-file "<path-to-template-with-preexisting-rg.json>" --parameters appId=<app-id-from-previous-step> appSecret=<password-from-previous-step> botId=<id or bot-app-service-name> newWebAppName=<bot-app-service-name> newAppServicePlanName=<name-of-app-service-plan> newAppServicePlanLocation=<region-location-name> --name "<bot-app-service-name>"
```

ПАРАМЕТР	ОПИСАНИЕ
name	Имя развертывания.
resource-group	Имя группы ресурсов Azure.
template-file	Путь к шаблону ARM. Обычно файл <code>template-with-preexisting-rg.json</code> размещается в папке <code>deploymentTemplates</code> проекта. Это путь к существующему файлу шаблона. Можно указать абсолютный или относительный путь к текущему каталогу. Все шаблоны бота создают файлы шаблонов ARM.
location	Расположение. Значения из <code>az account list-locations</code> . Расположение по умолчанию можно настроить с помощью <code>az configure --defaults location=&lt;location&gt;</code> .

ПАРАМЕТР	ОПИСАНИЕ
параметры	Параметры развертывания в формате списка пар "ключ — значение". Введите следующие значения параметров:

- `appId` — значение `appId`, созданное на шаге [3.1 Создание регистрации приложения](#).
- `appSecret` — пароль, указанный на шаге [3.1 Создание регистрации приложения](#).
- `botId` — имя создаваемого ресурса регистрации канала бота. Оно должно быть глобально уникальным. Оно используется как неизменяемый идентификатор бота и сохраняется в качестве отображаемого имени бота, но это значение вы можете изменить.
- `newWebAppName` — имя службы приложений для бота.
- `newAppServicePlanName` — имя создаваемого ресурса плана службы приложений.
- `appServicePlanLocation` — расположение плана службы приложений.

## 5. Подготовка кода к развертыванию

В следующих нескольких шагах создается файл, зависимый от языка, ZIP (сжимается) файлы проекта и отправляется в Azure. Для выполнения этих действий *Папка проекта Bot* является корневой папкой для Bot.

- Для ботов на C# это папка, в которой расположен CSPROJ-файл.
- Для ботов на JavaScript это папка, в которой расположен файл app.js или index.js.
- Для ботов на TypeScript это папка, которая содержит папку *src* (с файлами bot.ts и index.ts).
- Для ботов Python это папка, в которой расположен файл app.py.

### Получение или создание файлов, необходимых для IIS или Kudu

Перед развертыванием бота нужно подготовить файлы проекта.

Убедитесь, что вы находитесь в папке проекта программы-робота. Затем подготовьте код Bot для развертывания.

- [C#](#)
- [JavaScript](#)
- [TypeScript](#)
- [Python](#)

```
az bot prepare-deploy --lang Csharp --code-dir "." --proj-file-path "MyBot.csproj"
```

Необходимо указать путь к CSPROJ-файлу относительно папки `--code-dir`. Для этого можно применить аргумент `--proj-file-path`. Эта команда разрешит аргументы `--code-dir` и `--proj-file-path` в значение `./MyBot.csproj`.

Эта команда создаст файл `.deployment` в папке проекта бота.

### Архивация каталога кода вручную

Если для развертывания кода бота используются ненастроенные интерфейсы [API развертывания ZIP-файлов](#), Web App или Kudu демонстрирует следующее поведение.

*Kudu по умолчанию предполагает, что развертывание из ZIP-файлов готово к выполнению и не требует дополнительных шагов сборки, таких как npm install или dotnet restore/dotnet publish.*

Важно включить весь код сборки и все необходимые зависимости в развертываемый ZIP-файл, иначе бот не будет работать должным образом.

## IMPORTANT

Прежде чем архивировать файлы проекта, убедитесь, что вы зашли в папку проекта бота.

- Для ботов на C# это папка, в которой расположен CSPROJ-файл.
- Для ботов на JavaScript это папка, в которой расположен файл app.js или index.js.
- Для ботов на TypeScript это папка, которая содержит папку *src* (с файлами bot.ts и index.ts).
- Для ботов Python это папка, в которой расположен файл app.py.

Перед выполнением команды для создания ZIP-файла убедитесь, что **в** папке проекта выбраны все файлы и папки. В папке проекта будет создан один ZIP-файл. Если расположение корневой папки выбрано неверно, **бот не сможет запуститься на портале Azure**.

## Развертывание Bot в Azure

Теперь мы готовы развернуть код в виде веб-приложения Azure. Запустите следующую команду из командной строки, чтобы выполнить развертывание с помощью принудительного развертывания в Kudu из ZIP-файла веб-приложения.

```
az webapp deployment source config-zip --resource-group "<resource-group-name>" --name "<name-of-web-app>" -src "<project-zip-path>"
```

ПАРАМЕТР	ОПИСАНИЕ
resource-group	Имя группы ресурсов Azure, которая содержит бота. (Это будет группа ресурсов, которую вы использовали или создали при регистрации приложения для бота.)
name	Имя веб-приложения, которое вы использовали ранее.
src	Путь к созданному ранее ZIP-файлу проекта.

## NOTE

Этот шаг может занять несколько минут. Кроме того, после завершения развертывания бот станет доступным для тестирования через несколько минут.

## Тестирование в веб-чате

1. В браузере перейдите на [портал Azure](#).
2. На панели слева щелкните **Группы ресурсов**.
3. На панели справа найдите свою группу.
4. Щелкните имя группы.
5. Щелкните ссылку регистрации канала бота.
6. В области **Bot Channels Registration** (Регистрация канала бота) щелкните **Test in Web Chat** (Протестировать в Web Chat). Или на панели справа щелкните поле **Тест**.

См. сведения о регистрации каналов бота в руководстве по [регистрации бота с помощью службы Azure Bot](#).

## Дополнительные ресурсы

При развертывании бота на портале Azure обычно создаются следующие ресурсы.

РЕСУРСЫ	ОПИСАНИЕ
Бот веб-приложения	Бот в службе Azure Bot, который развернут в Службе приложений Azure.
Служба приложений	Позволяет создавать и размещать веб-приложения.
План обслуживания приложения	Определяет набор вычислительных ресурсов, на которых выполняется веб-приложение.

При создании бота через портал Azure вы сможете подготовить дополнительные ресурсы, например [Application Insights для телеметрии](#).

Для просмотра документации по команде `az bot` см. этот раздел [справки](#).

Если вы еще не знакомы с концепцией группы ресурсов в Azure, см. раздел со списком [терминов](#).

## Дальнейшие действия

[Использование QnA Maker в боте для ответов на вопросы](#).

# Руководство по Использование QnA Maker в работе для ответов на вопросы.

27.03.2021 • 15 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Вы можете использовать службу QnA Maker, чтобы создать базу знаний для добавления поддержки вопросов и ответов в программу Bot. Созданная база знаний заполняется готовыми вопросами и ответами.

В этом руководстве описано следующее:

- Создание службы QnA Maker и базы знаний
- Добавление сведений о базе знаний в файл конфигурации
- Обновление бота для запросов по базе знаний
- Повторная публикация бота

Если у вас еще нет подписки Azure, [создайте бесплатную учетную запись](#), прежде чем начинать работу.

## Предварительные требования

- Бот, созданный при работе с [предыдущим руководством](#). В программу-робот добавляется функция вопросов и ответов.
- Вам будет проще, если вы уже знакомы со службой [QnA Maker](#). Вы будете использовать [портал QnA Maker](#) для создания, обучения и публикации базы знаний для использования с программой-роботом.
- Опыт [создания бота QnA](#) с помощью службы Azure Bot.

Кроме того, уже должны быть выполнены [Предварительные требования для предыдущего руководства](#).

## Создание службы QnA Maker и базы знаний

Вы импортируете существующее определение базы знаний из примера QnA Maker в репозитории [BotBuilder-Samples](#) .

1. Клонируйте или скопируйте на компьютер репозиторий примеров.
2. Войдите на [портал QnA Maker](#) с учетными данными Azure.
3. Выберите **создать базу знаний** на портале QnA Maker.
  - a. Если потребуется, создайте службу QnA. (Можно использовать существующую службу QnA Maker или создать новую специально для работы с этим руководством.) Более подробные инструкции по QnA Maker см. в руководствах по [созданию службы QnA Maker](#) и [созданию, подготовке и публикации базы знаний QnA Maker](#).
  - b. Подключите службу QnA Maker к базе знаний.
  - c. Присвойте имя базе знаний.
  - d. Чтобы заполнить базу знаний, используйте файл **smartlightFAQ.tsv** из репозитория Samples. Если вы уже скачали примеры, отправьте файл smartLightFAQ.tsv со своего компьютера.
  - e. Щелкните **создать** базу знаний, чтобы создать ее.
4. Выберите **сохранить и обучить**.
5. Выберите **опубликовать**, чтобы опубликовать базу знаний.

После публикации QnA Maker приложения выберите **Параметры** и прокрутите вниз до пункта *сведения о развертывании*. Скопируйте следующие значения из запроса HTTP-примера *POST*.

```
POST /knowledgebases/<knowledge-base-id>/generateAnswer
Host: <your-hostname> // NOTE - this is a URL ending in /qnamaker.
Authorization: EndpointKey <qna-maker-resource-key>
```

Полная строка URL-адреса для имени узла будет выглядеть так: "https://<knowledge-base-name>.Azure.NET/qnamaker".

Эти значения будут использоваться в файле конфигурации Bot на следующем шаге.

Теперь база знаний готова для использования в боте.

## Добавление сведений о базе знаний в бота

Начиная с бот-платформы версии 4.3, файлы с расширением .bot больше не предоставляются в Azure для скачивания вместе с исходным кодом бота. Следуйте дальнейшим инструкциям, чтобы подключить свой бот на C#, JavaScript или Python к базе знаний.

- [C#](#)
- [JavaScript](#)
- [Python](#)

Добавьте следующие значения в **appsetting.json** файле:

```
{
  "MicrosoftAppId": "",
  "MicrosoftAppPassword": "",
  "ScmType": "None",

  "QnAKnowledgebaseId": "knowledge-base-id",
  "QnAAuthKey": "qna-maker-resource-key",
  "QnAEndpointHostName": "your-hostname" // This is a URL ending in /qnamaker
}
```

ПОЛЕ	ЗНАЧЕНИЕ
QnAKnowledgebaseId	Идентификатор базы знаний, автоматически созданный на портале QnA Maker.
QnAAuthKey (QnAEndpointKey на Python)	Ключ конечной точки, автоматически созданный на портале QnA Maker.
QnAEndpointHostName	URL-адрес узла, созданный на портале QnA Maker. Используйте полный формат URL-адреса, начиная с <code>https://</code> и заканчивая <code>/qnamaker</code> . Стока полного URL-адреса будет выглядеть так: "https://<knowledge-base-name>.Azure.NET/qnamaker".

Сохраните изменения.

## Обновление бота для запросов по базе знаний

Обновите код инициализации, чтобы загрузить данные службы для вашей базы знаний.

- [C#](#)

- [JavaScript](#)
- [Python](#)

1. Добавьте в проект пакет NuGet `Microsoft.Bot.Builder.AI.QnA`.

Это можно сделать с помощью диспетчера пакетов NuGet или командной строки:

```
dotnet add package Microsoft.Bot.Builder.AI.QnA
```

Дополнительные сведения о NuGet см. в [документации по NuGet](#).

2. В файле `Startup.cs` добавьте следующую ссылку на пространство имен.

`Startup.cs`

```
using Microsoft.Bot.Builder.AI.QnA;
```

3. Измените `ConfigureServices` метод в файле `Startup.cs`, чтобы создать `QnAMakerEndpoint` объект, который подключается к базе знаний, определенной в `appsettings.js` файла.

`Startup.cs`

```
// Create QnA Maker endpoint as a singleton
services.AddSingleton(new QnAMakerEndpoint
{
    KnowledgeBaseId = Configuration.GetValue<string>("QnAKnowledgebaseId"),
    EndpointKey = Configuration.GetValue<string>("QnAAuthKey"),
    Host = Configuration.GetValue<string>("QnAEndpointHostName")
});
```

4. В файле `echoBot.cs` добавьте следующие ссылки на пространства имен.

`Bots\EchoBot.cs`

```
using System.Linq;
using Microsoft.Bot.Builder.AI.QnA;
```

5. Добавьте `EchoBotQnA` свойство и добавьте конструктор для его инициализации.

`EchoBot.cs`

```
public QnAMaker EchoBotQnA { get; private set; }

public EchoBot(QnAMakerEndpoint endpoint)
{
    // connects to QnA Maker endpoint for each turn
    EchoBotQnA = new QnAMaker(endpoint);
}
```

6. Добавьте `AccessQnAMaker` метод:

`EchoBot.cs`

```
private async Task AccessQnAMaker(ITurnContext<IMessageActivity> turnContext, CancellationToken cancellationToken)
{
    var results = await EchoBotQnA.GetAnswersAsync(turnContext);
    if (results.Any())
    {
        await turnContext.SendActivityAsync(MessageFactory.Text("QnA Maker Returned: " +
    results.First().Answer), cancellationToken);
    }
    else
    {
        await turnContext.SendActivityAsync(MessageFactory.Text("Sorry, could not find an answer in the
knowledge base."), cancellationToken);
    }
}
```

7. Обновите `OnMessageActivityAsync` метод, чтобы вызвать новый `AccessQnAMaker` метод.

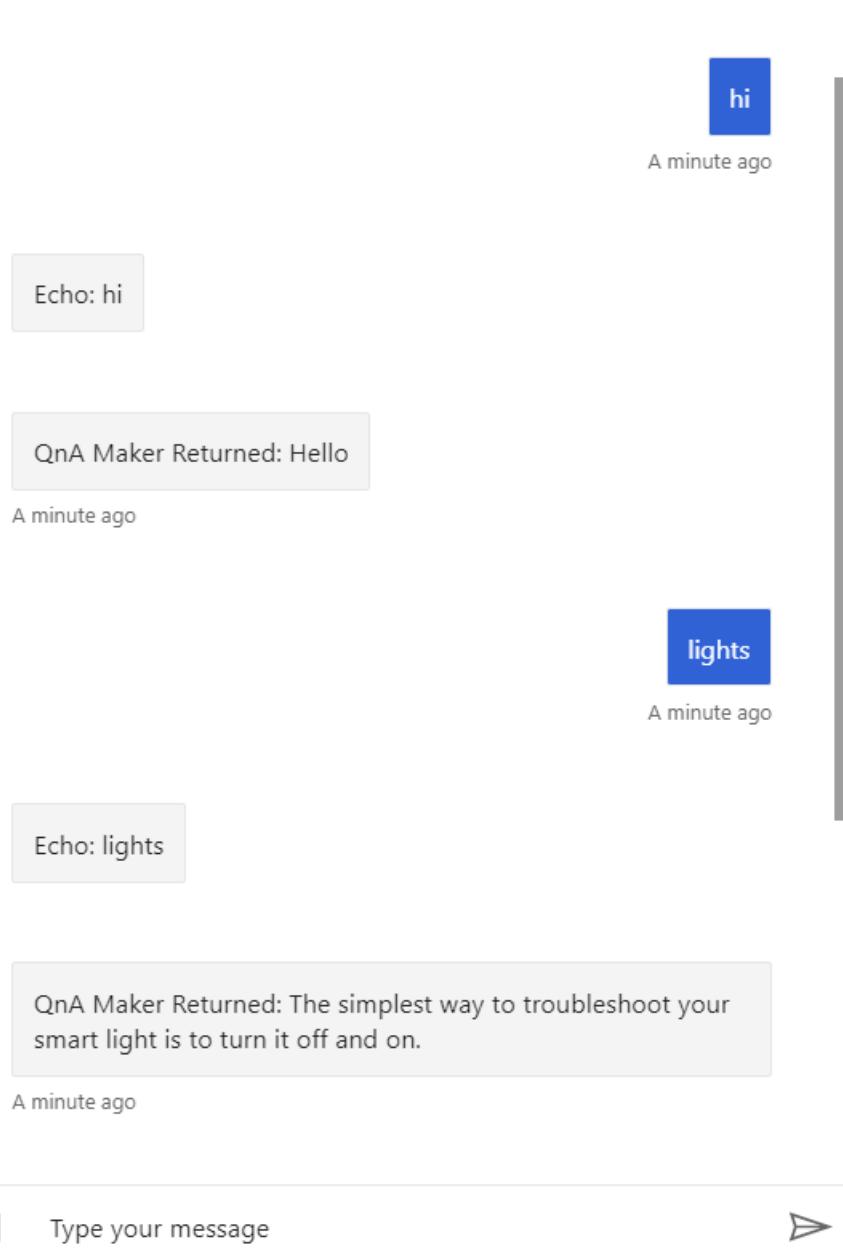
### EchoBot.cs

```
protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    var replyText = $"Echo: {turnContext.Activity.Text}";
    await turnContext.SendActivityAsync(MessageFactory.Text(replyText, replyText),
cancellationToken);

    await AccessQnAMaker(turnContext, cancellationToken);
}
```

## Локальная проверка бота

На этом этапе бот уже может отвечать на некоторые вопросы. Запустите бот на локальном компьютере и откройте его в эмуляторе.



## Повторная публикация бота

Теперь можно повторно опубликовать бота в Azure. Для этого необходимо заархивировать папку проекта и выполнить команду развертывания. Дополнительные сведения см. в статье [развертывание программы-Bot](#).

### Архивация папки проекта

Если для развертывания кода бота используются ненастроенные интерфейсы [API развертывания ZIP-файлов](#), Web App или Kudu демонстрирует следующее поведение.

*Kudu по умолчанию предполагает, что развертывание из ZIP-файлов готово к выполнению и не требует дополнительных шагов сборки, таких как `npm install` или `dotnet restore/dotnet publish`.*

Важно включить весь код сборки и все необходимые зависимости в развертываемый ZIP-файл, иначе бот не будет работать должным образом.

## IMPORTANT

Прежде чем архивировать файлы проекта, убедитесь, что вы зашли в папку проекта бота.

- Для ботов на C# это папка, в которой расположен CSPROJ-файл.
- Для ботов на JavaScript это папка, в которой расположен файл app.js или index.js.
- Для ботов на TypeScript это папка, которая содержит папку *src* (с файлами bot.ts и index.ts).
- Для ботов Python это папка, в которой расположен файл app.py.

Перед выполнением команды для создания ZIP-файла убедитесь, что **в** папке проекта выбраны все файлы и папки. В папке проекта будет создан один ZIP-файл. Если расположение корневой папки выбрано неверно, **бот не сможет запуститься на портале Azure**.

## Развертывание кода в Azure

### TIP

Если срок действия маркера сеанса истек, запустите его `az login` снова. Если вы не используете подписку по умолчанию, также сбросьте свою подпись.

Теперь мы готовы развернуть код в виде веб-приложения Azure. Запустите следующую команду из командной строки, чтобы выполнить развертывание с помощью принудительного развертывания в Kudu из ZIP-файла веб-приложения.

```
az webapp deployment source config-zip --resource-group "<resource-group-name>" --name "<name-of-web-app>" --src "<project-zip-path>"
```

ПАРАМЕТР	ОПИСАНИЕ
resource-group	Имя группы ресурсов Azure, которая содержит бота. (Это будет группа ресурсов, которую вы использовали или создали при регистрации приложения для бота.)
name	Имя веб-приложения, которое вы использовали ранее.
src	Путь к созданному ранее ZIP-файлу проекта.

### NOTE

Этот шаг может занять несколько минут. Кроме того, после завершения развертывания бот станет доступным для тестирования через несколько минут.

Если вы не собираетесь использовать это приложение в дальнейшем, удалите все связанные ресурсы, выполнив следующие действия.

1. На портале Azure откройте группу ресурсов бота.
2. Выберите **Удалить группу ресурсов**, чтобы удалить группу и все содержащиеся в ней ресурсы.
3. Введите имя группы ресурсов на панели подтверждения, а затем выберите **Удалить**.

## Дальнейшие действия

См. подробнее о сведения о том, как добавлять в бот новые функции, в инструкциях по **отправке и получению текстовых сообщений** и других руководствах по разработке.

Отправка и получение текстовых сообщений

# Базовый план безопасности Azure для службы Azure Bot

27.03.2021 • 66 minutes to read • [Edit Online](#)

Этот базовый план безопасности применяет рекомендации из [контрольного показателя безопасности Azure версии 2,0](#) к службе Microsoft Azure Bot. Azure Security Benchmark содержит рекомендации по обеспечению безопасности облачных решений в Azure. Содержимое группируется по **элементам управления безопасностью**, определенным в ходе тестирования системы безопасности Azure, и связанным рекомендациям, применимым к службе Azure Bot. **Элементы управления**, неприменимые к службе Azure Bot, были исключены.

Чтобы узнать, как служба Azure Bot полностью сопоставляется с тестовым показателем безопасности Azure, ознакомьтесь с [полным базовым файлом сопоставления безопасности службы Azure Bot](#).

## Безопасность сети

*Дополнительные сведения см. в статье [Azure Security Benchmark: безопасность сети](#).*

### NS-1: реализация безопасности для внутреннего трафика

**Руководство.** при развертывании ресурсов службы Azure Bot необходимо создать или использовать существующую виртуальную сеть. Убедитесь, что все виртуальные сети Azure следуют принципу сегментации предприятия, который соответствует бизнес-рискам. Любая система, которая может повлечь за собой повышенный риск для Организации, должна быть изолирована в собственной виртуальной сети и достаточно защищена с помощью группы безопасности сети (NSG) или брандмауэра Azure.

- Использование расширения Службы приложений Direct Line в виртуальной сети
- Создание группы безопасности сети с правилами безопасности
- Развёртывание и настройка брандмауэра Azure

**Мониторинг Центра безопасности Azure:** Да

**Ответственность:** Customer

### NS-2. Совместное подключение частных сетей

**Руководство.** Использование Azure ExpressRoute или виртуальной частной сети Azure (VPN) для создания частных подключений между центрами обработки данных Azure и локальной инфраструктурой в среде совместного размещения. Подключения ExpressRoute не проходят через общедоступный Интернет и обеспечивают повышенную надежность, скорость и задержку, чем обычные подключения к Интернету. Для VPN-подключения типа "точка — сеть" и VPN-подключения типа "сеть — сеть" можно подключить локальные устройства или сети к виртуальной сети, используя любое сочетание параметров VPN и Azure ExpressRoute.

Для совместного подключения нескольких виртуальных сетей в Azure используйте пикинг виртуальных сетей. Сетевой трафик между одноранговыми виртуальными сетями является частным и хранится в магистральной сети Azure.

- [Что такое модели подключения ExpressRoute](#)
- [Общие сведения об Azure VPN](#)

- Пиринг виртуальной сети

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

#### **NS-4: Защита приложений и служб от внешних сетевых атак**

**Руководство.** Защитите ресурсы службы Azure Bot от атак из внешних сетей, в том числе от распределенных атак типа "отказ в обслуживании" (от атак DDoS), атак, зависящих от приложения, а также незапрошенных и потенциально вредоносных интернет-трафика. Используйте брандмауэр Azure для защиты приложений и служб от потенциально вредоносного трафика из Интернета и других внешних расположений. Защитите свои ресурсы от атак от атак DDoS, включив стандартную защиту от атак DDoS в виртуальных сетях Azure. Используйте центр безопасности Azure для обнаружения рисков с несоответствующими настройками, связанными с сетевыми ресурсами.

Используйте возможности брандмауэра веб-приложения (WAF) в шлюзе приложений Azure, в передней дверце Azure и сети доставки содержимого Azure (CDN) для защиты приложений, работающих в службе Azure Bot, от атак уровня приложения.

- [Документация по брандмауэру Azure](#)
- [Управление защитой от атак DDoS Azure уровня "Стандартный" с помощью портала Azure](#)
- [Рекомендации Центра безопасности Azure](#)
- [Развертывание WAF Azure](#)

**Мониторинг Центра безопасности Azure:** Да

**Ответственность:** Customer

#### **NS-5: развертывание систем обнаружения вторжений и предотвращения вторжений (ИДЕНТИФИКАТОРы и IP-адреса)**

**Руководство.** Использование брандмауэра Azure с фильтрацией на основе интеллектуального анализа угроз для оповещения и/или блокировки трафика между известными вредоносными IP-адресами и доменами. IP-адреса и домены также передаются из канала Microsoft Threat Intelligence. Если требуется проверка полезной нагрузки, можно развернуть стороннее решение ИДЕНТИФИКАТОРов и адресов из Azure Marketplace с возможностями проверки полезной нагрузки. Кроме того, вы можете использовать ИДЕНТИФИКАТОРы или IP-адреса, основанные на узлах, или решение для обнаружения конечных точек и ответа (ЕДР) в сочетании с сетевыми ИДЕНТИФИКАТОРАми или IP-адресами.

Примечание. Если у вас есть нормативная информация или другие требования для использования ИДЕНТИФИКАТОРов и IP-адресов, убедитесь, что она всегда настроена на предоставление высококачественных оповещений для вашего решения SIEM.

- [Развертывание брандмауэра Azure](#)
- [Azure Marketplace включает возможности сторонних ИДЕНТИФИКАТОРОВ.](#)
- [Функция ЕДР АТР в защитнике Майкрософт](#)

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

#### **NS-6: Упростите правила безопасности сети.**

**Руководство.** Использование тегов службы виртуальной сети для определения элементов управления доступом к сети для групп безопасности сети или брандмауэра Azure, настроенных для ресурсов службы Azure Bot. Теги служб можно использовать вместо определенных IP-адресов при создании правил

безопасности. Указав имя тега службы (например: Азуреботсервице) в соответствующем поле источника или назначения правила, можно разрешить или запретить трафик для соответствующей службы. Корпорация Майкрософт управляет префиксами адресов, входящих в тег службы, и автоматически обновляет этот тег при изменении адресов.

- [Общие сведения и использование тегов служб](#)

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

## Управление удостоверениями

Дополнительные сведения см. в статье [Azure Security Benchmark: управление удостоверениями](#).

### IM-1: стандартизация Azure Active Directory в качестве централизованной системы идентификации и проверки подлинности

**Руководство.** служба Azure Bot использует Azure Active Directory (Azure AD) в качестве службы управления удостоверениями и доступом по умолчанию. Стандартизация Azure AD для управления удостоверениями и доступом в вашей организации в:

- Облачные ресурсы Майкрософт, такие как портал Azure, служба хранилища Azure, виртуальные машины Azure (Linux и Windows), Azure Key Vault, PaaS и приложения SaaS.
- Ресурсы организации, такие как приложения в Azure или ресурсы корпоративной сети.

Защита Azure AD должна обладать высоким приоритетом по соображениям безопасности в облаке вашей организации. Azure AD предоставляет оценку безопасности удостоверений, чтобы помочь оценить состояние безопасности удостоверений в соответствии с рекомендациями Майкрософт. Используйте оценку, чтобы понять, насколько точно ваша конфигурация соответствует рекомендациям, и улучшить состояние безопасности.

Примечание. Azure AD поддерживает внешние поставщики удостоверений, которые позволяют пользователям без учетной записи Майкрософт входить в свои приложения и ресурсы по внешнему удостоверению.

- [Аренда в Azure AD](#)
- [Создание и настройка экземпляра Azure AD](#)
- [Определение клиентов Azure AD](#)
- [Использование внешних поставщиков удостоверений для приложения](#)
- [Что собой представляет оценка безопасности удостоверений в Azure AD](#)

**Мониторинг Центра безопасности Azure:** Да

**Ответственность:** Customer

### IM-3: использование единого входа Azure AD (SSO) для доступа к приложениям

**Руководство.** Azure AD предоставляет управление удостоверениями и доступом к ресурсам Azure, облачным приложениям и локальным приложениям. Управление удостоверениями и доступом применяется к корпоративным удостоверениям, таким как сотрудники, а также к внешним удостоверениям, таким как партнеры, поставщики и поставщики.

Используйте единый вход Azure AD (SSO) для управления доступом к данным и ресурсам Организации в локальной среде и в облаке, а также для обеспечения безопасного доступа к ним. Подключение всех пользователей, приложений и устройств к Azure AD для обеспечения беспрепятственного и безопасного

доступа, а так же улучшенного контроля.

- [Общие сведения о единый вход приложений в Azure AD](#)

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

**IM-4: использование строгой проверки подлинности для всех операций доступа на основе Azure Active Directory**

**Руководство.** служба Azure Bot использует Azure Active Directory, поддерживающую строгие элементы управления проверкой подлинности, с помощью многофакторной проверки подлинности (MFA) и надежных методов.

- Многофакторная проверка подлинности: включение Azure AD MFA, отслеживание удостоверений центра безопасности Azure и рекомендации по управлению доступом для настройки MFA. MFA можно применять ко всем пользователям, выбирать пользователей или на уровне отдельных пользователей, учитывая условия входа и факторы риска.
- Проверка подлинности с паролем: доступны три варианта проверки подлинности с паролем: Windows Hello для бизнеса, Microsoft Authenticator приложение и локальные методы проверки подлинности, такие как смарт-карты.

Для администраторов и привилегированных пользователей убедитесь, что используется самый высокий уровень надежной проверки подлинности, а затем выполнив соответствующую политику строгой проверки подлинности другим пользователям.

Если для проверки подлинности Azure AD по-прежнему используется устаревшая проверка подлинности на основе паролей, имейте в виду, что учетные записи только для облака (учетные записи пользователей, созданные непосредственно в Azure) имеют стандартную политику паролей. И гибридные учетные записи (учетные записи пользователей, поступающие из локальной Active Directory) следуют локальным политикам паролей. При использовании проверки подлинности на основе пароля Azure AD предоставляет возможность защиты паролем, которая запрещает пользователям задавать пароли, которые легко угадать. Корпорация Майкрософт предоставляет глобальный список запрещенных паролей, которые обновляются на основе данных телеметрии, а клиенты могут дополнять список в соответствии с потребностями (например, фирменной символикой, культурными ссылками и т. д.). Эту защиту паролей можно использовать только для облачных и гибридных учетных записей.

Примечание. Проверка подлинности на основе только учетных данных пароля является уязвимой для популярных методов атак. Для повышения безопасности используйте строгую проверку подлинности, такую как MFA, и политику надежных паролей. Для приложений сторонних производителей и служб Marketplace, которые могут иметь пароли по умолчанию, их следует изменить во время начальной настройки службы.

- [Включение MFA в Azure](#)
- [Общие сведения о способах выполнения беспарольной проверки подлинности в Azure Active Directory](#)
- [Политика паролей по умолчанию в Azure AD](#)
- [Исключение неправильных паролей с помощью защиты паролем Azure AD](#)

**Мониторинг Центра безопасности Azure:** Да

**Ответственность:** Customer

**IM-5: мониторинг и оповещение об аномалиях учетной записи**

**Руководство.** служба Azure Bot интегрирована с Azure Active Directory, которая предоставляет

следующие источники данных:

- Входы. Отчет о входах содержит информацию об использовании управляемых приложений и входах пользователей.
- Журналы аудита — возможность отслеживания с помощью журналов всех изменений, внесенных при использовании разных функций в Azure AD. Примеры журналов аудита включают изменения, вносимые в любые ресурсы в Azure AD, например добавление или удаление пользователей, приложений, групп, ролей и политик.
- Входы, представляющие риск. Вход, представляющий риск, означает, что в систему пытался войти пользователь, который не является законным владельцем учетной записи.
- Пользователи, находящиеся в группе риска. Такая пометка означает, что конфиденциальность учетной записи пользователя, возможно, нарушена.

Эти источники данных можно интегрировать с Azure Monitor, Sentinel Azure или сторонними SIEM системами.

Центр безопасности Azure также может создавать оповещения о некоторых подозрительных действиях, например о чрезмерном количестве неудачных попыток проверки подлинности и нерекомендуемых учетных записях в подписке.

Расширенная защита от угроз Azure (ATP) — это решение для обеспечения безопасности, которое может использовать сигналы Active Directory для выявления, обнаружения и изучения дополнительных угроз, скомпрометированных удостоверений и внутренних вредоносных действий.

Отчеты о действиях аудита в Azure Active Directory <https://docs.microsoft.com/azure/active-directory/reports-monitoring/concept-audit-logs>

- [Просмотр рискованных входов в Azure AD](#)
- [Как определить пользователей Azure AD, помеченных для события риска](#)
- [Мониторинг пользовательских действий идентификации и доступа в Центре безопасности Azure](#)
- [Оповещения в модуле защиты от аналитических угроз в Центре безопасности Azure](#)
- [Как интегрировать журналы действий Azure в Azure Monitor](#)

**Мониторинг Центра безопасности Azure:** Да

**Ответственность:** Customer

#### **IM-7: исключение непреднамеренного раскрытия учетных данных**

**Руководство.** служба Azure Bot позволяет клиентам развертывать и запускать код, конфигурации или сохраненные данные, потенциально с удостоверениями и секретами. рекомендуется реализовать средство проверки учетных данных для идентификации учетных данных в коде или конфигурациях или сохраненных данных. Сканер учетных данных также рекомендует перемещать обнаруженные учетные данные в более безопасные расположения, такие как Azure Key Vault.

Для GitHub можно использовать функцию проверки собственных секретов для обнаружения учетных данных или других форм секретов в коде.

- [Как настроить сканер учетных данных](#)
- [Проверка секрета GitHub](#)

**Мониторинг Центра безопасности Azure:** в настоящее время недоступен.

**Ответственность:** Customer

## Привилегированный доступ

Дополнительные сведения см. в статье [Azure Security Benchmark: привилегированный доступ](#).

### PA-2: ограничение административного доступа к системам, критически важным для бизнеса

**Руководство.** служба Azure Bot использует Azure RBAC для изоляции доступа к критически важным бизнес-системам путем определения того, какие учетные записи получают привилегированный доступ к подпискам и группам управления, в которых они находятся.

Убедитесь, что вы также ограничиваете доступ к системам управления, идентификации и безопасности, которые имеют административный доступ к важнейшим ресурсам предприятия, таким как контроллеры домен Active Directory (DCs), средства безопасности и средства управления системой с агентами, установленными в критически важных для бизнеса системах. Злоумышленники, которые нарушают эти системы управления и безопасности, могут немедленно использовать их для взлома важных для бизнеса ресурсов.

Все типы элементов управления доступом должны быть согласованы с стратегией сегментации предприятия, чтобы обеспечить постоянный контроль доступа.

Обязательно назначьте отдельные привилегированные учетные записи, отличающиеся от стандартных учетных записей пользователей, используемых для задач электронной почты, обзора и повышения производительности.

- [Компоненты и эталонная модель Azure](#)
- [Доступ к группе управления](#)
- [Администраторы подписки Azure](#)

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

### PA-3: регулярная проверка и сверка доступа пользователей

**Руководство.** служба Azure Bot использует учетные записи Azure Active Directory (Azure AD) для управления ресурсами, регулярно изучите учетные записи пользователей и назначение доступа, чтобы убедиться, что учетные записи и их уровень доступа действительны. Вы можете использовать проверки доступа Azure AD для просмотра членства в группах, доступа к корпоративным приложениям и назначения ролей. Служба отчетов Azure AD может предоставлять журналы для облегчения поиска устаревших учетных записей. Можно также использовать Azure AD Privileged Identity Management для создания рабочего процесса отчета по проверке доступа, который упрощает процесс проверки.

Кроме того, управление привилегированными пользователями Azure можно настроить на оповещение при создании чрезмерного числа учетных записей администраторов и определении устаревших или неправильно настроенных учетных записей администраторов.

Примечание. Некоторые службы Azure поддерживают локальных пользователей и роли, которые не управляются с помощью Azure AD. Эти пользователи необходимо управлять по отдельности.

- [Создание проверки доступа для ролей ресурсов Azure в управление привилегированными пользователями \(PIM\)](#)
- [Использование проверок доступа для идентификации Azure AD](#)

**Мониторинг Центра безопасности Azure:** Да

**Ответственность:** Customer

### PA-4: настройка аварийного доступа в Azure AD

**Руководство.** служба Azure Bot использует Azure Active Directory для управления ресурсами. Чтобы предотвратить случайную блокировку вашей организации Azure AD, настройте учетную запись аварийного доступа для доступа, если обычные административные учетные записи использовать нельзя. Учетные записи аварийного доступа обычно имеют высокий уровень привилегий и не должны назначаться конкретным пользователям. Учетные записи для аварийного доступа используются только в непредвиденных ситуациях, в которых невозможно использовать обычные учетные записи администратора.

Следует убедиться, что учетные данные (например, пароль, сертификат или смарт-карта) для учетных записей аварийного доступа защищены и известны только тем лицам, которые имеют право использовать их только в экстренной ситуации.

- [Управление учетными записями для аварийного доступа в Azure AD](#)

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

#### **PA-5: Автоматизация управления назначениями**

**Руководство.** служба Azure Bot интегрирована с Azure Active Directory для управления ресурсами. Используйте функции управления обслуживанием Azure AD для автоматизации рабочих процессов запросов на доступ, включая назначения доступа, обзоры и срок действия. Поддерживается также два или несколько промежуточных утверждений.

- [Что такое проверки доступа Azure AD](#)
- [Что такое управление назначениями Azure AD](#)

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

#### **PA-6: использование рабочих станций с привилегированным доступом**

**Инструкции:** защищенные изолированные рабочие станции критически важны для защиты привилегированных ролей, таких как администраторы, разработчики и критические операторы обслуживания. Используйте высокозащищенные рабочие станции пользователей и (или) Azure бастиона для административных задач. Чтобы развернуть защищенную и управляемую рабочую станцию для административных задач, используйте Azure Active Directory, Advanced Threat Protection в Microsoft Defender и (или) Microsoft Intune. Защищенными рабочими станциями можно централизованно управлять, чтобы обеспечить безопасную настройку, включая строгую проверку подлинности, базовые параметры программного обеспечения и оборудования, ограниченный логический и сетевой доступ.

- [Общие сведения о рабочих станциях с привилегированным доступом](#)
- [Развертывание рабочей станции с привилегированным доступом](#)

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

#### **PA-7. Использование Just Enough Administration (принцип предоставления наименьших прав)**

**Руководство.** служба Azure Bot интегрирована с управлением доступом на основе РОЛЕЙ (RBAC) Azure для управления ресурсами. Azure RBAC позволяет управлять доступом к ресурсам Azure посредством назначения ролей. Эти роли можно назначить пользователям, группам субъектов-служб и управляемым удостоверениям. Для некоторых ресурсов существуют предварительно определенные встроенные роли. Эти роли могут быть инвентаризированы или запрошены с помощью таких средств, как Azure CLI, Azure PowerShell или портал Azure. Привилегии, назначаемые ресурсам через Azure RBAC, должны всегда ограничиваться возможностями, которые необходимы ролям. Этот метод дополняет собой JIT-подход

Azure AD Privileged Identity Management (PIM). Его необходимо периодически проверять.

Используйте встроенные роли для выделения привилегии. Создавать настраиваемые роли можно только при необходимости.

Что такое управление доступом на основе ролей в Azure (Azure RBAC)

<https://docs.microsoft.com/azure/role-based-access-control/overview>

- [Настройка RBAC в Azure](#)
- [Использование проверок доступа для идентификации Azure AD](#)

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

## Защита данных

Дополнительные сведения см. в статье [Azure Security Benchmark: защита данных](#).

### DP-2: защита конфиденциальных данных

**Рекомендации.** Защитите конфиденциальные данные путем запрета доступа с помощью управления доступом на основе ролей Azure (Azure RBAC), управления доступом на основе сети и определенных элементов управления в службах Azure (например, шифрования в SQL и других базах данных).

Чтобы гарантировать постоянное управление доступом, все его типы должны быть согласованы с вашей корпоративной стратегией сегментации. Корпоративная стратегия сегментации также должна содержать информацию о расположении конфиденциальных или критически важных для бизнеса данных и систем.

Для базовой платформы, управляемой корпорацией Майкрософт, корпорация Майкрософт считает все содержимое клиента конфиденциальным и защищает клиентов от потери данных и раскрытия информации. Чтобы обеспечить безопасность данных клиентов в Azure, корпорация Майкрософт реализовала несколько элементов управления и возможностей защиты данных по умолчанию.

- [Управление доступом на основе ролей \(RBAC\) в Azure](#)
- [Общие сведения о защите данных клиентов в Azure](#)

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Совмещаемая блокировка

### DP-3. Мониторинг несанкционированной передачи конфиденциальных данных

**Руководство.** Мониторинг несанкционированной передачи данных в расположения за пределами корпоративной видимости и управления. Обычно этот процесс предусматривает мониторинг за аномальными действиями (большими или необычными передачами данных), которые могут указывать на несанкционированную кражу данных.

Расширенная защита от угроз (ATP) для службы хранилища Azure и Azure SQL ATP может оповещать об аномальных передачах информации, что может указывать на несанкционированную передачу конфиденциальной информации.

Azure Information Protection (AIP) предоставляет возможности мониторинга сведений, которые были классифицированы и помечены.

Если необходимо обеспечить соответствие политики защиты от потери данных (DLP), можно использовать решение DLP на основе узла, чтобы принудительно применять выявляющие и (или) профилактические элементы управления для предотвращения кражи данных.

- [Включение Azure SQL ATP](#)
- [Включение ATP службы хранилища Azure](#)

**Мониторинг Центра безопасности Azure:** в настоящее время недоступен.

**Ответственность:** Customer

#### **DP-4: шифрование конфиденциальной информации во время передачи**

**Руководство.** все конечные точки службы Azure Bot доступны по протоколу HTTP, обеспечивающему использование TLS 1,2. При использовании принудительного протокола безопасности потребители, пытающиеся вызвать конечную точку службы Azure Bot, должны соблюдать следующие рекомендации:

- Клиентская операционная система должна поддерживать TLS 1,2.
- В языке (и платформе), используемом для вызова HTTP, необходимо указать TLS 1,2 в качестве части запроса. В зависимости от языка и платформы, указание протокола TLS выполняется явно или неявно.

Для дополнения элементов управления доступом данные в передаче должны быть защищены от атак с использованием аппаратного контроллера управления (например, для записи трафика) с помощью шифрования, чтобы злоумышленники не могли легко считывать или изменять данные.

Хотя это необязательно для трафика в частных сетях, это важно для трафика во внешних и общедоступных сетях. Для трафика HTTP убедитесь, что все клиенты, подключающиеся к ресурсам Azure, могут согласовать TLS v 1.2 или более поздней версии. Для удаленного управления используйте SSH (для Linux) или RDP/TLS (для Windows) вместо незашифрованного протокола. Устаревшие версии и протоколы SSL, TLS и SSH, а слабые шифры должны быть отключены.

По умолчанию Azure обеспечивает шифрование данных при передаче между центрами обработки данных Azure.

- [Служба Azure Bot, обеспечивающая безопасность транспортного уровня \(TLS\) 1,2](#)
- [Общие сведения о шифровании при передаче с помощью Azure](#)
- [Сведения о безопасности TLS](#)
- [Двойное шифрование для данных Azure при передаче](#)

**Мониторинг Центра безопасности Azure:** Да

**Ответственность:** Совмещаемая блокировка

#### **DP-5: шифрование конфиденциальных неактивных данных**

**Руководство.** для дополнения элементов управления доступом неактивных данных следует защищать от атак с использованием аппаратного контроллера управления (например, доступ к базовому хранилищу) с помощью шифрования. Это гарантирует, что злоумышленники не смогут легко считывать или изменять данные.

По умолчанию Azure обеспечивает шифрование неактивных данных. Для высокочувствительных данных вы можете реализовать дополнительные возможности шифрования при хранении во всех ресурсах Azure, где это возможно. Azure управляет ключами шифрования по умолчанию, но Azure предоставляет варианты для управления собственными ключами (ключи, управляемые клиентом) для определенных служб Azure.

Если требуется для обеспечения соответствия в ресурсах вычислений, реализуйте сторонние средства, такие как автоматизированное решение для защиты от потери данных на узлах, чтобы принудительно применять элементы управления доступом к данным даже при копировании данных из системы.

- [Общие сведения о шифровании неактивных данных в Azure](#)

- Настройка ключей шифрования, управляемых клиентом
- Модель шифрования и таблица управления ключами
- Двойное шифрование неактивных данных в Azure

**Мониторинг Центра безопасности Azure:** в настоящее время недоступен.

**Ответственность:** Customer

## управление ресурсами.

Дополнительные сведения см. в статье [Azure Security Benchmark: управление ресурсами](#).

### AM-1. Предоставление группе безопасности возможности просматривать угрозы безопасности для ресурсов

**Руководство.** Убедитесь, что группам безопасности предоставлены разрешения читателя сведений о безопасности в клиенте и подписках Azure, чтобы они могли отслеживать угрозы безопасности с помощью Центра безопасности Azure.

В зависимости от структуры обязанностей группы безопасности за отслеживание угроз безопасности может отвечать централизованная группа безопасности или локальная группа. С другой стороны, сведения о безопасности и угрозах безопасности всегда должны централизованно располагаться внутри организации.

Разрешения читателя сведений о безопасности можно расширить для всего клиента (корневой группы управления) или ограничить до определенной группы управления или конкретных подписок.

Примечание. Для наблюдения за рабочими нагрузками и службами могут потребоваться дополнительные разрешения.

- [Общие сведения о роли читателя сведений о безопасности](#)
- [Общие сведения о группах управления Azure](#)

**Мониторинг Центра безопасности Azure:** в настоящее время недоступен.

**Ответственность:** Customer

### AM-2: убедитесь, что группа безопасности имеет доступ к инвентаризации и метаданным активов.

**Руководство.** применение тегов к ресурсам Azure, группам ресурсов и подпискам для логической организации их в таксономию. Каждый тег состоит из пары "имя — значение". Например, имя Environment и значение Production можно применить ко всем ресурсам в рабочей среде.

Служба Azure Bot не поддерживает развертывание ресурсов на основе Azure Resource Manager и обнаружение с помощью графа ресурсов Azure.

- [Как создавать запросы с помощью обозревателя Azure Resource Graph](#)
- [Управление инвентаризацией активов в центре безопасности Azure](#)
- [Дополнительные сведения о разметке ресурсов см. в разделе Описание принятия решений по именованию и маркировке.](#)

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

### AM-3: использование только утвержденных служб Azure

**Руководство.** Использование политики Azure для аудита и ограничения служб, которые пользователи

могут подготавливать в вашей среде. Используйте Azure Resource Graph для запроса и обнаружения ресурсов в своих подписках. Можно также использовать Azure Monitor, чтобы создать правила для активации оповещений при обнаружении неутвержденной службы.

- [Настройка Политики Azure и управление ею](#)
- [Как отказаться от определенного типа ресурса с помощью Политики Azure](#)
- [Как создавать запросы с помощью обозревателя Azure Resource Graph](#)

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

#### **AM-4. Обеспечение безопасности для управления жизненным циклом ресурса**

**Руководство.** Неприменимо. Службу Azure Bot нельзя использовать для обеспечения безопасности ресурсов в процессе управления жизненным циклом. Ответственность за поддержание атрибутов и сетевых конфигураций ресурсов, которые считаются высоким влиянием, несет клиент. Рекомендуется, чтобы клиент создает процесс для записи изменений атрибутов и конфигурации сети, измеряя задачи изменения и создания исправлений, как применимо.

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

#### **AM-5: ограничение возможности пользователей взаимодействовать с Azure Resource Manager**

**Руководство.** Использование условного доступа Azure AD для ограничения возможности пользователей взаимодействовать с Azure Resource Manager путем настройки "блокировать доступ" для приложения "Управление Microsoft Azure".

- [Настройка условного доступа для блокировки доступа к диспетчеру ресурсов Azure](#)

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

## **Ведение журналов и обнаружение угроз**

*Дополнительные сведения см. в статье [Azure Security Benchmark: ведение журнала и обнаружение угроз](#).*

#### **LT-1: Включение обнаружения угроз для ресурсов Azure**

**Руководство.** служба Azure Bot не предоставляет собственные возможности для мониторинга угроз безопасности, связанных с ресурсами.

Перешлите журналы из службы Azure Bot в SIEM, которые можно использовать для настройки пользовательских обнаружений угроз. Убедитесь, что вы отслеживаете различные типы ресурсов Azure для потенциальных угроз и аномалий. Сосредоточьтесь на получении высококачественных оповещений, чтобы сократить число ложных срабатываний аналитиков для сортировки. Источником предупреждений могут быть данные журнала, агенты или другие данные.

- [Создание настраиваемых правил аналитики для обнаружения угроз](#)
- [Кибератакная аналитика угроз с помощью Sentinel Azure](#)

**Мониторинг Центра безопасности Azure:** Да

**Ответственность:** Customer

#### **LT-2. Включение функции обнаружения угроз для управления удостоверениями и доступом в Azure**

**Руководство.** Azure AD предоставляет следующие журналы пользователей, которые можно

просмотреть в отчетах Azure AD или интегрировать с Azure Monitor, Azure Sentinel или другими средствами мониторинга или SIEM для использования более расширенных функций мониторинга и анализа.

- Входы. Отчет о входах содержит информацию об использовании управляемых приложений и входах пользователей.
- Журналы аудита — возможность отслеживания с помощью журналов всех изменений, внесенных при использовании разных функций в Azure AD. Примеры журналов аудита включают изменения, внесенные в такие ресурсы в Azure AD, как добавление или удаление пользователей, приложений, групп, ролей и политик.
- Входы, представляющие риск. Вход, представляющий риск, означает, что в систему пытался войти пользователь, который не является законным владельцем учетной записи.
- Пользователи, находящиеся в группе риска. Такая пометка означает, что конфиденциальность учетной записи пользователя, возможно, нарушена.

Центр безопасности Azure также может оповещать о некоторых подозрительных действиях, например о чрезмерном числе неудачных попыток проверки подлинности, а также об устаревших учетных записях в подписке. Помимо базового уровня безопасности санации, модуль защиты от угроз в центре безопасности Azure также может получать более подробные оповещения системы безопасности от отдельных ресурсов вычислений Azure (например, виртуальных машин, контейнеров, службы приложений), ресурсов данных (таких как база данных SQL и хранилище) и уровней обслуживания Azure. Эта возможность позволяет просматривать аномалии учетной записи внутри отдельных ресурсов.

- [Отчеты о действиях аудита в Azure AD](#)
- [Включение Защиты идентификации Azure](#)
- [Защита от угроз с помощью Центра безопасности Azure](#)

**Мониторинг Центра безопасности Azure:** Да

**Ответственность:** Customer

#### LT-3: включение ведения журнала для сетевых операций Azure

**Руководство.** Включение и получение журналов ресурсов группы безопасности сети (NSG), журналов потоков NSG, журналов брандмауэра Azure и брандмауэра веб-приложения (WAF) для анализа безопасности с целью поддержки исследований инцидентов, обнаружения угроз и создания предупреждений системы безопасности. Журналы потоков можно отправить в рабочую область Azure Monitor Log Analytics, а затем использовать Аналитика трафика для получения ценных сведений.

Служба Azure Bot не создает и не обрабатывает журналы запросов DNS, которые необходимо включить.

- [Включение журналов потоков для групп безопасности сети](#)
- [Включение журналов потоков для групп безопасности сети](#)
- [Журналы и метрики Брандмауэра Azure](#)
- [Как включить и использовать Аналитика трафика](#)
- [Мониторинг с помощью наблюдателя за сетями](#)
- [Решения для мониторинга сетей Azure в Azure Monitor](#)

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

#### **LT-4: включение ведения журнала для ресурсов Azure**

**Руководство.** журналы действий, которые автоматически доступны, содержат все операции записи (размещение, публикация, удаление) для ресурсов служб Azure Bot, за исключением операций чтения (Get). Журналы действий можно использовать для поиска ошибок при устранении неполадок или для мониторинга того, как пользователь в вашей организации изменил ресурс.

Сейчас служба Azure Bot не создает журналы ресурсов Azure.

- [Как получить журналы и метрики платформы с помощью Azure Monitor](#)
- [Общие сведения о ведении журналов и различных типах журналов в Azure](#)

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

#### **LT-5: централизованные управление журналом безопасности и анализ**

**Руководство.** централизованное хранение журналов и анализ для включения корреляции. Для каждого источника журнала убедитесь, что вы назначили владельца данных, руководство по доступу, место хранения, какие средства используются для обработки данных и доступа к ним, а также требования к хранению данных.

Убедитесь, что вы интегрируете журналы действий Azure в центральное ведение журнала. Прием журналов с помощью Azure Monitor для объединения данных безопасности, создаваемых устройствами конечных точек, сетевыми ресурсами и другими системами безопасности. В Azure Monitor используйте рабочие области Log Analytics для запроса и выполнения анализа и используйте учетные записи хранения Azure для долгосрочного и архивного хранения.

Кроме того, включите и подключите данные к Azure Sentinel или сторонним SIEM.

Многие организации предпочитают использовать метку Azure для "горячих" данных, которые часто используются, а служба хранилища Azure — для "холодного" данных, которые используются реже.

- [Как получить журналы и метрики платформы с помощью Azure Monitor](#)
- [Подключение к Azure Sentinel](#)

**Мониторинг Центра безопасности Azure:** Да

**Ответственность:** Customer

#### **LT-6: Настройка хранения журналов**

**Руководство.** Убедитесь, что все учетные записи хранения Log Analytics или рабочие области, используемые для хранения журналов службы Azure Bot, имеют срок хранения журнала, заданный в соответствии с нормативными требованиями вашей организации.

В Azure Monitor можно задать срок хранения Log Analytics рабочей области в соответствии с нормативными требованиями Организации. Используйте службы хранилища Azure, Data Lake или Log Analytics учетные записи рабочих областей для долгосрочного и архивного хранения.

- [Настройка срока хранения Log Analytics рабочей области](#)
- [Хранение журналов ресурсов в учетной записи хранения Azure](#)

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

**реагирование на инциденты.**

*Дополнительные сведения см. в статье [Azure Security Benchmark: реагирование на инциденты](#).*

#### **IR-1: подготовка — процесс обновления реагирования на инциденты для Azure**

**Инструкции:** убедитесь, что в организации имеются процессы для реагирования на инциденты безопасности, эти процессы обновлены для Azure и они регулярно применяются для обеспечения готовности.

- [Реализация безопасности в среде предприятия](#)
- [Справочное руководство по реагированию на инциденты](#)

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

#### **IR-2: подготовка — настройка уведомлений об инцидентах**

**Инструкции:** укажите контактные данные для информирования об инцидентах безопасности в Центре безопасности Azure. Эта контактная информация используется корпорацией Майкрософт для связи с вами, если центр Microsoft Security Response Center (MSRC) обнаруживает, что доступ к данным был осуществлен незаконно или неавторизованным лицом. Вы также можете настроить оповещения и уведомления об инцидентах в разных службах Azure в зависимости от потребностей реагирования на инциденты.

- [Как задать контакт безопасности Центра безопасности Azure](#)

**Мониторинг Центра безопасности Azure:** Да

**Ответственность:** Customer

#### **IR-3: обнаружение и анализ — создание инцидентов на основе высококачественных оповещений**

**Инструкции:** убедитесь, что имеется процесс создания высококачественных оповещений и измерения их качества. Это позволит учитывать прошлые инциденты и определять приоритеты для аналитиков, чтобы они не тратили время на ложные срабатывания.

Высококачественные оповещения можно создавать на основе опыта прошлых инцидентов, проверенных источников сообщества и средств, предназначенных для создания и очистки оповещений путем отказа и корреляции различных источников сигналов.

Центр безопасности Azure обеспечивает высококачественные оповещения для многих ресурсов Azure. Вы можете использовать соединитель данных ASC для потоковой передачи оповещений в Azure Sentinel. Azure Sentinel позволяет создавать дополнительные правила оповещений, чтобы автоматически генерировать инциденты для исследования.

Экспортируйте оповещения и рекомендации Центра безопасности Azure с помощью функции экспорта с целью выявления рисков для ресурсов Azure. Экспортируйте оповещения и рекомендации как вручную, так и в постоянном, непрерывном режиме.

- [Настройка экспорта](#)
- [Как выполнить потоковую передачу оповещений в Azure Sentinel](#)

**Мониторинг Центра безопасности Azure:** в настоящее время недоступен.

**Ответственность:** Customer

#### **IR-4. Обнаружение и анализ. Исследование инцидента**

**Руководство.** Убедитесь, что аналитики могут запрашивать и использовать различные источники данных при исследовании потенциальных инцидентов, чтобы создать полное представление о том, что произошло. Чтобы избежать неясностей при отслеживании действий потенциального злоумышленника

на этапе нарушения безопасности, необходимо собрать данные из различных журналов. Кроме того, следует убедиться, что аналитические сведения и результаты изучения записываются для других аналитиков и для хронологической справки в будущем.

К источникам данных для исследования относятся централизованные источники ведения журналов, данные которых собраны из соответствующих служб и работающих систем. Кроме того, к этим источникам можно отнести следующие ресурсы:

- Сетевые данные. Используйте журналы потоков для групп безопасности сети, службу "Наблюдатель за сетями Azure" и Azure Monitor для сборов данных журналов сетевых потоков сети и других аналитических сведений.
- Моментальные снимки работающих систем:
  - Используйте возможности моментального снимка виртуальной машины Azure для создания моментального снимка диска работающей системы.
  - Используйте возможности дампа внутренней памяти операционной системы для создания моментального снимка памяти работающей системы.
  - Используйте функцию моментального снимка служб Azure или возможность программного обеспечения для создания моментальных снимков работающих систем.

Azure Sentinel предоставляет широкие возможности аналитики данных на любом виртуальном источнике журнала и на портале управления обращениями, чтобы контролировать полный жизненный цикл инцидентов. Сведения об анализе во время исследования можно связать с инцидентом с целью отслеживания и ведения отчетности.

- [Моментальный снимок диска компьютера с Windows](#)
- [Моментальный снимок диска компьютера с Linux](#)
- [Служба поддержки Microsoft Azure: сбор диагностических сведений и дампов памяти](#)
- [Исследование инцидентов с помощью Azure Sentinel](#)

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

#### IR-5: обнаружение и анализ — определение приоритетов инцидентов

**Руководство.** Предоставьте для аналитиков контекст, на каких инцидентах следует сосредоточиться в первую очередь, исходя из серьезности предупреждения и чувствительности ресурса.

Центр безопасности Azure назначает каждому оповещению серьезность, которая поможет определить, какие предупреждения следует расследовать первыми. Серьезность основывается на том, насколько надежным является центр безопасности в поиске или на основе анализа, который используется для выдачи предупреждения, а также от уровня достоверности, который был вредоносным намерением для действия, вызвавшего оповещение.

Кроме того, пометьте ресурсы и создайте систему именования, чтобы определить и классифицировать ресурсы Azure, особенно обрабатывающие конфиденциальные данные. Вы несете ответственность за назначение приоритета оповещениям, требующим действий по исправлению, в зависимости от важности ресурсов Azure и среды, в которой произошел инцидент.

- [Оповещения безопасности в Центре безопасности Azure](#)
- [использование тегов для упорядочения ресурсов в Azure](#)

**Мониторинг Центра безопасности Azure:** в настоящее время недоступен.

**Ответственность:** Customer

#### **IR-6: заражение, удаление и восстановление — автоматизация обработки инцидентов**

**Инструкции:** автоматизируйте выполнение повторяющихся задач, чтобы сократить время отклика и снизить нагрузку на аналитиков. Задачи, выполняемые вручную, выполняются дольше, замедляя обработку каждого инцидента и уменьшая количество инцидентов, которые может обработать аналитик. Задачи, выполняемые вручную, также увеличивают усталость аналитика, что повышает риск возникновения ошибки, вызванной человеческим фактором, и снижает способность аналитиков эффективно сосредоточиться на сложных задачах. Используйте функции автоматизации рабочих процессов в Центре безопасности Azure и Azure Sentinel, чтобы автоматически активировать действия или запустить сборник схем для реагирования на входящие оповещения системы безопасности. Сборник схем отправляет уведомления, отключает учетные записи и изолирует проблемные сети.

- Настройка автоматизации рабочих процессов в Центре безопасности
- Настройка автоматического реагирования на угрозы в Центре безопасности Azure
- настройке автоматического реагирования на угрозы в Azure Sentinel

**Мониторинг Центра безопасности Azure:** в настоящее время недоступен.

**Ответственность:** Customer

## Управление состоянием защиты и уязвимостью

*Дополнительные сведения см. в статье [Azure Security Benchmark: управление состоянием защиты и уязвимостью](#).*

#### **PV-8: регулярное моделирование атак**

**Инструкции:** при необходимости выполните тестирование на проникновение в ресурсы Azure или привлеките для участия "красные команды" и обеспечьте исправление всех обнаруженных проблем с безопасностью. Следуйте правилам тестирования Microsoft Cloud на проникновение, чтобы убедиться, что тесты на проникновение не нарушают политики Майкрософт. Используйте стратегию Майкрософт и рекомендации "красных команд", а затем выполните тест на проникновение в режиме реального времени для управляемых корпорацией Майкрософт облачной инфраструктуры, служб и приложений.

- Тестирование на проникновение в Azure
- Правила взаимодействия при выполнении тестирования на проникновение
- Привлечение "красных команд для тестирования" Microsoft Cloud

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Совмещаемая блокировка

## Архивация и восстановление

*Дополнительные сведения см. в статье [Azure Security Benchmark: резервное копирование и восстановление](#).*

#### **BR-1: обеспечение регулярного автоматического резервного копирования**

**Руководство.** Резервное копирование систем и данных для обеспечения непрерывности бизнес-процессов после непредвиденного события. Это должно быть определено любыми целями для цели точки восстановления (RPO) и целевого времени восстановления (RTO).

Включите Azure Backup и настройте источник резервного копирования (например, виртуальные машины Azure, SQL Server, базы данных HANA или файловые ресурсы), а также нужную частоту и срок хранения.

Для более высокого уровня защиты можно включить геоизбыточное хранилище для репликации данных резервных копий в дополнительный регион и восстановить их с помощью операции восстановления между регионами.

- [Непрерывность бизнес-процессов и аварийное восстановление в масштабах предприятия](#)
- [Включение Azure Backup](#)
- [Как включить восстановление между регионами](#)

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

#### **BR-2: шифрование данных резервных копий**

**Руководство.** обеспечение защиты резервных копий от атак. Это должно быть шифрование резервных копий для защиты от потери конфиденциальности.

Для локальных резервных копий, использующих Azure Backup, шифрование неактивных данных предоставляется с помощью предоставленной парольной фразы. Для обычных резервных копий служб Azure данные резервного копирования автоматически шифруются с помощью ключей, управляемых платформой Azure. Вы можете выбрать шифрование резервных копий с помощью ключа, управляемого клиентом. В этом случае убедитесь, что этот ключ, управляемый клиентом, в хранилище ключей также находится в области резервного копирования.

Используйте управление доступом на основе ролей в Azure Backup, Azure Key Vault или других ресурсах, чтобы защитить резервные копии и ключи, управляемые клиентом. Кроме того, можно включить дополнительные функции безопасности, чтобы требовать MFA, прежде чем можно будет изменить или удалить резервные копии.

- [Общие сведения о средствах безопасности в Azure Backup](#)
- [Шифрование данных резервных копий с помощью управляемых клиентом ключей](#)
- [Резервное копирование ключей Key Vault в Azure](#)
- [Функции безопасности, помогающие защитить гибридные резервные копии от атак](#)

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

#### **BR-3: проверка всех резервных копий, включая ключи, управляемый клиентом**

**Рекомендации.** периодически гарантируется, что можно восстановить резервные копии ключей, управляемых клиентом.

- [Восстановление ключей Key Vault в Azure](#)

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

#### **BR-4: снижение риска потери ключей**

**Руководство.** Убедитесь, что у вас есть меры по предотвращению и восстановлению после потери ключей. Включите обратимое удаление и очистку защиты в Azure Key Vault, чтобы защитить ключи от случайного или вредоносного удаления.

- [Включение обратимого удаления и очистки защиты в Key Vault](#)

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

## Система управления и стратегия

Дополнительные сведения см. в статье [Azure Security Benchmark: управление и стратегия](#).

### GS-1: определение стратегии управления ресурсами и защиты данных

**Инструкции:** убедитесь, что вы задокументировали и распространите понятную стратегию для непрерывного мониторинга и защиты систем и данных. Определите приоритеты обнаружения, оценки, защиты и мониторинга критически важных для бизнеса данных и систем.

Эта стратегия должна включать задокументированное руководство, политику и стандарты для следующих элементов:

- Стандарт классификации данных в соответствии с бизнес-рисками
- Видение организацией обеспечения безопасности в отношении рисков и инвентаризации ресурсов
- Утверждение организацией безопасности служб Azure для использования
- Безопасность ресурсов на протяжении их жизненного цикла
- Обязательная стратегия управления доступом в соответствии с классификацией данных организации
- Использование собственных средств защиты данных Azure и средств сторонних поставщиков
- Требования к шифрованию данных для передаваемых и хранимых данных
- Соответствующие криптографические стандарты

Дополнительные сведения см. в следующих ресурсах.

- [Рекомендации по архитектуре безопасности Azure — хранилище, данные и шифрование](#)
- [Основы безопасности Azure — безопасность, шифрование и хранение данных в Azure](#)
- [Cloud Adoption Framework — рекомендации по защите и шифрованию данных в Azure](#)
- [Azure Security Benchmark — управление ресурсами](#)
- [Azure Security Benchmark — защита данных](#)

### Мониторинг Центра безопасности Azure: Неприменимо

**Ответственность:** Customer

### GS-2: определение стратегии сегментации на уровне предприятия

**Инструкции:** создайте стратегию сегментированного доступа к ресурсам на уровне предприятия, используя сочетание удостоверений, сети, приложений, подписок, групп управления и других элементов управления.

Тщательно распределите потребность в разделении безопасности, чтобы обеспечить бесперебойную ежедневную работу систем, которые должны взаимодействовать друг с другом и получать доступ к данным.

Убедитесь, что стратегия сегментации реализована единообразно по всем типам элементов управления, включая безопасность сети, модели удостоверений и доступа, а также модели разрешения и доступа приложений, равно как и управление персоналом.

- [Руководство по стратегии сегментации в Azure \(видео\)](#)

- Руководство по стратегии сегментации в Azure (документ)
- Согласование сегментации сети с помощью стратегии сегментации на уровне предприятия

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

**GS-3: определение стратегии управления состоянием безопасности**

**Инструкции:** непрерывно измеряйте и снижайте риски для отдельных ресурсов и сред, в которых они размещены. Определите приоритеты для важных ресурсов и областей, где существует высокий риск атак, таких как опубликованные приложения, точки входа в сеть и выхода из нее, конечные точки пользователя и администратора и т. д.

- Azure Security Benchmark — управление состоянием защиты и уязвимостью

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

**GS-4: согласование ролей, обязанностей и подотчетности в организации**

**Инструкции:** убедитесь, что вы задокументировали и распространяли четкие стратегии для ролей и обязанностей в организации в отношении безопасности. Определение приоритетов для обеспечения четкой отчетности за принятие решений в области безопасности, обучение всех пользователей по общей модели ответственности и обучение технических команд по технологиям защиты облака.

- Рекомендации по обеспечению безопасности в Azure 1 — люди: обучение команд обеспечению безопасности облака
- Рекомендации по обеспечению безопасности в Azure 2 — люди: обучение команд технологиям обеспечения безопасности облака
- Рекомендации по обеспечению безопасности в Azure 3 — процесс: назначение ответственных за принятие решений по обеспечению безопасности облака

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

**GS-5: определение стратегии безопасности сети**

**Инструкции:** выработайте подход к безопасности сети Azure в рамках общей стратегии управления доступом к безопасности вашей организации.

Эта стратегия должна включать задокументированное руководство, политику и стандарты для следующих элементов:

- Централизованное управление сетью и ответственность за безопасность
- Модель сегментации виртуальной сети, согласуемая со стратегией сегментации на уровне предприятия
- Стратегия исправления в различных сценариях угроз и атак
- Стратегия входящего и исходящего трафика Интернета
- Стратегия взаимодействия гибридного облака и локальной среды
- Актуальные артефакты безопасности сети (например, схемы сети, эталонная архитектура сети)

Дополнительные сведения см. в следующих ресурсах.

- Рекомендации по обеспечению безопасности в Azure 11 — архитектура: единая стратегия обеспечения безопасности
- Azure Security Benchmark — безопасность сети
- Обзор сетевой безопасности Azure
- Стратегия архитектуры корпоративной сети

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

#### **GS-6: определение стратегии использования удостоверений и привилегированного доступа**

**Инструкции:** выработайте подход к использованию удостоверений Azure и привилегированного доступа в рамках общей стратегии управления доступом к безопасности вашей организации.

Эта стратегия должна включать задокументированное руководство, политику и стандарты для следующих элементов:

- Централизованная система идентификации и аутентификации, а также взаимодействие с другими внутренними и внешними системами идентификации
- Методы строгой проверки подлинности в различных вариантах использования и условиях
- Защита пользователей с высоким уровнем привилегий
- Мониторинг и обработка аномальных действий пользователей
- Процесс проверки удостоверений пользователей и доступа, а также процесс сверки пользователей

Дополнительные сведения см. в следующих ресурсах.

- Azure Security Benchmark — управление удостоверениями
- Azure Security Benchmark — привилегированный доступ
- Рекомендации по обеспечению безопасности в Azure 11 — архитектура: единая стратегия обеспечения безопасности
- Общие сведения о безопасности при управлении удостоверениями в Azure

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

#### **GS-7: определение стратегии ведения журналов и реагирования на угрозы**

**Инструкции:** выработайте стратегию ведения журналов и реагирования на угрозы для быстрого обнаружения и устранения угроз при соблюдении требований соответствия. Определите приоритеты для аналитиков с помощью высококачественных оповещений и за счет беспрепятственного взаимодействия, чтобы они могли сосредоточиться на угрозах, а не на интеграции и действиях, выполняемых вручную.

Эта стратегия должна включать задокументированное руководство, политику и стандарты для следующих элементов:

- Роль и обязанности организации по обеспечению безопасности (SecOps)
- Четко определенный процесс реагирования на инциденты, согласующийся с NIST или другой отраслевой платформой
- Сбор и хранение журналов для поддержки обнаружения угроз, реагирования на инциденты и

обеспечения соответствия требованиям

- Централизованная доступность и корреляция сведений об угрозах с использованием SIEM, собственных возможностей Azure и других источников
- План информирования и уведомления ваших клиентов, поставщиков и широкой публики
- Использование собственных платформ Azure и решений сторонних производителей для обработки инцидентов, например для ведения журналов и обнаружения угроз, проведения судебных расследований, устранения атак и удаления потенциально опасных компонентов
- Процессы обработки инцидентов и действий, выполняемых после инцидента, таких как получение выводов и хранение доказательств

Дополнительные сведения см. в следующих ресурсах.

- [Azure Security Benchmark — ведение журнала и обнаружение угроз](#)
- [Azure Security Benchmark — реагирование на инциденты](#)
- [Рекомендации по обеспечению безопасности в Azure 4 — процесс: обновление процессов реагирования на инциденты для облака](#)
- [Руководство по выбору Azure Adoption Framework, решения для ведения журналов и создания отчетов](#)
- [Масштабирование Azure Enterprise, управление и мониторинг](#)

**Мониторинг Центра безопасности Azure:** Неприменимо

**Ответственность:** Customer

## Следующие шаги

- См. [Обзор Azure Security Benchmark версии 2](#)
- Дополнительные сведения о [базовой конфигурации безопасности Azure](#).

# Шифрование службы Azure Bot для неактивных данных

27.03.2021 • 10 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Служба Azure Bot автоматически шифрует ваши данные при сохранении в облаке для защиты данных и соблюдения обязательств по обеспечению безопасности и соответствия требованиям Организации.

Шифрование и расшифровка прозрачны, то есть Управление шифрованием и доступом осуществляется за вас. Данные безопасны по умолчанию, и вам не нужно изменять код или приложения, чтобы воспользоваться преимуществами шифрования.

## Об управлении ключами шифрования

По умолчанию в подписке используются ключи шифрования, управляемые корпорацией Майкрософт. Мы также предоставляем возможность управлять ресурсами Bot с помощью собственных ключей, которые называются ключами, управляемыми клиентом (CMK). CMK обеспечивает большую гибкость в создании, повороте, отключении и отмене контроля доступа к данным хранилищам службы Azure Bot. Они также дают возможность выполнять аудит ключей шифрования, используемых для защиты ваших данных.

При шифровании данных служба Azure Bot шифруется с использованием двух уровней шифрования. В случае, когда CMK не включен, используются ключи, управляемые корпорацией Майкрософт. При настройке CMK данные шифруются как с помощью CMK, так и с помощью ключа, управляемого корпорацией Майкрософт.

## Управляемые клиентом ключи с использованием Azure Key Vault

Чтобы использовать функции, управляемые клиентом, необходимо хранить ключи и управлять ими в Azure Key Vault. Можно либо создать собственные ключи и хранить их в хранилище ключей, либо использовать API-интерфейсы Azure Key Vault для их генерации. Регистрация Bot и хранилище ключей должны находиться в одном клиенте Azure Active Directory (Azure AD), но они могут находиться в разных подписках. Дополнительные сведения о Azure Key Vault см. в разделе [что такое Azure Key Vault?](#).

При использовании ключа, управляемого клиентом, служба Azure Bot шифрует данные в хранилище, так что при отзыве доступа к этому ключу или удалении ключа программа Bot не сможет использовать службу Azure Bot для отправки и получения сообщений, и вы не сможете получить доступ к конфигурации и изменить конфигурацию регистрации Bot в портал Azure.

### IMPORTANT

Команда обслуживания Azure Bot не может восстановить управляемый клиентом код пользователя с ключом шифрования без доступа к ключу.

## Какие данные шифруются?

Служба Azure Bot хранит данные клиентов о работе, используемых им каналах, настройках конфигурации, заданных разработчиками, и при необходимости записывает текущие активные беседы. Кроме того, в течение 24 часов служба хранит сообщения, отправленные через прямые линии или каналы веб-чата, а также любые отправленные вложения.

Все данные клиента шифруются с помощью двух уровней шифрования в службе Azure Bot с помощью ключей шифрования, управляемых Майкрософт, или ключей шифрования, управляемых клиентами Майкрософт. Служба Azure Bot шифрует временные хранимые данные с помощью ключей шифрования, управляемых корпорацией Майкрософт, и, в зависимости от конфигурации регистрации Bot, шифрует долгосрочные данные с помощью ключей шифрования, управляемых корпорацией Майкрософт или клиентом.

#### NOTE

Так как служба Azure Bot существует для предоставления клиентам возможности доставки сообщений пользователям и другим службам за пределами службы Azure Bot, шифрование не распространяется на эти службы. Это означает, что при использовании нашего элемента управления данные будут храниться в зашифрованном виде в соответствии с инструкциями в этом документе. Однако при уходе службы для доставки в другую службу данные расшифровываются, а затем отправляются с использованием шифрования TLS 1,2 в целевую службу.

## Настройка экземпляра Azure Key Vault

Для использования ключей, управляемых клиентом, с помощью службы Azure Bot необходимо включить два свойства в экземпляре Azure Key Vault, который планируется использовать для размещения ключей шифрования: **обратимое удаление** и **Очистка защиты**. Эти функции гарантируют, что если по какой-либо причине ваш ключ случайно удален, его можно восстановить. Дополнительные сведения о защите от обратимого удаления и очистки см. в статье [обзор Azure Key Vault обратимого удаления](#).



Если вы используете имеющийся экземпляр Azure Key Vault, проверьте, включены ли эти свойства, на портале Azure в разделе **Свойства**. Если любое из этих свойств не включено, ознакомьтесь с разделом Key Vault в разделе [Включение защиты от обратимого удаления и очистки](#).

### Предоставление службе Azure Bot доступа к хранилищу ключей

Чтобы служба Azure Bot обладала доступом к хранилищу ключей, созданному для этой цели, необходимо задать политику доступа, которая предоставляет субъекту-службе Azure Bot текущий набор разрешений. Дополнительные сведения о Azure Key Vault, в том числе о создании хранилища ключей, см. в разделе [about Azure Key Vault](#).

1. Зарегистрируйте поставщик ресурсов службы Azure Bot в подписке, содержащей хранилище ключей.
  - a. Перейдите на [портал Microsoft Azure](#).
  - b. Откройте колонку **подписки** и выберите подписку, содержащую хранилище ключей.
  - c. Откройте колонку **поставщики ресурсов** и зарегистрируйте поставщик ресурсов Microsoft **ботсервице**.

Provider	Status
Microsoft.Web	Registered
microsoft.insights	Registered
Microsoft.ContainerRegistry	Registered
Microsoft.ManagedIdentity	Registered
Microsoft.Sql	Registered
Microsoft.PolicyInsights	Registered
Microsoft.Network	Registered
Microsoft.Compute	Registered
Microsoft.OperationalInsights	Registered
Microsoft.Management	Registered
Microsoft.AlertsManagement	Registered
Microsoft.Authorization	Registered
<b>Microsoft.BotService</b>	<b>Registered</b>
Microsoft.ClassicStorage	Registered
Microsoft.CognitiveServices	Registered
Microsoft.DocumentDB	Registered
Microsoft.ResourceHealth	Registered
Microsoft.KeyVault	Registered
Microsoft.OperationsManagement	Registered
Microsoft.Search	Registered

2. Настройте политику доступа к хранилищу ключей, предоставив субъекту-службе **Кмек Service** (из операций управления ключами) и разворачивать ключ и ключ перетекания (из операций шифрования).

- Откройте колонку **хранилища ключей** и выберите хранилище ключей.
- Убедитесь, что приложение Bot **Кмек произв**. Добавлено в качестве политики доступа и имеет следующие три разрешения. Может потребоваться добавить приложение Bot Service **кмек** в качестве политики доступа к хранилищу ключей.
- Нажмите кнопку **сохранить**, чтобы сохранить внесенные изменения.

The screenshot shows the 'Access policies' section of the Key Vault blade. The 'Key Permissions' dropdown is open, showing 'Bot Service CMEK Prod' selected. Other options in the dropdown include '3 selected', '0 selected', and '0 selected'. The 'Secret Permissions' and 'Certificate Permissions' dropdowns are also visible but empty. The 'Save' button is highlighted with a red box at the top left of the blade.

3. Разрешить Key Vault обход брандмауэра.

- Откройте колонку **хранилища ключей** и выберите хранилище ключей.
- Откройте колонку **сети** и перейдите на вкладку **брандмауэры и виртуальные сети**.
- Если параметр **Разрешить доступ из** установлен в значение **Частная конечная точка и выбранные сети**, установите флажок **разрешить доверенным службам Microsoft обходить этот брандмауэр** на **Да**.
- Нажмите кнопку **сохранить**, чтобы сохранить внесенные изменения.

The screenshot shows the 'Networking' section of the Azure Key Vault settings. On the left, there's a sidebar with 'Key vault' navigation. In the main area, under 'Firewalls and virtual networks', the 'Save' button is highlighted with a red box. Below it, 'Allow access from:' has two options: 'All networks' (radio button) and 'Private endpoint and selected networks' (radio button, which is selected and highlighted with a red box). A note says 'Configure network access control for your key vault. Learn more'. Under 'Virtual networks', there are buttons for '+ Add existing virtual networks' and '+ Add new virtual network'. At the bottom, there's a 'Firewall' section with an 'IPv4 address or CIDR' input field and an 'Exception' section where 'Allow trusted Microsoft services to bypass this firewall?' is set to 'Yes' (radio button selected, highlighted with a red box).

## Активация управляемых клиентом ключей

Чтобы зашифровать регистрацию Bot с помощью управляемого клиентом ключа шифрования, выполните следующие действия.

1. Откройте колонку ресурсов для Bot.
2. Откройте колонку **Шифрование** программы-робота и выберите **ключи, управляемые клиентом**, для **типа шифрования**.
3. Либо введите полный URI ключа, включая версию, либо щелкните **выбрать хранилище ключей и ключ**, чтобы найти ключ.
4. Щелкните **сохранить** в верхней части колонки.

The screenshot shows the 'Encryption' section of the Azure Bot Service settings. On the left, there's a sidebar with 'Bot Channels Registration' navigation. In the main area, the 'Save' button is highlighted with a red box. Below it, a note says 'The Azure Bot Service automatically encrypts your resource to protect your data and meet organizational security and compliance commitments.' Another note says 'By default, Microsoft-managed encryption keys are used. For greater flexibility in managing keys or controlling access to your subscription, select Customer-Managed Keys (CMK), also known as Bring Your Own Key (BYOK). Learn more about Azure Bot Service encryption' with a link. Under 'Encryption type', there are two radio buttons: 'Microsoft-Managed Keys' (unchecked) and 'Customer-Managed Keys' (checked and highlighted with a red box). A note below explains that this bot service resource will be granted access to the selected key vault. At the bottom, there's a 'Key URL \*' input field containing 'https://', a dropdown menu 'Select a key vault and a key' (highlighted with a red box), and an 'Encryption Status' section showing 'Microsoft managed encryption'.

После выполнения этих действий служба Azure Bot начнет процесс шифрования. Это может занять некоторое время (до 24 часов). В течение этого периода времени Bot будет полностью функциональным.

## Смена ключей, управляемых клиентом

Для смены ключа шифрования, управляемого клиентом, необходимо обновить ресурс службы Azure Bot, чтобы он использовал новый универсальный код ресурса (URI) для нового ключа (или новой версии существующего ключа).

Так как повторное шифрование с помощью нового ключа происходит асинхронно, убедитесь, что старый ключ остается доступным, чтобы данные можно было продолжать расшифровывать. В противном случае программа Bot может прерывать работу. Старый ключ следует хранить не менее одной недели.

#### Отозвать доступ к ключам, управляемым клиентом

Чтобы отозвать доступ, удалите политику доступа для субъекта-службы **Kmsk Service** в хранилище ключей.

##### NOTE

Отмена доступа приведет к нарушению большинства функций, связанных с программой-роботом. Чтобы отключить функцию "ключи, управляемые клиентом", отключите эту функцию перед отзывом доступа, чтобы убедиться, что Bot может продолжать работу.

## Следующие шаги

Дополнительные сведения о [Azure Key Vault](#)

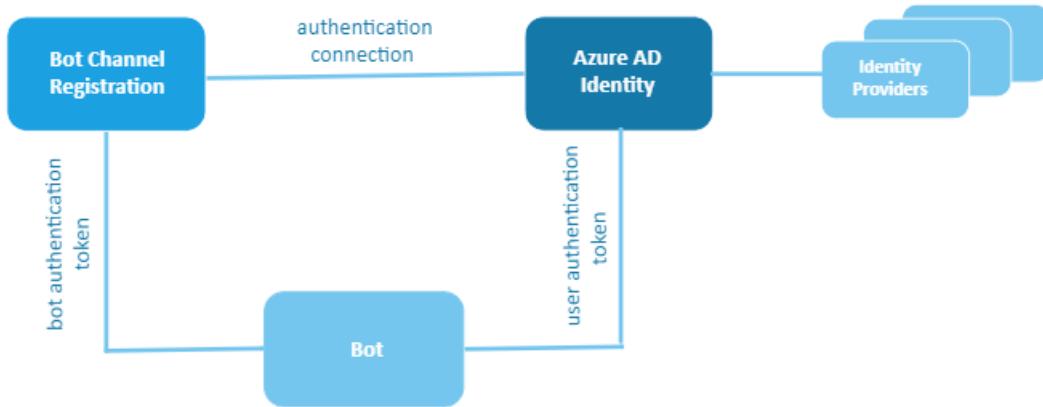
# Основы проверки подлинности на платформе Bot

27.03.2021 • 6 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Часто Bot должен обращаться к защищенным ресурсам, например к учетной записи электронной почты, от имени пользователя. Для этого необходимо **авторизовать** робота на основе учетных данных пользователя. Перед этим необходимо сначала **пройти проверку подлинности** пользователя. Более того, программа Bot должна быть известной сущностью, которая должна пройти проверку подлинности в контексте службы Azure Bot. Это происходит до того, как роботу присвоена авторизация для работы от имени пользователя.

Давайте посмотрим, можно ли упомянуть этот пакет с помощью представления "глаз" в контексте проверки подлинности на платформе Bot.



- **Регистрация канала Bot.** При регистрации программы Bot в Azure, например с помощью **регистрации каналов Bot**, Azure создает приложение регистрации Active Directory (AD). Это приложение имеет идентификатор приложения (`MicrosoftAppID`) и секрет клиента (`MicrosoftAppPassword`). Эти значения используются в файлах конфигурации Bot, как описано ниже. Обратите внимание, что вы можете получить аналогичные результаты, создав **робот веб-приложения**.
- **Удостоверение Azure AD.** Облачная служба идентификации Azure Active Directory (Azure AD) позволяет создавать приложения с поддержкой безопасного входа пользователей по стандартным отраслевым протоколам, таким как OAuth 2.0. Вы создаете приложение AD и используете его **идентификатор приложения и пароль** для выбора **поставщика удостоверений** и создания подключения для **проверки подлинности**. Это подключение добавляется к параметрам регистрации канала Bot. Имя подключения также добавляется в файлы конфигурации Bot, как описано ниже.
- **Бот.** Робот идентифицируется по его регистрации каналов (или **веб-приложению**) и **паролю**. Вы добавляете связанные значения в файлы конфигурации Bot (`appsettings.json` (.NET), (`.env` JavaScript), `config.py` (Python)) или в **Azure Key Vault**. В файлы также добавляется имя подключения. Bot использует **маркер** на основе идентификатора приложения и пароля для доступа к защищенным ресурсам. Кроме того, Bot использует **маркер** на основе подключения проверки подлинности для доступа к защищенным ресурсам пользователя.

## Проверка подлинности и авторизация на Bot

Ниже приведены основные шаги для проверки подлинности робота и авторизации для доступа к защищенным ресурсам пользователя.

1. Создайте приложение регистрации канала Bot.
2. Добавьте идентификатор приложения регистрации и пароль в файл конфигурации Bot. Это позволяет проверить подлинность робота для доступа к защищенным ресурсам.
3. Создайте приложение Azure AD, чтобы выбрать поставщика удостоверений для проверки подлинности пользователя.
4. Создайте подключение проверки подлинности и добавьте его в параметры регистрации канала.
5. Добавьте имя подключения в файлы конфигурации Bot. Это позволяет пользователю-роботу иметь право доступа к защищенным ресурсам пользователя.

Полный пример см. в разделе [Добавление проверки подлинности в Bot](#).

### Рекомендации

- Не заключайте регистрацию приложений AAD в исходное назначение приложению службы.
- Создайте дополнительное приложение AAD для проверки подлинности любого пользователя, чтобы получить более ограниченный контроль над отключением подключений проверки подлинности, откатом секретов или повторном использованием приложения AAD с другими приложениями.

Ниже перечислены некоторые проблемы, которые возникают, если для проверки подлинности также используется приложение регистрации AAD:

- Если необходимо обновить сертификат, подключенный к регистрации приложения AAD, это повлияет на пользователей, которые прошли проверку подлинности в других службах AAD с помощью сертификата.
- Как правило, он создает единую точку отказа и управляет всеми действиями, связанными с аутентификацией, с помощью программы-робота.

## См. также

В следующих статьях содержатся подробные сведения и примеры проверки подлинности на платформе Bot. Начните с просмотра [типов проверки подлинности](#) и [поставщиков удостоверений](#).

СТАТЬЯ	ОПИСАНИЕ
<a href="#">Типы проверки подлинности</a>	Описание двух типов проверки подлинности платформы Bot и используемых ими токенов.
<a href="#">Поставщики удостоверений</a>	Описывает использование поставщиков удостоверений. Они позволяют создавать приложения для безопасного входа пользователей с помощью стандартных отраслевых протоколов, таких как OAuth 2.0.
<a href="#">Проверка подлинности пользователей</a>	Описание проверки подлинности пользователя и связанного маркера для авторизации программы-робота для выполнения задач от имени пользователя.
<a href="#">Единый вход</a>	Описание проверки подлинности одного пользователя для доступа к нескольким защищенным ресурсам.
<a href="#">Регистрация каналов бота</a>	Здесь показано, как зарегистрировать робот с помощью службы Azure Bot.

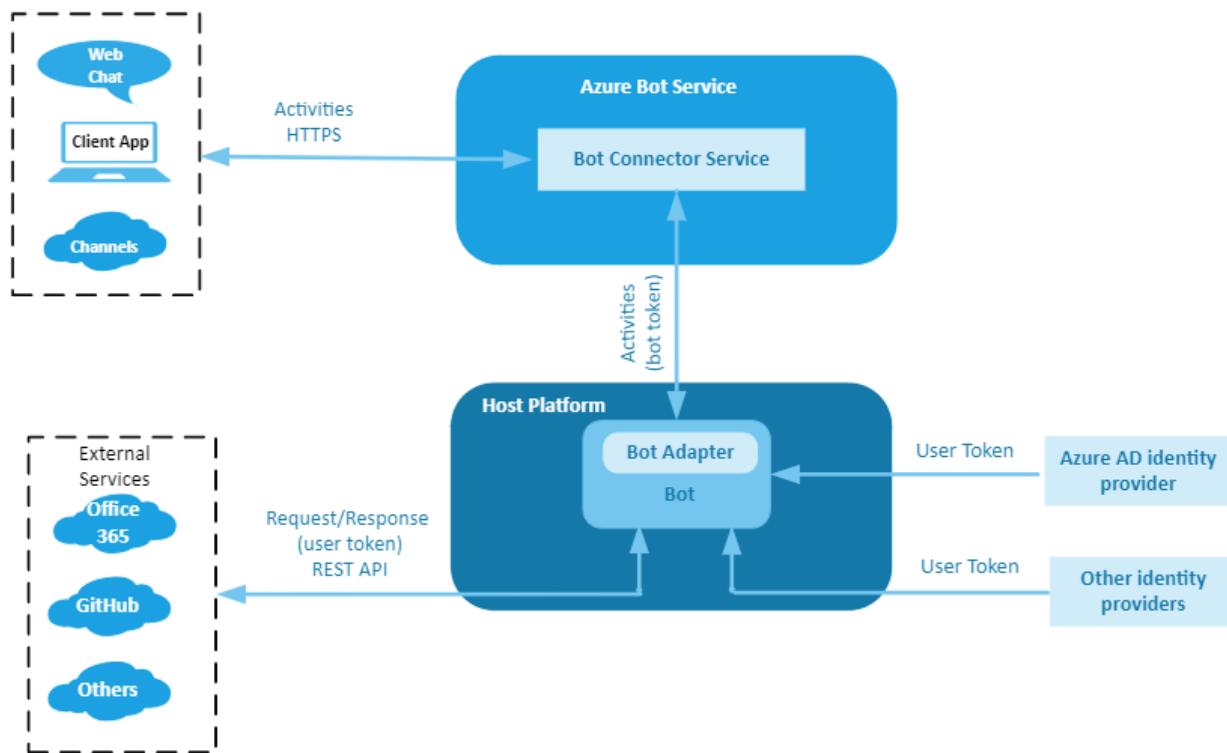
Статья	Описание
<a href="#">Рекомендации по безопасности Bot Framework</a>	Общие сведения о безопасности в целом и ее применении к платформе Bot.
<a href="#">Добавление аутентификации в веб-приложение</a>	Показывает, как создать регистрацию канала Bot, подключение проверки подлинности и подготовить код.
<a href="#">Добавление функции единого входа в бота</a>	Показывает, как добавить проверку подлинности единого входа в Bot.

# Типы проверки подлинности

27.03.2021 • 11 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В среде Bot существуют две широкие категории проверки подлинности: **Bot-аутентификация** и **Проверка подлинности пользователя**. У каждого есть связанный **маркер**, позволяющий получить доступ к защищенным ресурсам. На следующем рисунке показаны элементы, участвующие в обеих проверках подлинности: Bot и пользователя.



На рисунке ниже.

- **Платформа узла** — это платформа для размещения на сервере Bot. Это может быть Azure или любая выбранная платформа узла.
- **Служба соединителя** Bot упрощает обмен данными между Bot и каналом. Он преобразует сообщения, полученные из каналов, в объекты действий и отправляет их в конечную точку обмена сообщениями Bot. Аналогичным образом объекты действий, полученные от Bot, преобразуются в сообщения, понятные каналу, и отправляются в канал.
- **Адаптер Bot**. Это адаптер инфраструктуры Bot по умолчанию. Она выполняет следующие задачи:
  - Преобразует полезные данные JSON в объект. На этом этапе он уже является объектом действия благодаря службе соединителя Bot.
  - Создает контекст переворачивания и добавляет к нему объект действия.
  - Запускает по промежуточного слоя (при наличии).
  - Пересыпает контекст включения в Bot.

#### NOTE

Если используется адаптер настраиваемого канала, адаптер выполняет задачи, выполняемые службой соединителя Bot и адаптером Bot по умолчанию. Кроме того, он предоставляет механизм проверки подлинности для соответствующего API веб-перехватчика. Пример см. в статье [подключение ленты к резервному времени с помощью адаптера временного резерва](#).

## Проверка подлинности бота

Робот определяется его **микрософтаппид** и **микрософтаппассворд**, которые хранятся в файлах параметров программы-робота (`appsettings.json` (.NET), `.env` (JavaScript), `config.py` (Python)) или в Azure Key Vault. Дополнительные сведения см. в разделе [микрософтаппид и микрософтаппассворд](#).

При регистрации программы-робота в портал Azure, например с помощью **регистрации каналов Bot**, Azure создает приложение регистрации Active Directory (AD). Если вы используете интерфейс командной строки Bot Framework, необходимо специально выполнить шаг для создания регистрации AD. Эта регистрация содержит идентификатор приложения (`MicrosoftAppID`) и секрет клиента (`MicrosoftAppPassword`). Azure использует эти значения для создания **маркера**, с помощью которого Bot может получить доступ к защищенным ресурсам.

Когда канал отправляет запрос к Bot через службу соединителя Bot, он задает **маркер в заголовке авторизации** запроса. Программа-робот выполняет аутентификацию вызовов из службы соединителя Bot, проверяя подлинность маркера.

Когда Bot отправляет запрос каналу через **службу соединителя Bot**, он должен указать **маркер в заголовке авторизации** запроса. Все запросы должны включать маркер доступа, который проверяется службой соединителя Bot для авторизации запроса.

Описанные операции автоматически выполняются с помощью пакета SDK для Bot Framework.

Дополнительные сведения см. в REST API документации по [проверке подлинности запросов от службы соединителя Bot до программы Bot](#) и [проверки подлинности запросов от программы-робота к службе соединителя Bot](#).

### Каналы

Как правило, каналы взаимодействуют с Bot через **службу соединителя Bot**. Это означает, что в целом применяются предыдущие принципы проверки подлинности. Вы можете захотеть заметить характеристики конкретных каналов.

#### Direct Line

Помимо стандартных поддерживаемых каналов, клиентское приложение может взаимодействовать с Bot с помощью канала прямой линии.

Клиентское приложение проверяет подлинность запросов на прямую линию (версия 3,0) с помощью **секрета**, полученного на странице [настройки канала прямой линии](#) в портал Azure или, лучше, используя **маркер**, полученный во время выполнения. Секрет или токен задаются в заголовке авторизации каждого запроса.

#### IMPORTANT

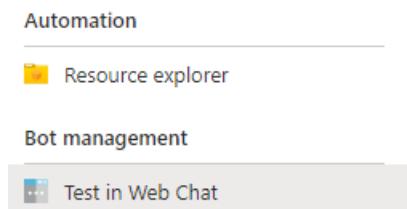
При использовании проверки подлинности службы Azure Bot для Web Chat важно учитывать несколько моментов, связанных с безопасностью. См. сведения о [безопасности](#) в руководстве по проверке подлинности REST.

Дополнительные сведения см. в [статьях скрытие секрета, Обмен секретами на маркер и встраивание](#).

## Веб-чат.

Веб-чат имеет две реализации: **канал и элемент управления**.

- При регистрации программы Bot в Azure канал Интернет-чата автоматически настраивается для тестирования робота.



Дополнительные сведения см. в статье [Подключение программы-робота к Интернету](#).

- Для предоставления доступа к роботу в клиентском приложении можно использовать веб-элемент управления Chat с прямым каналом линии. Дополнительные сведения об элементе управления см. в разделе [веб-чат для Bot Framework](#).

## Навыки

Навык и потребитель опыта — это два отдельных программы-роботы, каждый из которых имеет свой идентификатор приложения и пароль.

- Потребитель может перенаправить действия пользователя к навыку и перенаправить ответы навыка пользователю.
- Для навыка потребитель навыков выступает в качестве канала. Потребитель имеет конечную точку узла навыка, которая выступает в качестве URL-адреса службы, к которому отправляются действия навыка.
- Дополнительные сведения о навыках см. в [обзоре навыков](#).

Проверка подлинности на уровне службы выполняется службой Bot Connector. Эта платформа использует токены носителя и идентификаторы приложения бота для идентификации каждого бота.

### IMPORTANT

Для этого требуется, чтобы все программы-роботы (потребитель опыта и все навыки, используемые им) имели действительные учетные данные приложения.

## Проверка утверждений

В дополнение к этому базовому уровню проверки подлинности необходимо добавить *проверяющий элемент управления утверждения* в конфигурацию проверки подлинности и получателя навыков. Утверждения оцениваются после заголовка проверки подлинности. Это позволяет каждому роботу ограничить другие программы-роботы, от которых он будет принимать действия.

Пример проверки утверждений см. в разделе [Реализация навыка](#) и [Реализация потребителя навыков](#).

## Bot Framework Emulator

Эмулятор Bot Framework имеет собственный поток проверки подлинности и его собственные токены. Эмулятор имеет собственный канал и встроенный сервер.

## Аутентификация пользователей

Иногда Bot должен получать доступ к защищенным сетевым ресурсам от имени пользователя. Для этого необходимо авторизовать робота. Это связано с тем, что для выполнения определенных операций, таких как проверка электронной почты, проверка состояния рейса или размещение заказа, Bot должен вызывать

внешнюю службу, например Microsoft Graph, GitHub или службу RESTFUL компании. OAuth используется для проверки подлинности пользователя и авторизации робота.

#### NOTE

Для доступа робота к ресурсам пользователя используются два макроса.

1. **Проверка подлинности.** Процесс проверки удостоверения пользователя.
2. **Авторизация.** Процесс проверки того, что Bot может получить доступ к ресурсам пользователя.

Если первый шаг выполнен успешно, выдается маркер на основе учетных данных пользователя. На втором шаге Bot использует маркер для доступа к ресурсам пользователя.

Дополнительные сведения см. в разделе [Проверка подлинности пользователя](#).

#### Поставщики удостоверений

Поставщик удостоверений выполняет проверку подлинности удостоверений пользователя или клиента и выдает одноразовые маркеры безопасности. Он предоставляет проверку подлинности пользователей как услугу. Клиентские приложения, например веб-приложения, делегируют аутентификацию доверенному поставщику удостоверений.

Робот может использовать доверенный поставщик удостоверений для:

- Включите функции единого входа (SSO), что позволит ему получить доступ к нескольким защищенным ресурсам.
- Подключайтесь к облачным вычислительным ресурсам от имени пользователя, уменьшая необходимость повторной проверки подлинности пользователей.

#### NOTE

Маркер, выданный во время программы- **робота**, не совпадает с токеном, выданным при **проверке подлинности пользователя**. Первый используется для установления безопасного обмена данными между Bot, каналами и, в конечном итоге, клиентскими приложениями. Второй используется для авторизации робота для доступа к защищенному ресурсу от имени пользователя.

Обратите внимание, что каналы предоставляют собственную, отдельную проверку подлинности пользователя, чтобы пользователь мог войти в канал.

Дополнительные сведения о том, как программы-роботы может использовать поставщики удостоверений для доступа к ресурсам от имени пользователя, см. в статье [поставщики удостоверений](#).

# Аутентификация пользователей

27.03.2021 • 9 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Иногда Bot должен обращаться к защищенным сетевым ресурсам от имени пользователя, например проверять электронную почту, проверять состояние рейса или размещать заказ. Пользователь должен авторизовать программу-робота сделать это от своего имени и, чтобы авторизовать робота, пользователь должен пройти проверку подлинности. OAuth используется для проверки подлинности пользователя и авторизации робота. См. также [типы проверки подлинности](#).

Если вы хотите освежить свои знания об OAuth, см. следующие статьи:

- [обзор OAuth](#) (более простое изложение, чем в формальной спецификации);
- [спецификация OAuth](#).

## Проверка подлинности пользователей в беседе

Для выполнения определенных операций от имени пользователя, включая проверку электронной почты, создание ссылки на календарь, проверку состояния рейсов или размещение заказа, боту нужно вызывать внешнюю службу, например Microsoft Graph, GitHub или корпоративные службы REST. У каждой внешней службы есть свой метод для защиты таких вызовов. Часто запросы выдаются с использованием [токена пользователя](#), который однозначно определяет пользователя во внешней службе (иногда его называют [JSON Web Token](#) (JWT)).

Чтобы защитить вызов к внешней службе, бот должен попросить пользователя выполнить вход, чтобы он получил токен пользователя для этой службы. Многие службы поддерживают получение токенов по протоколу OAuth или OAuth 2.

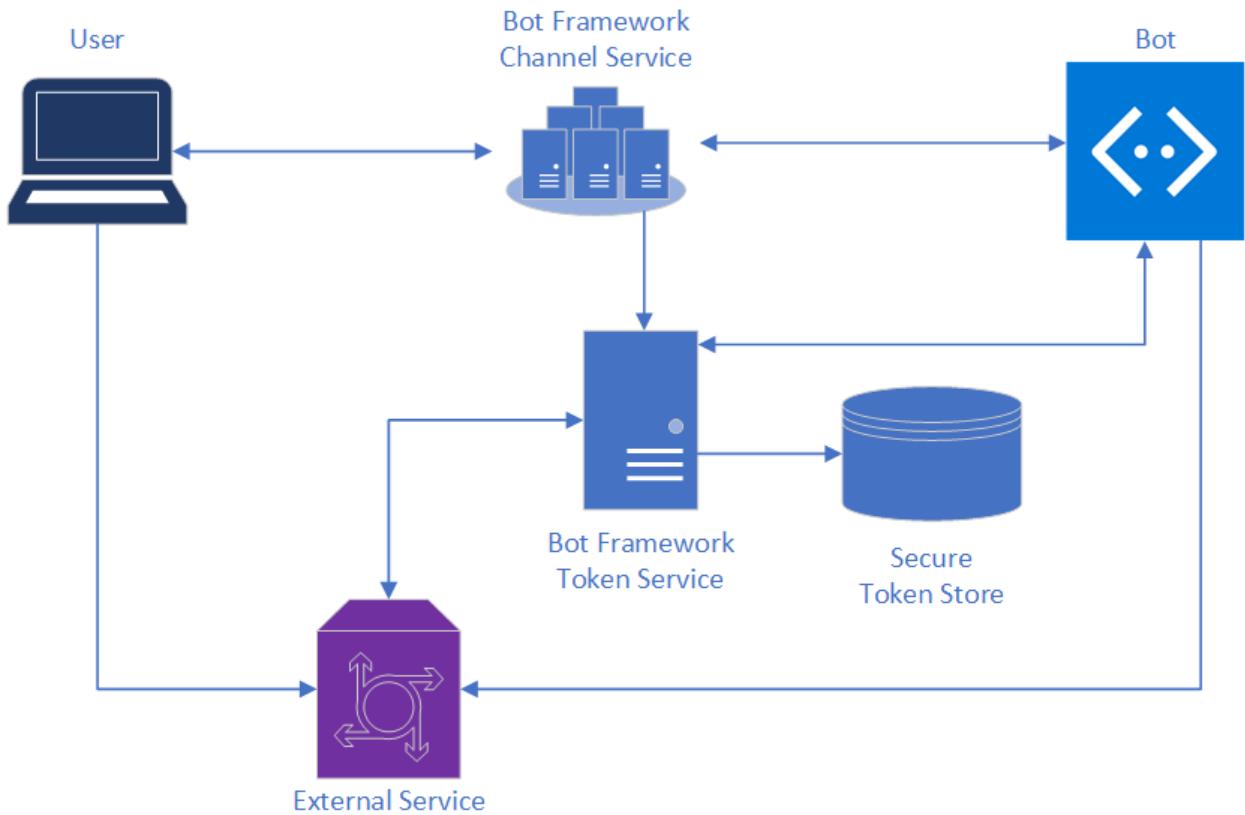
Служба Azure Bot предоставляет специализированные карты [для входа](#) и службы, которые работают с протоколом OAuth и управляют жизненным циклом токенов. Бот может использовать эти функции, чтобы получить токен пользователя.

- В рамках конфигурации бота в ресурсе службы Azure Bot в Azure регистрируется [подключение OAuth](#).

Это подключение содержит сведения об используемом [поставщике удостоверений](#), а также секрет и допустимый идентификатор клиента OAuth, разрешенные области OAuth и другие метаданные подключения, требуемые этим поставщиком удостоверений.

- В коде бота параметр подключения OAuth используется для входа пользователей и получения токена пользователя.

На следующем рисунке показаны элементы, который участвуют в процессе аутентификации.



## О службе токенов Bot Framework

Служба токенов Bot Framework отвечает за следующие задачи:

- упрощение использования протокола OAuth с разными внешними службами;
- безопасное хранение токенов для определенного бота, канала, диалога и пользователя;
- получение токенов пользователей.

### TIP

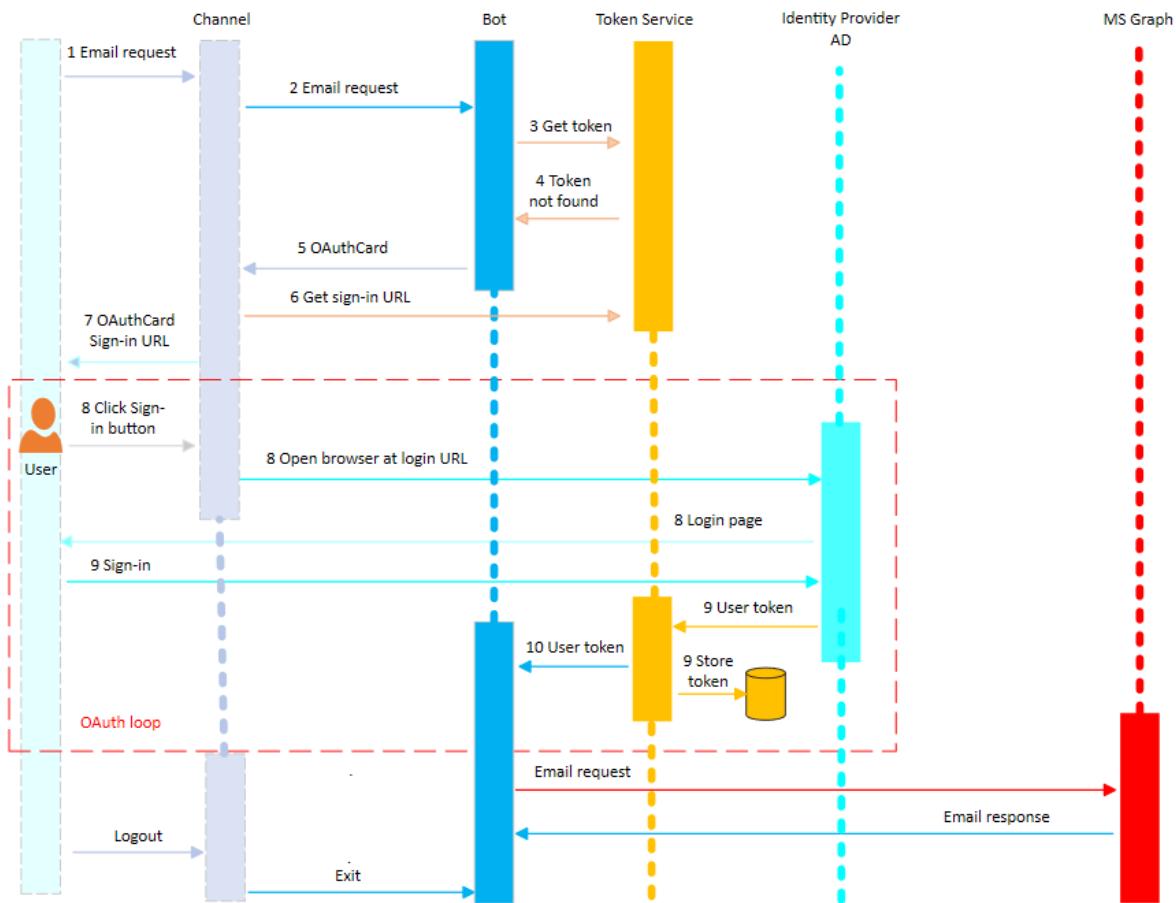
Если в боте применяется токен пользователя с истекшим сроком действия, бот должен сделать следующее.

- Выход пользователя
- Инициировать поток входа снова.

Например, если бот будет проверять последние сообщения электронной почты пользователя через API Microsoft Graph, ему нужно получить токен пользователя у **поставщика удостоверений** — в нашем примере это Azure Active Directory. Разработчик бота на этапе разработки выполняет следующие два важных шага:

1. Регистрирует приложение Azure Active Directory, то есть поставщик удостоверений, в службе Token Framework Bot через портал Azure.
2. Настраивает для бота подключение OAuth (например, `GraphConnection`).

На следующем рисунке показана последовательность взаимодействия пользователя с ботом при выполнении запроса по электронной почте через службу Microsoft Graph.



- Пользователь отправляет по электронной почте запрос к боту.
- Действие с этим сообщением отправляется от пользователя в службу канала Bot Framework. Служба канала гарантирует, что поле `userid` настроено для действия, а сообщение отправлено боту.

#### NOTE

Идентификаторы пользователя зависят от канала, включая идентификатор Facebook или номер телефона, с которого отправляются SMS.

- Бот отправляет запрос службе токенов Bot Framework, чтобы узнать, если ли у нее токен для `UserId` для подключения OAuth (`GraphConnection`).
- Так как пользователь взаимодействует с ботом впервые, у службы токенов Bot Framework еще нет токена для этого пользователя, поэтому она возвращает боту ответ *NotFound*.

#### NOTE

Если токен найден, шаги проверки подлинности пропускаются, и бот может выполнить запрос по электронной почте с сохраненным токеном.

- Бот создает OAuthCard с именем подключения `GraphConnection` и обращается к пользователю, чтобы он выполнил вход с использованием этой карты.
- Действие передается в службу канала Bot Framework, которая создает вызов в службу токенов Bot Framework для создания допустимого URL-адреса для входа (OAuth) для этого запроса. Этот URL-адрес для входа добавляется в OAuthCard, а сама карта затем возвращается пользователю.
- Пользователь получает приглашение выполнить вход с помощью соответствующей кнопки

OAuthCard.

8. Когда пользователь нажимает кнопку входа, служба канала открывает веб-браузер и вызывает внешнюю службу для загрузки страницы входа.
9. Пользователь выполняет вход на этой странице для внешней службы. Затем внешняя служба завершает обмен данными со службой токенов Bot Framework по протоколу OAuth и отправляет токен пользователя в службу токенов Bot Framework. Служба токенов Bot Framework безопасно сохраняет этот токен и отправляет действие боту с использованием этого токена.
10. Бот получает действие с токеном и может использовать его для создания вызовов к API MS Graph.

## Защита URL-адреса для входа

Настраивая упрощенную процедуру входа для пользователя с помощью службы Bot Framework, важно обеспечить защиту URL-адреса для входа. Пользователю предоставляется URL-адрес для входа, который связан с определенными идентификаторами диалога и пользователя для этого бота. Этот URL-адрес не следует предоставлять в общий доступ, так как это вызовет проблемы со входом в определенный диалог с ботом. Чтобы предотвратить атаки на систему безопасности, нацеленные на общее использование URL-адреса для входа, убедитесь, что пользователь, который переходит по этому URL-адресу на соответствующем компьютере, — это *владелец* окна диалога.

Некоторые каналы, такие как Microsoft Teams, Direct Line и Chat, могут сделать это без участия пользователя. Например, WebChat использует файлы cookie сеанса, чтобы убедиться, что поток входа выполнен в том же браузере, в котором ведется диалог WebChat. Другие каналы часто определяют пользователя по шестизначному коду. Это похоже настроенную многофакторную проверку подлинности, так как служба токенов Bot Framework не выдаст токен боту, пока пользователь не пройдет окончательную проверку подлинности, подтверждающую, что у выполнившего вход пользователя есть доступ к возможностям чата с использованием шестизначного кода.

### IMPORTANT

Учитывайте эти важные вопросы, связанные с безопасностью. См. сведения в следующей записи блога: [Using WebChat with Azure Bot Service Authentication](#) (Использование WebChat при проверке подлинности в службе Azure Bot).

## Следующие шаги

Теперь, когда вы знакомы с проверкой подлинности пользователей, давайте посмотрим, как применить его к роботу.

[Добавление аутентификации в веб-приложение](#)

## См. также раздел

- [Поставщики удостоверений](#)
- [Проверка подлинности соединителя REST](#)
- [Проверка подлинности REST Directline](#)

# Поставщики удостоверений

27.03.2021 • 12 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Поставщик удостоверений выполняет проверку подлинности удостоверений пользователя или клиента и выдает одноразовые маркеры безопасности. Он предоставляет проверку подлинности пользователей как услугу.

Клиентские приложения, например веб-приложения, делегируют аутентификацию доверенному поставщику удостоверений. В этом случае говорят, что клиентские приложения являются федеративными или используют федеративную идентификацию. Дополнительные сведения см. в статье [шаблон федеративного удостоверения](#).

Использование доверенного поставщика удостоверений обеспечивает следующее:

- Поддержка функций единого входа (SSO), которые позволяют приложению получать доступ к нескольким защищенным ресурсам.
- Упрощение установки подключений между пользователями и ресурсами облачных вычислений, что минимизирует необходимость в повторной проверке подлинности для пользователей.

## Единый вход

Единый вход относится к процессу проверки подлинности, который позволяет пользователю войти в систему один раз с одним набором учетных данных для доступа к нескольким приложениям или службам.

Пользователь входит в систему с одним набором идентификатора и пароля для использования любой из нескольких связанных программных систем. См. сведения о [едином входе](#).

Многие поставщики удостоверений поддерживают операцию выхода, которая отзывает маркер пользователя и отзывает доступ к связанным приложениям и службам.

### IMPORTANT

Единый вход повышает удобство использования, избавляя пользователей от необходимости многократного ввода учетных данных. Кроме того, он обеспечивает защиту, сокращая поверхность потенциальной атаки.

## Поставщик удостоверений Azure Active Directory

Azure Active Directory — это служба идентификации для платформы Microsoft Azure, которая предоставляет возможности управления удостоверениями и доступом. Она позволяет безопасно выполнять вход пользователей с помощью стандартных отраслевых протоколов, таких как OAuth 2.0.

Вы можете выбрать одну из двух реализаций поставщика удостоверений Active Directory с разными параметрами, как показано ниже.

### NOTE

Описанные здесь параметры используются при настройке **параметров подключения OAuth** в приложении регистрации бота Azure. Дополнительные сведения см. в разделе [Добавление проверки подлинности в Bot](#).

- [Azure AD версии 1](#)
- [Azure AD версии 2](#)

## AAD версии 1

Используйте параметры, показанные для настройки платформы разработчика Azure AD (v 1.0), также известной как конечная точка Azure AD **версии v1**. Это позволяет создавать приложения, которые безопасно подписывают пользователей с помощью рабочей или учебной учетной записи Майкрософт. См. сведения об [Azure Active Directory для разработчиков \(версия 1.0\)](#).

СВОЙСТВО	ОПИСАНИЕ	ЗНАЧЕНИЕ
<b>Имя:</b>	Имя подключения	<Имя подключения>
<b>Поставщик услуг</b>	Поставщик удостоверений Azure AD	<code>Azure Active Directory</code>
<b>Идентификатор клиента</b>	Идентификатор приложения поставщика удостоверений Azure AD	<Идентификатор приложения поставщика AAD>
<b>Секрет клиента</b>	Секрет приложения поставщика удостоверений Azure AD	<Секрет приложения поставщика AAD>
<b>Тип предоставления разрешения</b>		<code>authorization_code</code>
<b>Login URL (URL-адрес для входа)</b>		<code>https://login.microsoftonline.com</code>
<b>Идентификатор клиента</b>		<идентификатор каталога (арендатора)> или <code>common</code> . См. примечание.
<b>URL-адрес ресурса</b>		<code>https://graph.microsoft.com/</code>
<b>Области действия</b>		
<b>URL-адрес обмена токенами</b>	Используется для единого входа в Azure AD версии 2	

## Примечание

- Введите **идентификатор арендатора**, записанный для приложения поставщика удостоверений AAD, если вы выбрали один из следующих вариантов:
  - Учетные записи только в этом каталоге организации (*только Microsoft — один клиент*)
  - Учетные записи в любом каталоге организации (*каталог Microsoft AAD — мультитенантный*)
- Ведите `common`, если вы выбрали Учетные записи в любом каталоге организации (*любой каталог AAD — несколько клиентов*) и личные учетные записи Майкрософт (*например, Skype, Xbox, Outlook.com*). В противном случае приложение поставщика удостоверений AAD будет проверять клиент по выбранному для него идентификатору и исключать личные учетные записи Майкрософт.

Дополнительные сведения см. в разделе:

- Зачем выполнять обновление до платформы удостоверений Майкрософт (версия 2.0)?
- Платформа удостоверений Майкрософт (ранее — Azure Active Directory для разработчиков)

## Другие поставщики удостоверений

Azure поддерживает несколько поставщиков удостоверений. Вы можете получить полный список, а также связанные сведения, выполнив следующую команду консоли Azure:

```
az bot authsetting list-providers
```

Список этих поставщиков можно также просмотреть в [портал Azure](#) при определении параметров подключения OAuth для приложения регистрации Bot.

### New Connection Setting

Save Delete

Name \*

myoauthconnection

Service Provider \*

Adobe EchoSign

appFigures

AWeber

Azure Active Directory

Azure Active Directory v2

Basecamp

Bitly

Box

DocuSign

DropBox

Dynamics CRM Online

Facebook

Flickr

Generic Oauth 2

### Универсальные поставщики OAuth

Azure поддерживает универсальный протокол OAuth 2, позволяя вам использовать собственные поставщики удостоверений.

Вы можете выбрать одну из двух реализаций универсального поставщика удостоверений с разными параметрами, как показано ниже.

#### NOTE

Описанные здесь параметры используются при настройке **параметров подключения OAuth** в приложении регистрации бота Azure.

- Универсальный протокол OAuth 2
- Универсальный поставщик OAuth 2

### Универсальный протокол OAuth 2

Используйте этот поставщик для настройки любого универсального поставщика удостоверений OAuth 2, который функционально аналогичен поставщику Azure AD, в частности AD версии 2. Вы можете использовать ограниченное количество свойств, так как длина полезных данных в строке запроса и в тексте запроса ограничена. Для введенных значений можно увидеть, как параметры для различных URL-адресов, строк запросов и тела находятся в фигурных скобках {} .

Свойство	Описание	Значение
<b>Имя:</b>	Имя подключения	<Your name for the connection>
<b>Поставщик услуг</b>	Поставщик удостоверений	В раскрывающемся списке выберите <b>Универсальный протокол OAuth 2</b>
<b>Идентификатор клиента</b>	Идентификатор приложения поставщика удостоверений	<provider ID>
<b>Секрет клиента</b>	Секрет приложения поставщика удостоверений	<секрет поставщика>
<b>URL-адрес авторизации</b>		<a href="https://login.microsoftonline.com/common/oauth2/v2.0/authorize">https://login.microsoftonline.com/common/oauth2/v2.0/authorize</a>
<i>Строка запроса URL-адреса авторизации</i>		?client_id={ClientId}&response_type=code&redirect_uri={RedirectUrl}&scope={Scopes}&state={State}
<b>URL-адрес токена</b>		<a href="https://login.microsoftonline.com/common/oauth2/v2.0/token">https://login.microsoftonline.com/common/oauth2/v2.0/token</a>
<b>Текст токена</b>	Текст для отправки для обмена токенами	code={Code}&grant_type=authorization_code&redirect_uri={RedirectUrl}&client_id={ClientId}&client_secret={ClientSecret}
<b>URL-адрес обновления</b>		<a href="https://login.microsoftonline.com/common/oauth2/v2.0/token">https://login.microsoftonline.com/common/oauth2/v2.0/token</a>
<i>Шаблон текста обновления</i>	Текст для отправки с токеном обновления	refresh_token={RefreshToken}&redirect_uri={RedirectUrl}&grant_type=refresh_token&client_id={ClientId}&client_secret={ClientSecret}
<b>Области действия</b>	Список разрешений API, предоставленных ранее для приложения проверки подлинности Azure AD, с разделителями-запятыми	Такие значения, как,,,     И  

# Единый вход

27.10.2020 • 4 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Единый вход (SSO) позволяет клиенту, например виртуальному помощнику или каналу WebChat, взаимодействовать с ботом или навыком от имени пользователя. Сейчас поддерживается только поставщик удостоверений [Azure AD версии 2](#).

Единый вход используется в следующих сценариях:

- Виртуальный помощник обращается к одному или нескольким ботам навыков. Пользователь может выполнить вход в виртуальный помощник один раз. Затем этот помощник вызывает навыки от имени пользователя. См. сведения о [виртуальном помощнике](#).
- Веб-чат, внедренный в веб-сайт. Пользователь входит на веб-сайт. Затем веб-сайт вызывает бот или навык от имени пользователя.

Единый вход обеспечивает следующие преимущества:

- Пользователю не нужно снова входить в систему, если он уже вошел в виртуальный помощник или на веб-сайт.
- У виртуального помощника или веб-сайта нет данных о разрешениях пользователей.

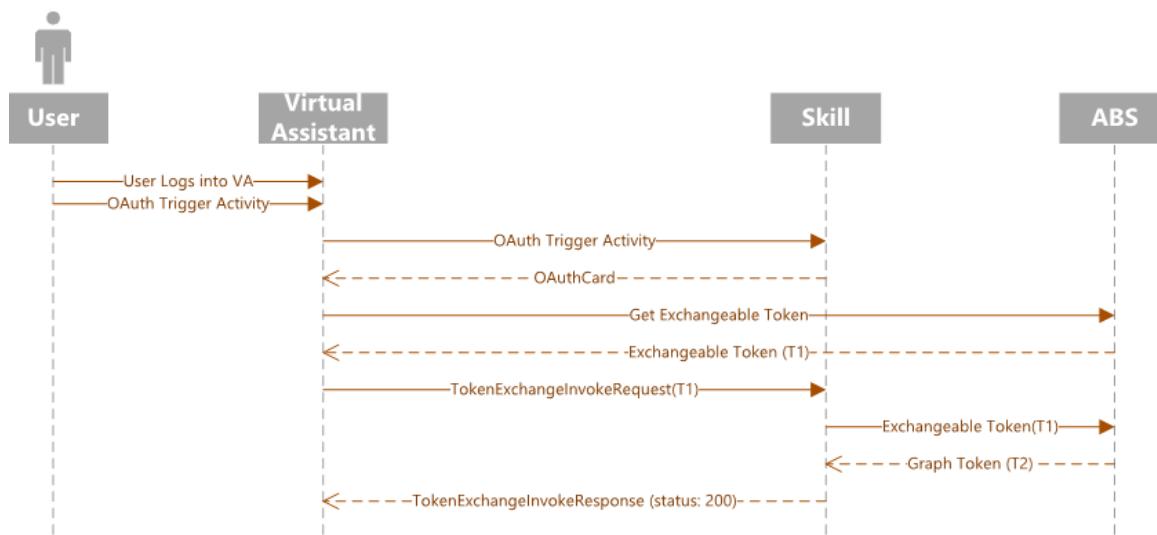
## NOTE

Функция единого входа появилась в пакете SDK для Bot Framework версии 4.8.

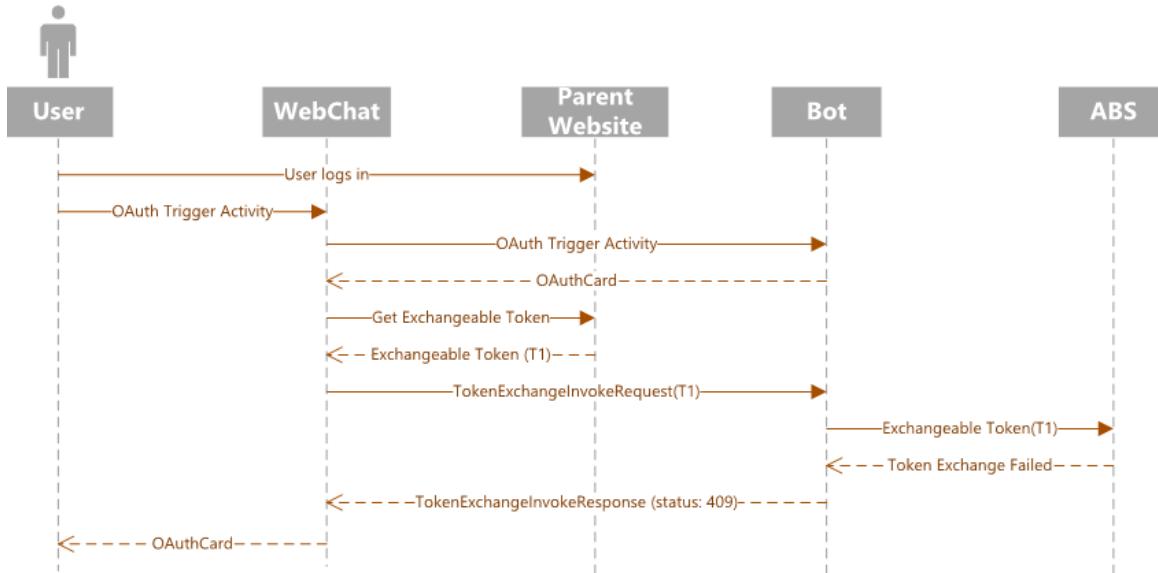
## Взаимодействие компонентов единого входа

На следующих схемах потоков показано взаимодействие между компонентами единого входа.

- На следующей схеме представлен обычный поток при использовании клиента виртуального помощника.



- Ниже показаны обычный и резервный потоки при использовании клиента WebChat.



В случае сбоя единый вход отображает карту OAuth. Ошибка может быть связана, например, с необходимостью получить согласие пользователя или со сбоем при обмене токенами.

Давайте проанализируем этот поток.

- Клиент начинает беседу с ботом, активируя сценарий OAuth.
- Бот отправляет этому клиенту карту OAuth.
- Клиент перехватывает карту OAuth. Прежде чем отображать ее пользователю, он проверяет наличие в ней свойства `TokenExchangeResource`.
- Если это свойство существует, клиент отправляет боту `TokenExchangeInvokeRequest`. Клиент должен предоставить для обмена токен пользователя, соответствующий токену Azure AD версии 2, аудитория которого соответствует свойству `TokenExchangeResource.Uri`. Клиент отправляет в бот действие `Invoke` с текстом, как показано ниже.

```
{
    "type": "Invoke",
    "name": "signin/tokenExchange",
    "value": {
        "id": "<any unique Id>",
        "connectionName": "<connection Name on the skill bot (from the OAuth Card)>",
        "token": "<exchangeable token>"
    }
}
```

- Бот обрабатывает `TokenExchangeInvokeRequest` и возвращает клиенту `TokenExchangeInvokeResponse`. Клиент должен дождаться получения `TokenExchangeInvokeResponse`.

```
{
    "status": "<response code>",
    "body": {
        "id": "<unique Id>",
        "connectionName": "<connection Name on the skill bot (from the OAuth Card)>",
        "failureDetail": "<failure reason if status code is not 200, null otherwise>"
    }
}
```

- Если `TokenExchangeInvokeResponse` имеет `status` со значением `200`, клиент не отображает карту OAuth. См. схему *обычного потока*. При любых других значениях `status` или при отсутствии

`TokenExchangeInvokeResponse` клиент отображает полученную карту OAuth пользователю. См. схему *резервного потока*. Это гарантирует, что поток единого входа применит обычный режим карты OAuth при любых ошибках или несоответствии зависимостей, таких как согласие пользователя.

## Дальнейшие действия

[Добавление функции единого входа в бота](#)

# Добавление проверки подлинности в бот

27.03.2021 • 38 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Пакет SDK для службы Azure Bot версии 4 упрощает разработку программы-роботы, которая может получать доступ к сетевым ресурсам, требующим проверки подлинности пользователей. Роботу не требуется управлять маркерами проверки подлинности, так как Azure позволяет использовать OAuth 2.0 для создания маркера на основе учетных данных каждого пользователя. Бот будет использовать маркеры, созданные в Azure, для доступа к нужным ресурсам. Таким образом, пользователю не нужно передавать идентификатор и пароль для доступа к защищенному ресурсу в бот, а только доверенному поставщику удостоверений.

Общие сведения о том, как платформа Bot обрабатывает такой тип проверки подлинности, см. в разделе [Проверка подлинности пользователя](#).

## NOTE

Проверка подлинности также работает с Bot Builder версии 3. Но в этой статье рассматривается только пример кода версии 4.

Эта статья содержит ссылки на два примера. Один из них демонстрирует, как получить маркер проверки подлинности. Другой является более сложным и показывает, как получить доступ к [Microsoft Graph](#) от имени пользователя. В обоих случаях маркер OAuth для бота можно получить, используя Azure Active Directory (AD) версии 1 или 2 в качестве поставщика удостоверений. В этой статье рассматриваются следующие вопросы:

- [Создание регистрации бота в Azure](#)
- [Создание поставщика удостоверений Azure AD](#)
- [Регистрация поставщика удостоверений Azure AD с помощью программы-робота](#)
- [Подготовка кода бота](#)

Завершив работу с этой статьей, вы получите готовый бот, который умеет выполнять несколько простых задач. В примере для Microsoft Graph можно отправить электронное сообщение, отобразить информацию о себе и проверить последние электронные сообщения. Вам не обязательно публиковать бота, чтобы протестировать возможности OAuth, но боту требуется допустимый идентификатор приложения Azure и пароль к нему.

## Рекомендации по веб-чатам и Direct Line

### IMPORTANT

Чтобы снизить риски безопасности при подключении к роботу с помощью элемента управления "веб-чат", необходимо использовать прямую линию с включенной расширенной проверкой подлинности.

Дополнительные сведения см. в разделе [Прямая строка Расширенная проверка подлинности](#).

## Предварительные требования

- Понимание принципов [работы ботов, управления состоянием, использования библиотек диалогов, реализации последовательного процесса общения](#), а также [повторного использования диалогов](#).

- Понимание процессов разработки для Azure и OAuth 2.0.
- Visual Studio 2017 или более поздней версии для платформы .NET.
- Node.js для JavaScript.
- Python 3.6 или 3.7 для Python.
- Один из перечисленных ниже примеров.

ОБРАЗЕЦ	ВЕРСИЯ BOT BUILDER	ЧТО ДЕМОНСТРИРУЕТ
<b>Проверка подлинности в C# или JavaScript или Python</b>	версия 4	Поддержка OAuthCard
<b>Мсграф проверки подлинности в C# или JavaScript или Python</b>	версия 4	Поддержка API Microsoft Graph с использованием OAuth 2

### Сведения об образцах

Чтобы выполнить указанные в этой статье примеры, вам потребуется следующее:

1. Приложение Azure Active Directory (AAD) для регистрации ресурса бота в Azure. Это приложение позволяет боту обращаться к внешнему защищенному ресурсу, например Microsoft Graph. Также оно позволяет пользователю взаимодействовать с ботом через несколько каналов, таких как Web Chat.
2. Отдельное приложение AAD в роли поставщика удостоверений. Это приложение предоставит учетные данные, необходимые для установки соединения OAuth между ботом и защищенным ресурсом. Обратите внимание, что в этой статье в качестве поставщика удостоверений используется Active Directory. Поддерживаются и другие поставщики.

#### IMPORTANT

Каждому зарегистрированному в Azure боту назначается приложение AAD. Но это приложение защищает доступ из канала к боту. Вам потребуются дополнительные приложения AAD для каждого внешнего защищенного ресурса, к которым бот должен обращаться от имени пользователя.

## Создание регистрации бота в Azure

В этом разделе показано, как зарегистрировать ресурс бота в Azure, чтобы разместить в нем код бота.

1. В браузере перейдите на [портал Azure](#).
2. В области слева выберите "Создать ресурс".
3. На панели справа выполните поиск типов ресурсов, в имени которых есть слово *бот*, и выберите **Регистрация каналов бота**.
4. Нажмите кнопку **Создать**.
5. В области **Регистрация канала бота** введите требуемые сведения. На следующем рисунке представлен пример.

## Bot Channels Registration

Bot Service

Bot handle \*

TestingBotAuth

Subscription \*

FUSE Temporary

Resource group \*

TestDeployment

Create new

Location \*

(US) West US 2

Pricing tier (View full pricing details)

S1 (1K Premium Msgs/Unit)

Messaging endpoint

https URL

Application Insights

On

Off

Microsoft App ID and password



Auto create App ID and password

6. Щелкните **Автоматически назначить код приложения и пароль** и выберите **Создать новый**.
7. Щелкните **Создать код приложения на портале регистрации приложений**. Откроется новая страница.
8. На странице **Регистрация приложений** щелкните **Новая регистрация** в верхнем левом углу.
9. Введите имя приложения бота, которое вы хотите зарегистрировать. В этой статье используется имя *TestingBotAuth*, но у каждого бота оно будет уникальным.
10. В поле **Поддерживаемые типы учетных записей** выберите вариант *Учетные записи в любом каталоге организации (любой каталог Azure AD — мультитенантный) и персональные учетные записи Майкрософт (например, Skype, Xbox)*.
11. Щелкните **Зарегистрировать**. После завершении Azure отобразит страницу сведений о регистрации приложения.
12. Скопируйте **Идентификатор приложения (клиента)** и сохраните его в файл.
13. На панели слева щелкните **Сертификаты и секреты**.
  - a. В разделе **Секреты клиента** щелкните **New client secret** (Новый секрет клиента).
  - b. Добавьте описание, чтобы отличать этот секрет от других секретов, которые могут вам понадобиться для создания этого приложения.
  - c. В поле **Срок действия истекает** укажите вариант **Никогда**.
  - d. Нажмите кнопку **Добавить**.
  - e. Скопируйте новый секрет клиента и сохраните его в файл.

#### WARNING

Храните этот секрет только для настройки бота. Не копируйте его копию, если у вас нет веских причин, в данном случае не оставляйте его в надежном месте.

14. Вернитесь к окну *Регистрация канала бота* и вставьте значения **Идентификатор приложения** и **Секрет клиента** в поля **Код приложения Майкрософт** и **Пароль** соответственно.

15. Нажмите кнопку **OK**.

16. Наконец, нажмите кнопку **Создать**.

#### NOTE

Вы назначите **идентификатор приложения (клиента)** и **секрет клиента**, сохраненный в файле, в переменные конфигурации Bot: `microsoftAppId` И `MicrosoftAppPassword`. См. раздел [Подготовка кода Bot](#).

Когда Azure завершит регистрацию бота, эта регистрация и служба приложения бота будут помещены в выбранную вами группу ресурсов.

## Служба удостоверений Azure AD

Облачная служба идентификации Azure Active Directory (Azure AD) позволяет создавать приложения с поддержкой безопасного входа пользователей по стандартным отраслевым протоколам, таким как OAuth 2.0.

Вы можете выбрать одну из двух служб идентификации:

1. Платформа разработчиков Azure AD (версия 1.0). Это конечная точка AAD **версии 1**, которая позволяет создавать приложения с поддержкой безопасного входа пользователей с рабочей или учебной учетной записью Майкрософт. См. сведения об [Azure Active Directory для разработчиков \(версия 1.0\)](#).
2. Платформа удостоверений Майкрософт (версия 2.0) Это конечная точка AAD **версии 2** — доработанная и улучшенная версия платформы AAD (версия 1.0). С ее помощью вы можете создавать приложения, которые поддерживают вход через любых поставщиков удостоверений Майкрософт и получение токенов для вызова таких программных API Майкрософт, как API Microsoft Graph или других API, созданных разработчиками. См. сведения о [платформе удостоверений Майкрософт \(версия 2.0\)](#).

См. сведения о различиях между конечными точками версий 1 и 2 в статье [Зачем выполнять обновление до платформы удостоверений Майкрософт \(версия 2.0\)?](#) См. сведения в статье [Платформа удостоверений Майкрософт \(ранее Azure Active Directory для разработчиков\)](#).

### Создание поставщика удостоверений Azure AD

В этом разделе показано, как создать поставщик удостоверений Azure AD, использующий OAuth2 для проверки подлинности Bot. Вы можете использовать конечные точки AAD версии 1 или 2.

#### TIP

Вам понадобится создать и зарегистрировать приложение Azure AD в клиенте, в котором можно дать согласие на делегирование разрешений, запрошенных приложением.

1. Откройте панель [Azure Active Directory](#) на портале Azure. Если вы не попадете сразу в нужный арендатор, щелкните действие **Переключить каталог**. (См. инструкции по [созданию арендатора](#)

с помощью портала.)

2. Откройте панель **Регистрация приложений**.
3. На панели **Регистрация приложений** щелкните **Новая регистрация**.
4. Заполните обязательные поля и создайте регистрацию приложения.
  - a. Присвойте имя приложению.
  - b. Выберите **Поддерживаемые типы учетных записей** для приложения. (Все эти параметры будут работать с этим примером.)
  - c. Для **URI перенаправления**:
    - a. Выберите **Интернет**.
    - b. Укажите для URL-адреса значение `https://token.botframework.com/.auth/web/redirect`.
  - d. Щелкните **Зарегистрировать**.
    - После создания в Azure отображается страница **Обзор** для приложения.
    - Запишите **идентификатор приложения (клиента)**. Это значение будет использоваться позже в качестве *идентификатора клиента* при создании строки подключения и регистрации поставщика Azure AD с регистрацией Bot.
    - Также запишите **идентификатор каталога (арендатора)**. Кроме того, вы будете использовать его для регистрации приложения поставщика с помощью программы Bot.
5. В области навигации щелкните **Сертификаты и секреты**, чтобы создать секрет для приложения.
  - a. В разделе **Секреты клиента** щелкните **New client secret** (Новый секрет клиента).
  - b. Добавьте описание, чтобы отличать этот секрет от других секретов, которые могут вам понадобиться для создания этого приложения, включая `bot login`.
  - c. Задайте для параметра **Срок действия истекает** значение **Никогда**.
  - d. Нажмите кнопку **Добавить**.
  - e. Перед закрытием этой страницы запишите секрет. Это значение вам нужно будет ввести позднее в поле *Секрет клиента* при регистрации приложения AAD в боте.
6. В области навигации щелкните **Разрешения API**, чтобы открыть панель **Разрешения API**. Рекомендуется явно задать разрешения API для приложения.
  - a. Щелкните **Добавить разрешение**, чтобы отобразить область **Запрос разрешений API**.
  - b. Для этого примера выберите API Microsoft и Microsoft Graph.
  - c. Выберите **Делегированные разрешения** и убедитесь, что выбраны все разрешения, требуемые для этого примера.

**NOTE**

Для любого разрешения, отмеченного как **Требуется согласие администратора**, необходимо, чтобы пользователь и администратор клиента вошли в систему. Поэтому для бота лучше использовать поменьше разрешений такого типа.

- openid
- profile
- Mail.Read
- Mail.Send

- User.Read
  - User.ReadBasic.All
- d. Щелкните **Добавить разрешения**. (При первом обращении пользователя к этому приложению через бота требуется предоставить согласие.)

Теперь приложение Azure AD настроено.

#### NOTE

При создании строки подключения и регистрации поставщика удостоверений с регистрацией Bot вы назначаете **идентификатор приложения (клиент)** и **секрет клиента**. См. следующий раздел.

### Регистрация поставщика удостоверений Azure AD с помощью программы-робота

Следующим шагом является регистрация только что созданного приложения AAD в боте.

- [Azure AD версии 2](#)
- [Azure AD версии 1](#)

#### AAD версии 2

1. Перейдите к странице регистрации каналов бота на [портале Azure](#).
2. Щелкните **Параметры**.
3. В разделе **OAuth Connection Settings** (Параметры подключения OAuth), который находится в нижней части страницы, щелкните **Добавить настройку**.
4. Заполните форму следующим образом.
  - a. **Имя**. Введите имя для подключения, которое было использовано в коде бота.
  - b. **Поставщик услуг**. Выберите **Azure Active Directory v2**. После выбора этого параметра появятся отдельные поля Azure AD.
  - c. **Идентификатор клиента**. Введите идентификатор приложения (клиента), записанный для поставщика удостоверений Azure AD v2.
  - d. **Секрет клиента**. Введите секрет, который вы записали для поставщика удостоверений Azure AD v2.
  - e. **URL-адрес обмена маркерами**. Оставьте это поле пустым, так как оно используется только для единого входа в Azure AD v2.
  - f. **Идентификатор клиента**. Введите **идентификатор каталога (клиента)**, записанный ранее для приложения AAD, или **Общие** в зависимости от поддерживаемых типов учетных записей, выбранных при создании приложения Azure AD. Чтобы решить, какое значение следует задать, руководствуйтесь следующими критериями:
    - Если при создании приложения AAD вы выбрали параметр *Учетные записи только в этом каталоге организации (только Майкрософт — один клиент)*, введите **идентификатор клиента**, который вы записали ранее для этого приложения AAD.
    - Если же вы выбрали *Учетные записи в любом каталоге организации (любой каталог Azure AD — мультитенантный и личные учетные записи Майкрософт (например, Xbox, Outlook.com))* или *Учетные записи в любом каталоге организации (любой каталог Azure AD — мультитенантный)*, введите слово **общие** вместо идентификатора клиента. В противном случае приложение AAD будет проверять клиент по выбранному для него идентификатору и исключать личные учетные записи Майкрософт.

Данный клиент будет связан с пользователями, которые могут пройти проверку

подлинности. Дополнительные сведения см. в [этой статье](#).

- g. В поле **области** введите имена разрешений, которые вы выбрали при регистрации приложения. В целях тестирования можно просто ввести: `openid profile`.

#### NOTE

Для приложения Azure AD версии 2 в поле **Области** принимаются списки разделенных пробелами значений с учетом регистра.

5. Выберите команду **Сохранить**.

#### NOTE

Используя API Microsoft Graph, эти значения позволяют приложению получать доступ к данным Office 365. Кроме того, поле **URL-адрес обмена маркерами** следует оставить пустым, так как оно используется только для единого входа в Azure AD версии 2.

### Тестирование подключения

1. Щелкните запись созданного подключения, чтобы открыть его.
2. Щелкните **Проверка подключения** в верхней части области Service Provider Connection Setting (Параметры подключения поставщика службы).
3. При выполнении данного действия в первый раз должна открыться новая вкладка браузера с перечисленными разрешениями, которые запрашивает приложение, и предложение их принять.
4. Нажмите кнопку **Принимаю**.
5. Вы перейдете на страницу **Проверка подключения к <your-connection-name> успешно выполнена**.

Для получения токенов пользователя можно использовать имя этого подключения в коде бота.

## Подготовка кода бота

Вам потребуются идентификатор приложения бота и пароль, чтобы выполнить этот процесс.

- [C#](#)
- [JavaScript](#)
- [Python](#)

1. Клонируйте нужный пример из репозитория GitHub: [Аутентификация бота](#) или [Аутентификация бота \(MSGraph\)](#).
2. Обновите файл `appsettings.json`:

- параметру `ConnectionName` присвойте значение имени подключения OAuth, которое вы добавили в бот;
- параметрам `MicrosoftAppId` И `MicrosoftAppPassword` присвойте значения идентификатора приложения бота и секрета приложения.

В зависимости от символов, из которых состоит секрет бота, может потребоваться XML-экранирование пароля. Например, символ амперсанда (&) потребуется кодировать как `&amp;`.

```
{  
    "MicrosoftAppId": "",  
    "MicrosoftAppPassword": "",  
    "ConnectionName": ""  
}
```

### 3. Обновление запуска. CS:

Чтобы использовать OAuth в *не общедоступных облачах Azure*, например в облаке для государственных организаций, необходимо добавить в файл следующий код `startup.cs`:

```
string uri = "https://api.botframework.azure.us";  
MicrosoftAppCredentials.TrustServiceUrl(uri);  
OAuthClientConfig.OAuthEndpoint = uri;
```

Чтобы получить **идентификатор приложения Microsoft** и значения **пароля Microsoft App**, см. статью [Получение пароля регистрации](#).

#### NOTE

Теперь код бота можно опубликовать в подписке Azure (щелкните проект правой кнопкой мыши и выберите **Опубликовать**), но для этой статьи это не требуется. Вам нужно настроить конфигурацию публикации, которая использует план приложения и размещения, с помощью которого выполнялась настройка бота на портале Azure.

## Тестирование программы-робота с помощью эмулятора

Установите [Bot Framework Emulator](#), если вы этого еще не сделали. См. также [Отладка с помощью эмулятора](#).

Чтобы имя входа в образце Bot работало, необходимо настроить эмулятор, как показано в [подокне Настройка эмулятора для проверки подлинности](#).

#### Тестирование

Когда вы настроите механизм аутентификации, можно протестировать пример бота.

#### NOTE

Возможно, вам будет предложено ввести *магический код*, так как реализуется пример бота. Этот магический код рассматривается в документе [RFC#7636](#) и предназначен для добавления дополнительного элемента безопасности. Удаление магического кода повышает риск возникновения угрозы безопасности. Это можно устраниТЬ с помощью прямой линии с включенной расширенной проверкой подлинности. Дополнительные сведения см. в статье о [расширенной проверке подлинности в Bot Framework](#).

1. Запустите пример бота на локальном компьютере.
2. Запустите эмулятор Bot Framework.
3. При подключении к боту необходимо указать идентификатор приложения бота и пароль.
  - Идентификатор приложения и пароль можно получить в области регистрации приложений Azure. Это те же значения, которые вы назначили боту в файле `appsettings.json` ИЛИ `.env`. В эмуляторе эти значения присваиваются в файле конфигурации или при первом подключении к Bot.
  - Вы также можете экранировать пароль (XML) в коде бота.
4. Чтобы просмотреть список доступных команд для бота и проверить функции проверки подлинности,

введите `help`.

5. После выполнения входа и до момента выхода не требуется повторно предоставлять учетные данные.
6. Чтобы выйти и отменить проверку подлинности, введите `logout`.

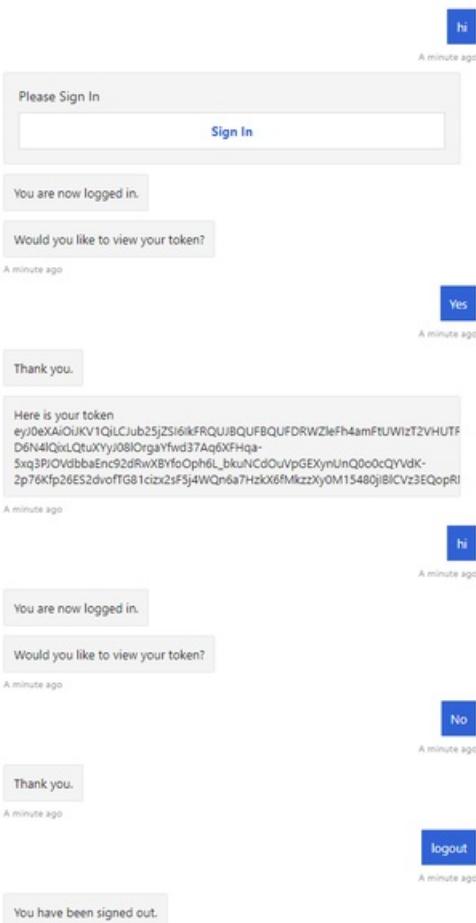
#### NOTE

Для использования проверки подлинности бота требуется служба Bot Connector. Эта служба использует сведения о регистрации каналов бота.

## Пример проверки подлинности

В примере **Аутентификация бота** диалог получает маркер пользователя после входа пользователя в систему.

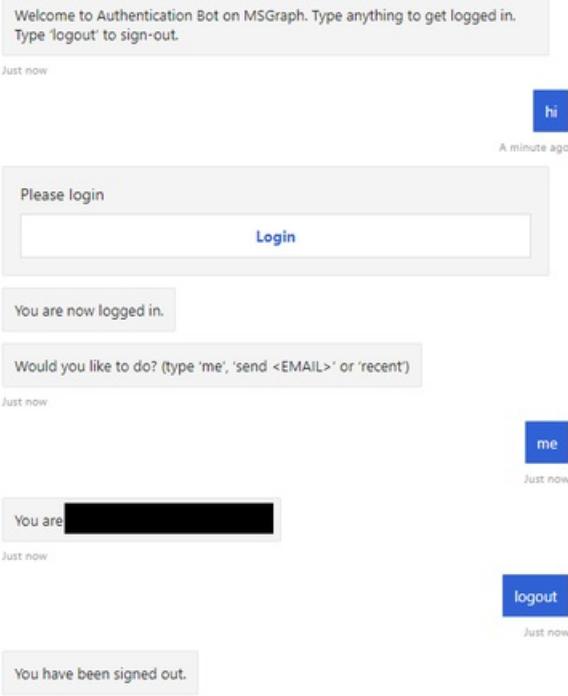
<http://localhost:3978/api/messages>



## Пример проверки подлинности Msграф

В примере **Аутентификация бота MSGraph** диалог принимает ограниченный набор команд после входа пользователя в систему.

<http://localhost:3978/api/messages>



## Дополнительные сведения

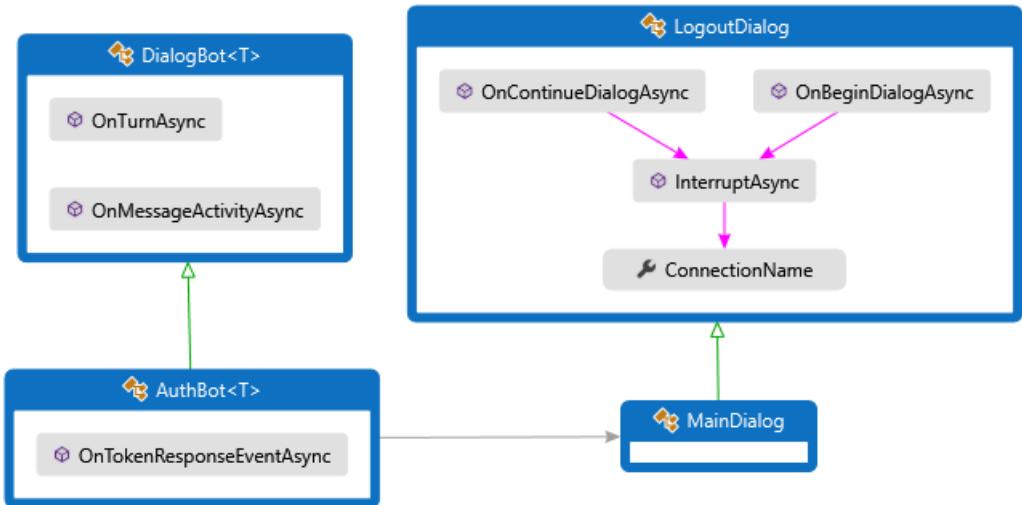
Когда пользователь приказывает боту выполнить некоторые действия, которые требуют пользовательского входа, бот может с помощью `OAuthPrompt` инициировать извлечение токена для данного соединения. `OAuthPrompt` создает поток получения маркера, который выполняет следующие действия.

1. Проверяет, имеет ли служба Azure Bot маркер для текущего сочетания пользователя и подключения.  
Если маркер есть, он возвращается.
2. Если служба Azure Bot не имеет нужного маркера в кэше, создается `OAuthCard` с кнопкой входа, на которую может нажать пользователь.
3. Когда пользователь нажимает на кнопку входа `OAuthCard`, служба Azure Bot выполняет одно из двух действий: напрямую отправляет боту маркер пользователя или предоставляет пользователю шестизначный код аутентификации для ввода в окно чата.
4. Если пользователю предоставлен код аутентификации, бот позднее обменивает этот код на маркер пользователя.

В следующих разделах объясняется, как этот пример реализует некоторые распространенные задачи проверки подлинности.

### Использование запроса OAuth для входа пользователя и получения маркера

- [C#](#)
- [JavaScript](#)
- [Python](#)



## Dialogs\MainDialog.cs

Добавьте запрос OAuth в `MainDialog` с помощью конструктора. Здесь мы получаем значение имени подключения из файла `appsettings.json`.

```

AddDialog(new OAuthPrompt(
    nameof(OAuthPrompt),
    new OAuthPromptSettings
    {
        ConnectionName = ConnectionName,
        Text = "Please Sign In",
        Title = "Sign In",
        Timeout = 300000, // User has 5 minutes to login (1000 * 60 * 5)
    }));

```

В шаге диалога укажите `BeginDialogAsync` для запуска командной строки OAuth, в которой пользователю будет предложено войти в систему.

- Если пользователь уже выполнил вход, будет создано событие ответа маркера без подтверждения пользователя.
- В противном случае пользователю будет предложено войти в систему. Служба Azure Bot отправляет событие ответа маркера после попытки входа пользователя.

```

return await stepContext.BeginDialogAsync(nameof(OAuthPrompt), null, cancellationToken);

```

В следующем шаге диалога проверьте наличие маркера в результате, полученном от предыдущего шага. Если он не равен NULL, значит пользователь успешно выполнил вход.

```

// Get the token from the previous step. Note that we could also have gotten the
// token directly from the prompt itself. There is an example of this in the next method.
var tokenResponse = (TokenResponse)stepContext.Result;

```

## Ожидание метода TokenResponseEvent

Когда вы запускаете командную строку OAuth, она ожидает события получения маркера, из которого затем извлекает маркер пользователя.

- C#

- [JavaScript](#)
- [Python](#)

## Bots\AuthBot.cs

AuthBot наследуется от `ActivityHandler` и явным образом обрабатывает действия ответа маркера. Здесь мы продолжаем активный диалог, что позволяет командной строке OAuth обработать событие и получить маркер.

```
protected override async Task OnTokenResponseEventAsync(ITurnContext<IEventActivity> turnContext,
CancellationToken cancellationToken)
{
    Logger.LogInformation("Running dialog with Token Response Event Activity.");

    // Run the Dialog with the new Token Response Event Activity.
    await Dialog.RunAsync(turnContext, ConversationState.CreateProperty<DialogState>(nameof(DialogState)),
cancellationToken);
}
```

## Выход пользователя

Мы рекомендуем предоставить пользователям возможность явно выходить из системы, не полагаясь на автоматический разрыв подключения при превышении времени ожидания.

- [C#](#)
- [JavaScript](#)
- [Python](#)

## Dialogs\LogoutDialog.cs

```
private async Task<DialogTurnResult> InterruptAsync(DialogContext innerDc, CancellationToken
cancellationToken = default(CancellationToken))
{
    if (innerDc.Context.Activity.Type == ActivityTypes.Message)
    {
        var text = innerDc.Context.Activity.Text.ToLowerInvariant();

        if (text == "logout")
        {
            // The bot adapter encapsulates the authentication processes.
            var botAdapter = (BotFrameworkAdapter)innerDc.Context.Adapter;
            await botAdapter.SignOutUserAsync(innerDc.Context, ConnectionName, null, cancellationToken);
            await innerDc.Context.SendActivityAsync(MessageFactory.Text("You have been signed out."),
cancellationToken);
            return await innerDc.CancelAllDialogsAsync(cancellationToken);
        }
    }

    return null;
}
```

## Добавление аутентификации Teams

В контексте OAuth Teams работает несколько иначе, чем другие каналы. Чтобы правильно реализовать аутентификацию, нужно внести несколько изменений. Мы добавим код из примера бота аутентификации Teams ([C#/JavaScript](#)).

Одно из различий между другими каналами и Teams заключается в том, что Teams отправляет в бот действие *invoke*, а не действие *event*.

- [C#](#)

- [JavaScript](#)
- [Python](#)

## Bots/TeamsBot.cs

```
protected override async Task OnTeamsSigninVerifyStateAsync(ITurnContext<IInvokeActivity> turnContext, CancellationToken cancellationToken)
{
    Logger.LogInformation("Running dialog with signin/verifystate from an Invoke Activity.");

    // The OAuth Prompt needs to see the Invoke Activity in order to complete the login process.

    // Run the Dialog with the new Invoke Activity.
    await Dialog.RunAsync(turnContext, ConversationState.CreateProperty<DialogState>(nameof(DialogState)), cancellationToken);
}
```

При использовании запроса *OAuth* это действие *invoke* должно быть переадресовано в диалог. Это будет сделано в `TeamsActivityHandler`. Добавьте в файл основного диалога следующий код.

- [C#](#)
- [JavaScript](#)
- [Python](#)

## Bots/DialogBot.cs

```
public class DialogBot<T> : TeamsActivityHandler where T : Dialog
```

Наконец, добавьте соответствующий файл `TeamsActivityHandler` (`TeamsActivityHandler.cs` для ботов C# `teamsActivityHandler.js` для ботов JavaScript) в папку бота самого верхнего уровня.

`TeamsActivityHandler` также отправляет действия *message reaction*. Действие *message reaction* ссылается на исходное действие с помощью поля *reply to ID*. Это действие также должно отображаться в [веб-канале активности](#) в Microsoft Teams.

### NOTE

Вам нужно создать манифест и включить `token.botframework.com` в раздел `validDomains`. В противном случае при нажатии кнопки **входа** в OAuthCard окно проверки подлинности не будет открываться. Создайте манифест с помощью [App Studio](#).

## Дополнительные материалы

- [Дополнительные ресурсы Bot Framework](#) содержат ряд ссылок с сопроводительной информацией.
- Репозиторий с [пакетом SDK Bot Framework](#) содержит дополнительные сведения о репозиториях, примерах, средствах и спецификациях для пакета SDK Bot Builder.

# Добавление функции единого входа в бот

27.03.2021 • 20 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье показано, как использовать функцию единого входа (SSO) в боте. В этом случае для взаимодействия с ботом *навыка* используется бот-получатель, также известный как *корневой бот*.

После входа в корневой бот пользователям не обязательно входить в каждый бот навыка, они могут использовать этих ботов с помощью корневого бота. Это связано с единым входом. Без него пользователям пришлось бы выполнять вход каждый раз, когда они общаются с другим ботом навыка.

## NOTE

Бот-получатель также называют *корневым* или *родительским* ботом. Бот *навыка* также называются *дочерним* ботом.

## Рекомендации по веб-чатам и Direct Line

### IMPORTANT

При использовании проверки подлинности службы Azure Bot для Web Chat важно учитывать несколько моментов, связанных с безопасностью. См. сведения о [безопасности](#) в руководстве по проверке подлинности REST.

## Предварительные требования

- Понимание основных принципов работы ботов, управления состоянием, библиотек диалогов, реализации последовательного процесса общения, а также повторного использования диалогов.
- Понимание процессов разработки для Azure и OAuth 2.0.
- Visual Studio 2017 или более поздней версии для платформы .NET.
- Node.js для JavaScript.
- Python 3.6 или 3.7 для Python.
- Примеры перечислены ниже.

ОБРАЗЕЦ	ВЕРСИЯ BOT BUILDER	ЧТО ДЕМОНСТРИРУЕТ
Единый вход с простым получателем навыков и навыком на CSharp	версия 4	Поддержка единого входа

## Сведения об образцах

Эта статья ссылается на два примера: RootBot и SkillBot. RootBot пересыпает действия в SkillBot. Они моделируют этот *типичный* *сценарий* использования навыка.

- Корневой бот вызывает один или несколько ботов навыка.
- Корневой бот и боты навыка реализуют обычную проверку подлинности, описанную в статье [Добавление проверки подлинности в бот](#).

- Пользователь входит в корневой бот.
- Благодаря единому входу и выполненному входу в корневой бот он входит в бот навыка автоматически.

Общие сведения о том, как платформа Bot обрабатывает аутентификацию, см. в разделе [Проверка подлинности пользователя](#). Общие сведения о едином входе доступны в разделе [Единый вход](#).

RootBot поддерживает единый вход пользователя. Он взаимодействует со SkillBot от имени пользователя без необходимости повторной аутентификации в SkillBot.

Для каждого проекта в примере необходимо следующее.

1. Приложение Azure AD для регистрации ресурса бота в Azure.
2. Приложение поставщика удостоверений Azure AD для аутентификации.

**NOTE**

Сейчас поддерживается только поставщик удостоверений [Azure AD версии 2](#).

## Создание регистрации RootBot в Azure

1. Создайте регистрацию бота на [портале Azure](#) для [RootBot](#). Выполните действия, описанные в разделе [Создание регистрации бота в Azure](#).
2. Скопируйте и сохраните **идентификатор приложения** и **секрет клиента** для регистрации бота.

## Создание удостоверения Azure AD для RootBot

Облачная служба идентификации Azure AD позволяет создавать приложения с поддержкой безопасного входа пользователей по стандартным отраслевым протоколам, таким как OAuth 2.0.

1. Создайте приложение удостоверений для [RootBot](#), которое использует Azure AD версии 2 для аутентификации пользователя. Выполните действия, описанные в разделе [Создание поставщика удостоверений Azure AD](#).
2. Нажмите кнопку **Манифест** в области слева.
3. Задайте для [accessTokenAcceptedVersion](#) значение 2.
4. Выберите команду **Сохранить**.
5. В области слева щелкните **Предоставление API**.
6. В области справа щелкните **Добавить область**.
7. В крайнем правом разделе *Добавить область* щелкните **Сохранить и продолжить**.
8. В появившемся окне в разделе *Кто может давать согласие?* выберите **Администраторы и пользователи**.
9. Введите остальные необходимые сведения.
10. Щелкните **Добавить область**.
11. Скопируйте и сохраните значение области.

### Создание параметров подключения OAuth

1. Создайте подключение Azure AD версии 2 в регистрации бота [RootBot](#) и введите значения, как описано в статье [Поставщики удостоверений](#), а также значение, описанное ниже.

2. Оставьте значение URL-адреса обмена маркерами пустым.
3. В поле **Области** введите значение области `RootBot`, сохраненное ранее.
4. Скопируйте и сохраните имя подключения.

## Создание регистрации SkillBot в Azure

1. Создайте регистрацию бота на [портале Azure](#) для `SkillBot`. Выполните действия, описанные в разделе [Создание регистрации бота в Azure](#).
2. Скопируйте и сохраните **идентификатор приложения** и **секрет клиента** для регистрации бота.

## Создание удостоверения Azure AD для SkillBot

Облачная служба идентификации Azure AD позволяет создавать приложения с поддержкой безопасного входа пользователей по стандартным отраслевым протоколам, таким как OAuth 2.0.

1. Создайте приложение удостоверений для `SkillBot`, которое использует Azure AD версии 2 для аутентификации бота. Выполните действия, описанные в разделе [Создание поставщика удостоверений Azure AD](#).
2. Нажмите кнопку **Манифест** в области слева.
3. Задайте для `accessTokenAcceptedVersion` значение 2.
4. Выберите команду **Сохранить**.
5. В области слева щелкните **Предоставление API**.
6. В области справа щелкните **Добавить область**.
7. В крайнем правом разделе *Добавить область* щелкните **Сохранить и продолжить**.
8. В появившемся окне в разделе *Кто может давать согласие?* выберите **Администраторы и пользователи**.
9. Введите остальные необходимые сведения.
10. Щелкните **Добавить область**.
11. Скопируйте и сохраните значение области.
12. Щелкните **Добавить клиентское приложение**. В крайнем правом разделе в поле **Идентификатор клиента** введите **идентификатор RootBot**, сохраненный ранее. Убедитесь, что используется идентификатор `RootBot`, а не идентификатор приложения регистрации.
13. В разделе *Authorized scope* (Разрешенная область) установите флажок для значения области.
14. Нажмите **Добавить приложение**.
15. Слева в области навигации щелкните **Разрешения API**. Рекомендуется явно задать разрешения API для приложения.
  - a. В области справа щелкните **Добавить разрешение**.
  - b. Выберите API **Майкрософт**, затем щелкните Microsoft Graph.
  - c. Выберите **Делегированные разрешения** и убедитесь, что выбраны все разрешения. Ниже перечислены разрешения, требуемые для этого примера.

#### NOTE

Для любого разрешения, отмеченного как **Требуется согласие администратора**, необходимо, чтобы пользователь и администратор арендатора вошли в систему.

- openid
- profile
- User.Read
- User.ReadBasic.All

d. Щелкните **Добавить разрешения**.

#### Создание параметров подключения OAuth

1. Создайте подключение Azure AD версии 2 в регистрации бота `Skillbot` и введите значения, как описано в статье [Поставщики удостоверений](#), а также значения, описанные ниже.
2. В поле **Token Exchange URL** (URL-адрес обмена маркерами) введите значение области `SkillBot`, сохраненное ранее.
3. В поле **Области** введите следующие значения, разделенные пробелом: `profile User.Read User.ReadBasic.All openid`.
4. Скопируйте и сохраните имя файла подключения.

## Проверка подключения

1. Щелкните запись созданного подключения, чтобы открыть его.
2. Щелкните **Проверка подключения** в верхней части области **Service Provider Connection Setting** (Параметры подключения поставщика службы).
3. При выполнении данного действия в первый раз должна открыться новая вкладка браузера с перечисленными разрешениями, которые запрашивает приложение, и предложение их принять.
4. Нажмите кнопку **Принимаю**.
5. Вы перейдете на страницу **Проверка подключения к <your-connection-name> успешно выполнена**.

Дополнительные сведения см. в разделах [Общие сведения об Azure Active Directory \(версии 1.0\) для разработчиков](#) и [Общие сведения о платформе удостоверений Майクロфорт версии 2.0](#). См. сведения о различиях между конечными точками версий 1 и 2 в статье [Зачем выполнять обновление до платформы удостоверений Майкрофорт \(версия 2.0\)?](#) См. сведения в статье [Платформа удостоверений Майкрофорт \(ранее Azure Active Directory для разработчиков\)](#).

## Подготовка кода примеров

Необходимо обновить файл `appsettings.json` в обоих примерах, как описано ниже.

- **C#**
  1. В репозитории GitHub клонируйте пример использования [единого входа с простым получателем навыков и навыком](#).
  2. Откройте файл `appsettings.json` проекта `SkillBot`. В сохраненном файле назначьте приведенные ниже значения.

```
{  
    "MicrosoftAppId": "<SkillBot registration app ID>",  
    "MicrosoftAppPassword": "<SkillBot registration password>",  
    "ConnectionName": "<SkillBot connection name>",  
    "AllowedCallers": [ "<RootBot registration app ID>" ]  
}
```

3. Open the `RootBot` project `appsettings.json` file. From the saved file, assign the following values:

```
{  
    "MicrosoftAppId": "<RootBot registration app ID>",  
    "MicrosoftAppPassword": "<RootBot registration password>",  
    "ConnectionName": "<RootBot connection name>",  
    "SkillHostEndpoint": "http://localhost:3978/api/skills/",  
    "BotFrameworkSkills": [  
        {  
            "Id": "SkillBot",  
            "AppId": "<SkillBot registration app ID>",  
            "SkillEndpoint": "http://localhost:39783/api/messages"  
        }  
    ]  
}
```

## Тестирование примеров

Для тестирования используйте следующее.

- Команды `RootBot`.
  - Команда `login` позволяет пользователю войти в регистрацию Azure AD с помощью `RootBot`. После входа пользователя функция единого входа выполняет вход и в `SkillBot`. Пользователю не нужно повторно входить в систему.
  - Команда `token` отображает маркер пользователя.
  - Команда `logout` выполняет выход пользователя из `RootBot`.
- Команды `SkillBot`.
  - Команда `skill login` позволяет `RootBot` войти в `SkillBot` от имени пользователя. Пользователь не отображается на карте входа, если он уже вошел в систему, а единственный вход сработал без сбоя.
  - Команда `skill token` отображает маркер пользователя из `SkillBot`.
  - Команда `skill logout` выполняет выход пользователя из `SkillBot`.

### NOTE

При первом использовании единого входа пользователям может быть предложено выполнить вход с помощью карты OAuth. Это связано с тем, что они еще не дали согласие для приложения Azure AD навыка. Чтобы избежать этого, они могут предоставить согласие администратора для любых разрешений графа, запрашиваемых приложением Azure AD.

- Эмулятор
- Веб-чат

## Тестирование с помощью эмулятора

Установите [Bot Framework Emulator](#), если вы этого еще не сделали. См. также [Отладка с помощью](#)

эмулятора.

Чтобы имя входа в образце Bot работало, необходимо настроить эмулятор, как показано в [подокне Настройка эмулятора для проверки подлинности](#).

Когда вы настроите механизм аутентификации, можно протестировать пример бота.

1. В Visual Studio откройте решение `ssowithSkills.sln` и настройте его для запуска [отладки с несколькими процессами](#).
2. Начните отладку на локальном компьютере. Обратите внимание на то, что в файле `appsettings.json` проекта `RootBot` имеются следующие параметры.

```
"SkillHostEndpoint": "http://localhost:3978/api/skills/"  
"SkillEndpoint": "http://localhost:39783/api/messages"
```

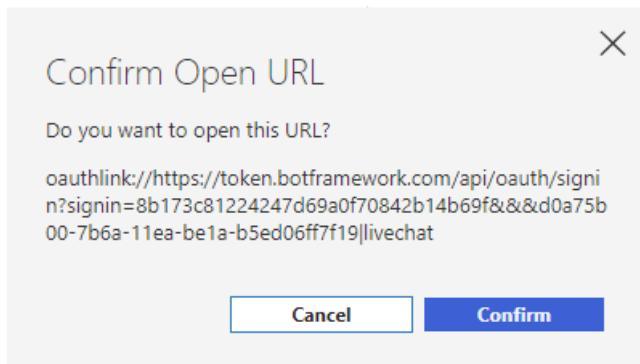
#### NOTE

Эти параметры подразумевают, что на локальном компьютере будет выполняться `RootBot`, и `SkillBot`. Эмулятор взаимодействует с `RootBot` через порт 3978 и `RootBot` взаимодействует с `SkillBot` портом 39783. Как только вы начнете отладку, откроются два окна браузера по умолчанию. Одно будет использовать порт 3978, а другое — порт 39783.

1. Запустите эмулятор Bot Framework.
2. При подключении к боту необходимо указать идентификатор приложения регистрации `RootBot` и пароль.
3. Введите `hi`, чтобы начать беседу.
4. Введите `login`. `RootBot` отобразит карту аутентификации *Sign In to AAD* (Вход в AAD).



5. Щелкните **Войти**. Откроется всплывающее диалоговое окно *Confirm Open URL* (Подтверждение открытия URL-адреса).



6. Щелкните **Confirm** (Подтвердить). Вы войдете в систему, и отобразится маркер `RootBot`.
7. Введите `token`, чтобы снова отобразить маркер.



Here is your token:  
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6IkN0VHV0TUptRDVNN0RMZH  
pEMnYyeDNRS1NSWS.9eyJhdWQiOiJhYzI2OGNjYy0zYTlLTQ0YTItYThjNy1INzk  
5NTjODg4MjMiLCJpc3MiOiJodHRwczovL2xvZ2luLm1p3Jvc29mdG9ubGluZS5j

Теперь вы можете взаимодействовать с `SkillBot`. После выполнения входа с помощью `RootBot` и до момента выхода не потребуется повторно предоставлять учетные данные. Это означает, что единый вход работает.

8. Введите **навык входа** в поле эмулятора. Вам не будет предложено еще раз выполнить вход.

Вместо этого отобразится маркер SkillBot.

9. Теперь введите `skill login`, чтобы снова отобразить маркер.

Just now



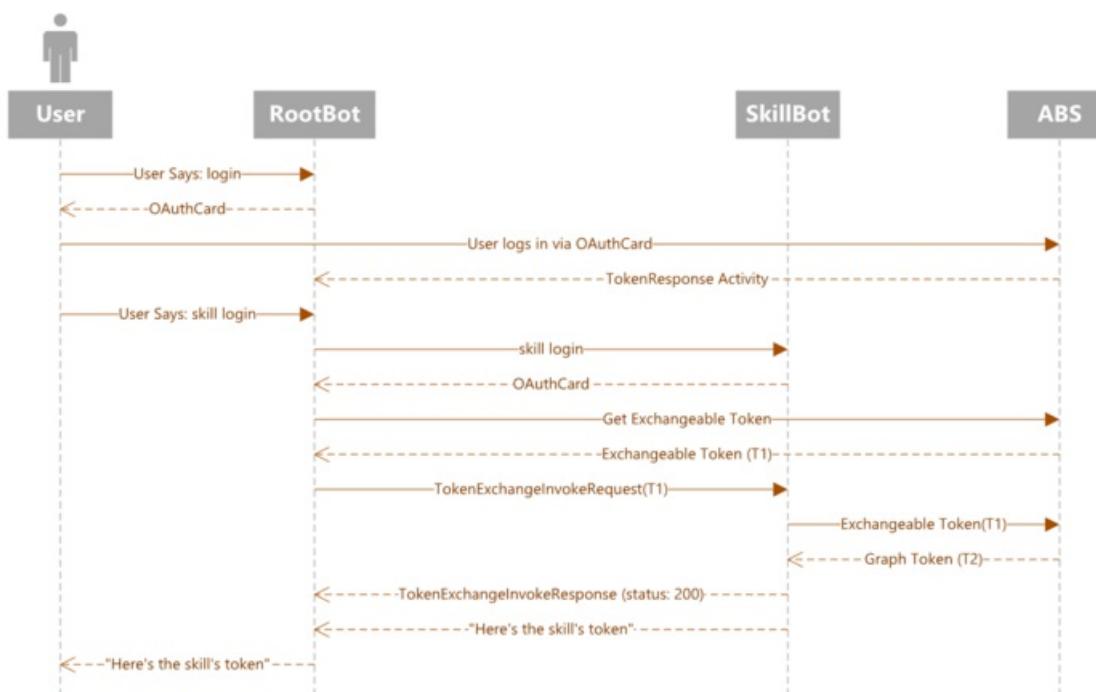
Just now

Here is your token for the skill:  
eyJ0eXAiOiJKV1QiLCJub25jZSI6Im92T2I2TjVpUkhaLU15dTIPcz  
N2VjRIOWxYMVp5VDILYU0QndldmpHbmMiLCJhbGciOiJSUzI  
1NiIsIng1dCI6IiINRUxIVDBndmlwbXhvU0RvWWZvbWpxZmpZ  
VSIsImtpZCI6IiINRUxIVDBndmlwbXhvU0RvWWZvbWpxZmpZ

10. Теперь вы можете ввести `skill logout`, чтобы выйти из `SkillBot`. Затем введите `logout`, чтобы выйти из `SimpleRootBoot`.

## Дополнительные сведения

Приведенная ниже схема последовательности операций относится к примерам, используемым в этой статье, и показывает взаимодействие между различными компонентами. *ABS* — это *Служба Azure Bot*.



- Пользователь вводит команду `login` для `RootBot` в первый раз.
- `RootBot` отправляет карту `OAuthCard`, предлагая пользователю выполнить вход.

3. Пользователь вводит учетные данные для аутентификации, которые отправляются ABS (Службу Azure Bot).
4. ABS отправляет маркер аутентификации, созданный на основе учетных данных пользователя, в RootBot.
5. RootBot отображает маркер корневого бота пользователю.
6. Пользователь вводит команду `skill login` для SkillBot.
7. SkillBot отправляет OAuthCard в RootBot.
8. RootBot запрашивает **заменяемый маркер** из ABS.
9. На этом этапе начинается процесс единого входа, по завершении которого **маркер навыка** передается из SkillBot в RootBot.
10. RootBot отображает маркер навыка пользователю. Обратите внимание на то, что маркер навыка был создан без входа пользователя в SkillBot. Это возможно благодаря единому входу.

Чтобы увидеть, как происходит обмен маркерами, ознакомьтесь с примером ниже. Соответствующую функцию можно найти в [TokenExchangeSkillHandler.cs](#).

- [C#](#)

```

private async Task<bool> InterceptOAuthCards(ClaimsIdentity claimsIdentity, Activity activity)
{
    var oauthCardAttachment = activity.Attachments?.FirstOrDefault(a => a?.ContentType ==
OAuthCard.ContentType);
    if (oauthCardAttachment != null)
    {
        var targetSkill = GetCallingSkill(claimsIdentity);
        if (targetSkill != null)
        {
            var oauthCard = ((JObject)oauthCardAttachment.Content).ToObject<OAuthCard>();

            if (!string.IsNullOrWhiteSpace(oauthCard?.TokenExchangeResource?.Uri))
            {
                using (var context = new TurnContext(_adapter, activity))
                {
                    context.TurnState.Add<IIIdentity>("BotIdentity", claimsIdentity);

                    // AAD token exchange
                    try
                    {
                        var result = await _tokenExchangeProvider.ExchangeTokenAsync(
                            context,
                            _connectionName,
                            activity.Recipient.Id,
                            new TokenExchangeRequest() { Uri = oauthCard.TokenExchangeResource.Uri
                        }).ConfigureAwait(false);

                        if (!string.IsNullOrEmpty(result?.Token))
                        {
                            // If token above is null, then SSO has failed and hence we return false.
                            // If not, send an invoke to the skill with the token.
                            return await SendTokenExchangeInvokeToSkill(activity,
                                oauthCard.TokenExchangeResource.Id, result.Token, oauthCard.ConnectionName, targetSkill,
                                default).ConfigureAwait(false);
                        }
                    }
                    catch
                    {
                        // Show oauth card if token exchange fails.
                        return false;
                    }
                }
            }
        }
    }
    return false;
}

```

## Дополнительные материалы

- Общие сведения о [едином входе](#).
- [Поставщики удостоверений](#) обеспечивают проверку подлинности пользователей в качестве службы
- [Проверка подлинности пользователя](#) для доступа к защищенным сетевым ресурсам от имени пользователя

# Прямая строка Расширенная проверка подлинности

27.03.2021 • 6 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье описываются потенциальные угрозы безопасности при подключении пользователей к роботу, например с помощью элемента управления "веб-чат". Кроме того, в нем отображаются решения по устранению проблем с использованием параметров прямой строки [расширенной проверки подлинности](#) и безопасной обработки [кода пользователя](#).

Необходимо иметь в виду, что существует два удостоверения пользователя:

- Удостоверение пользователя канала. Злоумышленник может использовать его для [олицетворения](#).
- Удостоверение пользователя от поставщика удостоверений, которое использует Bot для проверки подлинности пользователя. Злоумышленник может использовать его для [подмены удостоверения](#).

## Олицетворение

Олицетворение относится к действию злоумышленника, который делает робот, который посчитает его чужим. Например, при обращении в Интернет-чат злоумышленник может олицетворять кого-то еще, **изменяя идентификатор пользователя** для экземпляра Web Chat.

### Устранение проблем олицетворения

- Сделайте **идентификатор пользователя** неполным.
- [Подключите робот к прямой линии](#).
- Включите [расширенную проверку подлинности](#) канала прямой линии, чтобы служба Azure Bot смогла дальше обнаруживать и отклонять любые изменения, внесенные в идентификатор пользователя. Это означает, что идентификатор пользователя (`Activity.From.Id`) в сообщениях от прямой линии к Bot всегда будет таким же, как и тот, который использовался для инициализации элемента управления "веб-чат".

#### NOTE

Прямая линия создает **маркер** на основе прямой секретной строки и внедряет `User.ID` в маркер. Он гарантирует, что сообщения, отправленные в Bot, `User.ID` как `from.ID` действия. Если клиент отправляет сообщение прямой строке с другим `from.ID`, он будет заменен на **идентификатор, внедренный в маркер** перед пересылкой сообщения в Bot. Поэтому нельзя использовать другой идентификатор пользователя после инициализации секрета канала с этим ИДЕНТИФИКАТОРом.

Для этого компонента требуется запуск идентификатора пользователя `d1_`, как показано ниже:

```

public class HomeController : Controller
{
    private const string secret = "<TODO: DirectLine secret>";
    private const string dlUrl = "https://directline.botframework.com/v3/directline/tokens/generate";

    public async Task<ActionResult> Index()
    {
        HttpClient client = new HttpClient();
        var userId = $"dl_{Guid.NewGuid()}";

        HttpRequestMessage request = new HttpRequestMessage(HttpMethod.Post, dlUrl);
        request.Headers.Authorization = new AuthenticationHeaderValue("Bearer", secret);
        request.Content = new StringContent(
            JsonConvert.SerializeObject(
                new { User = new { Id = userId } }),
            Encoding.UTF8,
            "application/json");

        var response = await client.SendAsync(request);

        string token = String.Empty;
        if (response.IsSuccessStatusCode)
        {
            var body = await response.Content.ReadAsStringAsync();
            token = JsonConvert.DeserializeObject<DirectLineToken>(body).token;
        }

        var config = new ChatConfig()
        {
            Token = token,
            UserId = userId
        };

        return View(config);
    }
}

```

Созданный токен на основе прямой секрет строки затем используется в элементе управления веб-чата, как показано ниже:

```

@model Bot_Auth_DL_Secure_Site_MVC.Models.ChatConfig
 @{
     ViewData["Title"] = "Home Page";
 }
 <div id="webchat" role="main" />
 <head>
     <script src="https://cdn.botframework.com/botframework-webchat/latest/webchat.js"></script>
 </head>
 <body>
     <script>
         window.WebChat.renderWebChat({
             directLine: window.WebChat.createDirectLine({ token: '@Model.Token' }),
             userID: '@Model.UserId'
         }, document.getElementById('webchat'));
     </script>
 </body>

```

## Подмена удостоверения

Подмена удостоверения означает действие злоумышленника, принимающего удостоверение законного пользователя, а затем использующее это удостоверение для выполнения вредоносных целей.

Когда программа-робот запрашивает у пользователя канал А вход в поставщик удостоверений, процесс входа должен гарантировать, что пользователь А является единственным, который подписывается в поставщик. Если другому пользователю также разрешено входить в поставщик, он будет иметь доступ к ресурсам пользователя через Bot.

### Устранение рисков для подмены удостоверения пользователя

В элементе управления "веб-чат" есть два механизма, которые гарантируют, что пользователь вошел в подходящую учетную запись.

- Волшебный код.** В конце процесса входа пользователю предоставляется случайный код из 6 цифр (**волшебный код**). Пользователь должен ввести этот код в диалоге, чтобы завершить процесс входа. Это приводит к неправильному опыту пользователя. Кроме того, он обычно подвергается фишинговым атакам. Злоумышленник может обманным путем вынудить другого пользователя выполнить вход, чтобы получить магический код путем фишинга.

#### WARNING

Использование волшебного кода является устаревшим. Вместо этого рекомендуется использовать **прямую линейную проверку подлинности**, описанную ниже.

- Прямая строка Расширенная проверка подлинности.** Из-за проблем с методом **волшебного кода** служба Azure Bot удалила свою потребность. Служба Azure Bot гарантирует, что процесс входа может выполняться только **в том же сеансе браузера**, в котором запущен экземпляр Web Chat. Чтобы включить эту защиту, необходимо запустить веб-чат с **прямым маркером**, который содержит **список доверенных доменов, на которых может размещаться клиент Интернет-чата**. С помощью расширенных параметров проверки подлинности можно статически указать список доверенных доменов (доверенных источников) на странице конфигурации с прямой линией. См. раздел [Настройка расширенной проверки подлинности](#).

# Рекомендации по безопасности Bot Framework

27.03.2021 • 10 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Использование ботов становится все более распространенным в ключевых сферах бизнеса, таких как финансовые услуги, розничная торговля, туризм и т. д. Bot может получать конфиденциальные данные, такие как кредитные карты, SSN, банковские счета и другие персональные данные. Поэтому важно обеспечить безопасность и защиту программы-робота от распространенных угроз и уязвимостей.

Для повышения безопасности программы-робота можно использовать стандартные предупредительные меры. Некоторые меры безопасности похожи на те, которые используются в других программных системах, в то время как некоторые из них относятся к платформе Bot. Последние сведения см. в статье о [производительности системы безопасности Azure](#). В этом тесте содержатся рекомендации по защите облачных решений в Azure.

## Проблемы безопасности в двух словах

В этой статье вопросы безопасности группируются в две категории:

- **Угрозы.** тактика, которую кто-то может использовать для нарушения работы программы-робота, например подмена, искажение, неразглашение информации, отказ в обслуживании и т. д.
- **Уязвимости.** способы, с помощью которых робот или управление программой Bot могут быть подвержены такой тактике, как ошибки или слабая безопасность.

Сокращение уязвимостей — хороший способ устраниć угрозы, а известное средство сокращения уязвимостей — реализовать контрольные точки безопасности в процессе разработки и развертывания.

## Общие рекомендации по безопасности

В следующих областях рассматриваются некоторые стандартные рекомендации по обеспечению безопасности, общие для всех приложений.

### Протоколы и шифрование

Данные могут быть изменены во время передачи. Существуют протоколы, которые обеспечивают шифрование для избежания проблем неправильного использования и незаконного изменения. Поэтому боты должны передавать данные только через зашифрованные каналы. Это затрудняет для всех, кроме получателя и отправителя, просмотр какой либо части сообщения или транзакции.

Шифрование является одним из наиболее надежных методов обеспечения безопасности Bot и компаний, которые должны заблаговременно гарантировать ее эффективность, принимая меры для персонализации и шифрования конфиденциальных данных.

Для обмена данными по сети любая защищенная система должна использовать протокол HTTPS , который передает данные по протоколу HTTP в зашифрованных подключениях , [защищенных с помощью](#) протокола TLS или SSL (SSL). См. также [RFC 2818-HTTP через TLS](#).

### Самостоятельно деструктурировать сообщения

Безвозвратно удаляйте конфиденциальные данные, как только они больше не нужны, обычно после завершения обмена сообщениями или через определенное время. Сюда могут входить личные сведения, идентификаторы, ПИН-коды, пароли, контрольные вопросы и ответы и т. д.

## **Хранилище данных**

Рекомендации по хранению информации в безопасном состоянии на определенный период времени и их отмене позже после обслуживания.

Ниже перечислены некоторые распространенные методы обеспечения безопасности.

### **Брандмауэры базы данных**

- По умолчанию брандмауэры запрещают доступ к трафику. Допустимый трафик должен исходить из конкретных приложений или веб-серверов, которым требуется доступ к данным.
- Также следует развернуть брандмаэр веб-приложения. Это обусловлено тем, что атаки, такие как атаки путем внедрения кода SQL, направленные на веб-приложение, можно использовать для захвата или удаления данных из базы данных.

### **Усиление защиты базы данных**

- Убедитесь, что база данных по-прежнему поддерживается поставщиком, и вы используете последнюю версию базы данных со всеми установленными исправлениями безопасности для удаления известных уязвимостей.
- Удалите или отключите все ненужные функции или службы и убедитесь, что пароли всех учетных записей по умолчанию изменены. или более того, удалите все ненужные учетные записи по умолчанию.
- Убедитесь, что включены все элементы управления безопасностью базы данных, предоставляемые базой данных, если нет конкретной причины для отключения.

### **Сократите ценную информацию**

- Убедитесь, что не хранятся конфиденциальные сведения, которые не должны находиться в базе данных.
- Данные, сохраняемые для соответствия или другие цели, можно переместить в более безопасное хранилище (возможно, вне сети), что менее уязвимо для угроз безопасности базы данных.
- Не забудьте удалить все файлы журнала, записанные сервером во время исходной процедуры установки. Если установка прошла успешно, эти файлы не имеют значения, но могут содержать сведения, которые потенциально могут быть использованы.

## **Образование**

Программы-роботы предоставляет инновационное средство взаимодействия между компанией и ее клиентами. Но они потенциально могут предоставить программы трояны для изменения веб-сайта компании. Таким образом, компания должна гарантировать, что ее разработчики понимают важность программы Bot-безопасности в рамках обеспечения безопасности сайта. Более того, ошибки пользователей могут быть проблемой. Для этого потребуется некоторое образование, как можно безопасно использовать программы-роботы, например:

- Для разработчиков стратегия должна включать в себя внутреннее обучение использованию программы Bot в безопасном режиме.
- Клиентам могут быть предоставлены рекомендации о безопасном взаимодействии с Bot.

## **Специальные рекомендации по безопасности для Bot**

В следующих областях рассматриваются некоторые стандартные рекомендации по обеспечению безопасности для приложений-Bot Framework. Следующие рекомендации описывают Рекомендуемые меры безопасности для платформы Bot. Дополнительные сведения см. в разделе [вопросы и ответы по безопасности и конфиденциальности](#).

### **Проверка подлинности соединителя Bot**

Служба соединителя Bot изначально использует протокол HTTPS для обмена сообщениями между Bot и каналами (пользователями). Пакет SDK для Bot Framework автоматизирует простую проверку

подлинности на основе ленты в канале.

#### WARNING

Очень важно сделать это правильно. Реализуя все шаги, описанные в статье о проверке подлинности, можно снизить риск того, что злоумышленник сможет читать сообщения, отправленные на ваш робот, отправлять сообщения, имитирующие программы-роботы, и украсть секретные ключи.

## Аутентификация пользователей

**Функция проверки подлинности службы Azure Bot** позволяет проверять подлинность пользователей и получать маркеры доступа от различных поставщиков удостоверений, например *Azure Active Directory, GitHub, Uber* и т. д. Кроме того, настроить проверку подлинности можно для пользовательского поставщика удостоверений OAuth2. Все это позволяет использовать только **один фрагмент кода проверки подлинности**, который будет работать для всех поддерживаемых поставщиков удостоверений и всех каналов. Чтобы использовать эти возможности, выполните следующие действия:

1. Статически настройте `settings` в боте, который содержит сведения о регистрации приложения у поставщика удостоверений.
2. Используйте `OAuthCard`, указав сведения о приложении, которые используются для входа пользователя и были указаны на предыдущем шаге.
3. Получите маркеры доступа с помощью API **службы Azure Bot**. Рекомендуется ограничить время, в течение которого прошедший проверку подлинности пользователь может оставаться *в системе*.

Дополнительные сведения см. в статье [Проверка подлинности пользователя](#).

### Веб-чат.

При использовании элемента управления "веб-чат" необходимо помнить о некоторых важных вопросах, касающихся олицетворения и подмены удостоверений. Дополнительные сведения см. в разделе [Прямая строка Расширенная проверка подлинности](#).

## Дополнительные сведения

- [Проверка подлинности пользователей](#)
- [Добавление аутентификации для бота с помощью службы Azure Bot](#)
- [Включить безопасность и тестирование на localhost](#)
- [Секреты и маркеры](#)
- [Технологии проверки подлинности](#)
- [Улучшенные функции прямой проверки подлинности](#)
- [Рекомендации по безопасности в Центре безопасности Azure](#)
- [Защита от угроз с помощью Центра безопасности Azure](#)
- [Azure Security Center Data Security](#) (Защита данных в Центре безопасности Azure)
- [Защита контейнеров в Центре безопасности](#)

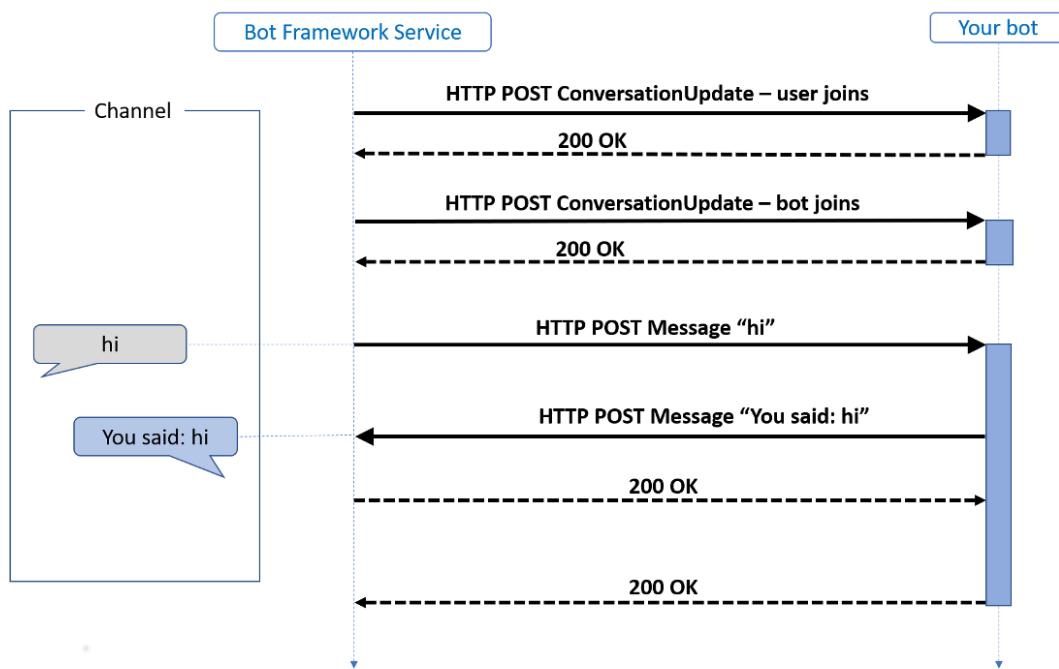
# Принципы работы бота

27.03.2021 • 23 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Бот — это приложение, с которым пользователи взаимодействуют, общаясь с помощью текста, графики (карт или изображений) или речи. Служба Azure Bot — это облачная платформа. Он размещает программы-роботы и делает их доступными для каналов.

Служба платформы Bot, которая является компонентом службы Azure Bot, отправляет данные между приложением, подключенным к Bot (например, Facebook или резервным) и т. д., которое мы вызываем канал, и Bot. Каждый канал может передавать в отправляемых событиях дополнительные сведения. Прежде чем создавать ботов, важно хорошо разобраться в использовании объектов действия для общения с пользователями бота. Для начала мы рассмотрим действия, которые передаются при выполнении простого бота проверки связи.



Здесь представлены действия двух типов: *обновление диалога* и *сообщение*.

Служба Bot Framework Service может отправлять события обновления беседы при добавлении к общению новых участников. Например, при начале беседы с эмулятором Bot Framework вы увидите два действия обновления диалога (один для подключаемого пользователя и второй для подключаемого бота). Чтобы отличать эти действия обновления диалога, проверьте, кто входит в свойство действия *добавленные члены*.

Действие сообщения передает между сторонами информацию общения. В нашем примере бота проверки связи события сообщений передают простой текст, который будет отображаться в выбранном канале. Кроме того, действие сообщения может содержать голосовые сообщения, предлагаемые действия или карты для отображения.

В нашем примере создается бот и отправляется действие сообщения в ответ на полученное входящее действие сообщения. Тем не менее, Bot может отвечать другими способами на действие полученного сообщения; программа-робот не часто реагирует на действие обновления диалога, отправляя некоторый текст приветствия в действие сообщения. Дополнительные сведения см. в подокне [приветствия](#)

пользователя.

## Пакет SDK для Bot Framework

Пакет SDK для Bot Framework позволяет создавать программы-роботы, которые могут размещаться в службе Azure Bot. Служба определяет REST API и протокол действий для взаимодействия между Bot и каналами или пользователями. Пакет SDK строится на основе этого REST API и предоставляет абстракцию службы, чтобы вы могли сосредоточиться на логике взаимодействия. Хотя вам не нужно понимать службу RESTFUL для использования пакета SDK, понимание некоторых его возможностей может быть полезным.

Программы-роботы — это приложения с интерфейсом для общения. Они помогут вам перенести на автоматизированные системы простые и повторяющиеся задачи, такие как резервирование столовиков в ресторане или сбор сведений для профиля без необходимости прямого участия человека. Пользователи взаимодействуют с ботом, используя текстовые сообщения, интерактивные карты и речь.

Взаимодействие с ботом может ограничиваться простыми вопросами и ответами или представлять собой сложное интеллектуальное общение с предоставлением доступа к службам.

### NOTE

Поддержка функций, предоставляемых пакетом SDK и REST API, зависит от канала. Вы можете протестировать Bot с помощью эмулятора Bot Framework, но необходимо протестировать все компоненты программы Bot на каждом канале, где вы планируете сделать Bot доступным.

Взаимодействие затрагивает обмен *действиями*, который обрабатывается *в свою* работу.

### Действия

Каждое взаимодействие между пользователем (или каналом) и Bot представлено в виде *действия*. [Схема действий](#) Bot Framework определяет действия, которыми можно обмениваться между пользователем или каналом и Bot. Действия могут представлять человеческий текст или речь, уведомления между приложениями, реакции на другие сообщения и т. д.

### Очередность

При общении люди обычно говорят по очереди. Как правило, бот реагирует на вводимые пользователем данные. В пакете SDK Bot Framework *ход* состоит из входящего действия пользователя и всех действий, которые бот немедленно возвращает пользователю в ответ. Вы можете подумать о том, как обработка, связанная с программой-роботом, получающим данное действие.

Например, пользователь может попросить робота выполнить определенную задачу. Bot может ответить на вопрос, чтобы получить дополнительные сведения о задаче, после чего эта задача заканчивается. При следующем включении Bot получает новое сообщение от пользователя, которое может содержать ответ на вопрос программы-робота, или может представлять изменение субъекта или запроса на игнорирование первоначального запроса на выполнение задачи.

## Структура приложения Bot

Пакет SDK определяет класс *Bot*, который обрабатывает диалоговые окна для приложения-робота.

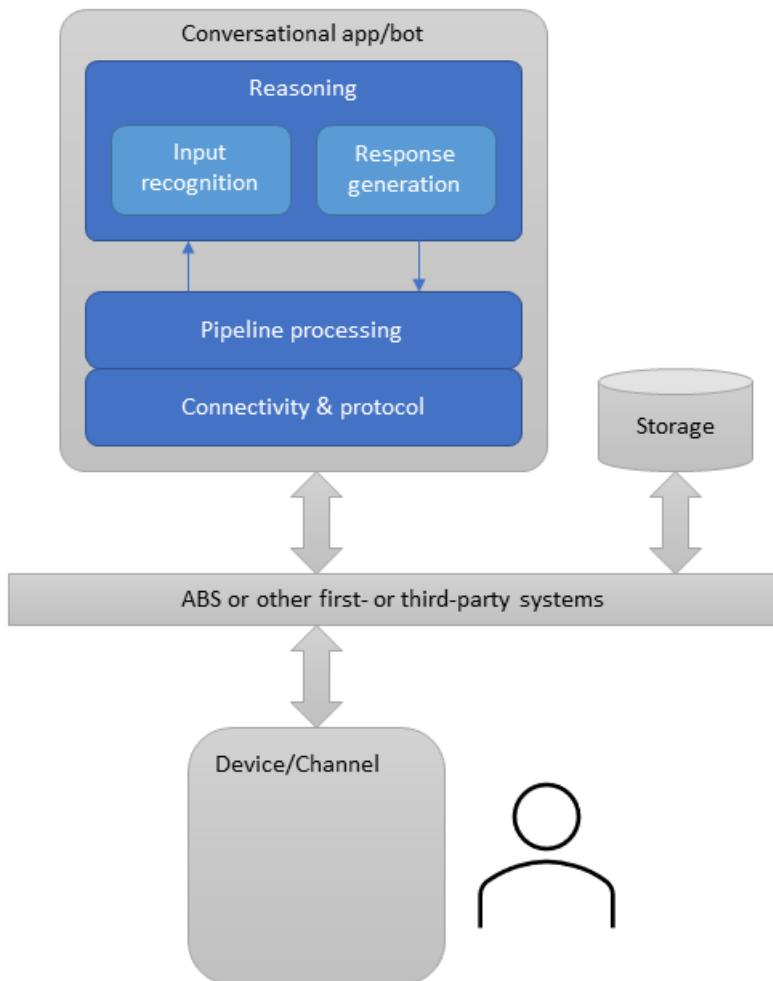
Класс *Bot*:

- Распознает и интерпретирует входные данные пользователя.
- Причины входных данных и выполнение соответствующих задач.
- Создает ответы о действиях, выполняемых программой-Bot.

Пакет SDK также определяет класс *адаптера*, который обрабатывает подключение к каналам. Адаптер:

- Предоставляет метод для обработки запросов из методов и для создания запросов к каналу пользователя.
- Включает конвейер по промежуточного слоя, который включает обработку за пределами обработчика поворачивания.
- Вызывает обработчик поворачивания Bot и перехватывает ошибки, не обработанные в обработчике событий иным образом.

Кроме того, программы-роботы часто приходится извлекать и хранить состояние каждый раз. Это осуществляется с помощью классов *хранилища*, *Bot-состояния* и *методов доступа к свойствам*. Пакет SDK не предоставляет встроенное хранилище, но предоставляет абстракции для хранения и несколько реализаций уровня хранилища. В разделе [Управление состоянием](#) описываются эти функции состояния и хранилища.



Пакет SDK не требует использования определенного уровня приложения для отправки и получения веб-запросов. В среде Bot есть шаблоны и примеры для ASP.NET (C#), restify (JavaScript) и aiohttp (Python). Однако для приложения можно использовать разные уровни приложений.

При создании программы-робота с помощью пакета SDK вы предоставляете код для получения HTTP-трафика и его пересылки на адаптер. Платформа Bot предоставляет несколько шаблонов и примеров, которые можно использовать для разработки собственных программы-роботы.

### Логика бота

Объект Bot содержит цепочку или логику для включения, а также предоставляет *обработчик включения*, который является методом, который может принимать входящие действия от адаптера Bot.

Пакет SDK предоставляет несколько различных парадигм для управления логикой Bot.

- *Обработчики действий* предоставляют модель, управляемую событиями, в которой типы входящих

действий и подтипы являются событиями. Это может быть хорошим для программы-роботы, которые имеют ограниченные и короткие взаимодействия с пользователем.

- Используйте [обработчик действий](#) и Реализуйте обработчики для каждого типа действия или подтипа, который будет распознать и реагировать в коде Bot.
- Используйте [обработчик активности команд](#) для создания программы-роботы, которые могут подключаться к каналу команд. (Каналу команд требуется, чтобы программа-робот обрабатывала определенное поведение канала.)
- [Библиотека диалогов](#) предоставляет модель на основе состояния для управления долго выполняющимся диалогом с пользователем.
  - Используйте обработчик действий и [диалоговое окно компонента](#) для практически последовательных диалогов. Дополнительные сведения см. в разделе [о диалоговых окнах Component и каскадного характера](#).
  - Используйте [диспетчер диалоговых окон](#) и [Адаптивное диалоговое окно](#) для гибкого потока диалоговых окон, которое может работать с более широким спектром действий пользователя. Дополнительные сведения см. в статье [Введение в адаптивные диалоговые окна](#).
- Реализуйте собственный класс Bot и предоставьте собственную логику для обработки каждой очереди. Узнайте, как [создать собственные приглашения для сбора входных данных пользователя](#), чтобы получить пример того, как это может выглядеть.

### Адаптер бота

Адаптер имеет метод *действия процесса* для начала работы.

- Он принимает текст запроса (полезные данные запроса, преобразуется в действие) и заголовок запроса в качестве аргументов.
- Он проверяет, является ли заголовок проверки подлинности допустимым.
- Он создает объект *контекста* для включения.
- Это выполняется с помощью конвейера по *промежуточного слоя*.
- Он отправляет действие обработчику переворачивания объекта Bot.

Адаптер также:

- Форматирует и отправляет действия ответа. Эти ответы обычно являются сообщениями для пользователя, но могут также содержать информацию, которая будет использоваться в канале пользователя напрямую.
- Предоставляет другие методы, предоставляемые соединителем Bot REST API, такие как *сообщение об обновлении* и *Удаление сообщения*.
- Перехватывает ошибки или исключения, которые в противном случае не были перехвачены для поочередности.

### Контекст переворачивания

Объект *контекста шага* предоставляет сведения о действии. К таким объектам относятся, например, идентификаторы канала, отправителя и получателя, а также другие необходимые данные для обработки действия. Кроме того, можно добавлять в объект контекста шага сведения при обработке шага на разных уровнях бота.

Контекст шага является одной из важнейших абстракций в пакете SDK. Он не только передает входящие действия всем компонентам по промежуточного слоя и логике приложения, но также предоставляет механизм, с помощью которого компоненты по промежуточного слоя и логика Bot могут отсылать исходящие действия.

### ПО промежуточного слоя

ПО промежуточного слоя действует так же, как любое средство обработки сообщений, и состоит из линейного набора компонентов, каждый из которых выполняется в строгом порядке и получает возможность применить некоторую логику реагирования на событие. Последним этапом конвейера ПО

промежуточного слоя является обратный вызов обработчика шагов из класса бота, регистрируемого приложением с использованием метода *действия обработки* адаптера. По промежуточному слоя реализует метод *включения*, который вызывает адаптер.

Обработчик «поверх» принимает контекст «поверх» в качестве аргумента, как правило, логика приложения, выполняемая внутри функции обработчика, выполняет обработку содержимого входящего действия и создает одно или несколько действий в ответе, отправляя их с помощью функции *Send действия* в контексте «выключить». Вызов *send activity* для контекста шага приводит к вызову компонентов ПО промежуточного слоя для исходящих действий. Компоненты по промежуточному слоя выполняются до и после функции обработчика поворачивания Bot. Выполнение по сути является вложенным и, как таковое, иногда называется калькой.

В разделе по [промежуточного слоя](#) подробно описывается по промежуточного слоя.

### Состояние и хранилище Bot

Как и в случае с другими веб-приложениями, Bot по своей природе не имеет состояния. Состояние в роботе соответствует тем же парадигмам, что и современные веб-приложения, а пакет SDK для Bot Framework предоставляет абстракции уровня хранилища и управления состоянием для упрощения управления состоянием.

В разделе [Управление состоянием](#) описываются эти функции состояния и хранилища.

### Конечная точка обмена сообщениями и подготовка

Как правило, приложению потребуется конечная точка RESTFUL, в которой будут приходить сообщения. Также потребуется подготавливать ресурсы для программы-робота в соответствии с используемой платформой.

Выполните одно из кратких руководств ([C#](#), [JavaScript](#), [Python](#)), чтобы создать и протестировать простой робот эха.

## Сведения об HTTP

Действия поступают в бот из службы Bot Framework через запросы HTTP POST. В ответ на входящий запрос POST бот возвращает код состояния HTTP 200. Действия, отправляемые из бота в канал, отправляются в отдельном запросе POST к службе Bot Framework. В ответ на него также поступает код состояния HTTP 200.

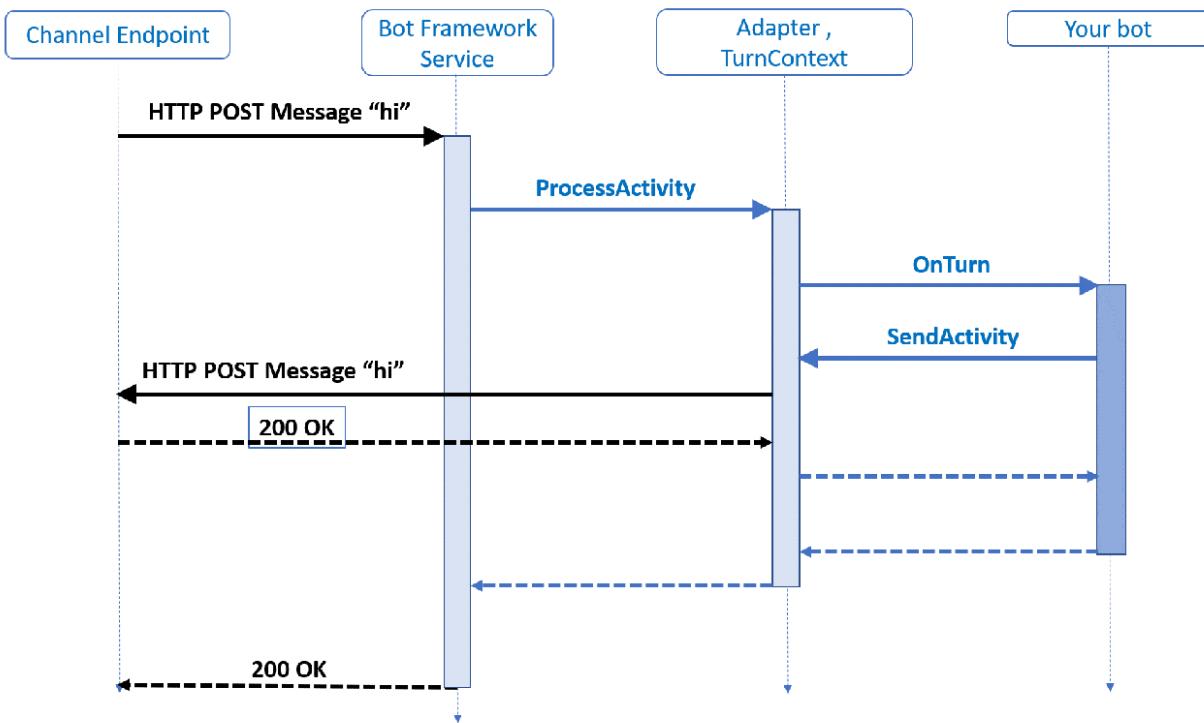
Протокол не указывает порядок, в котором выполняются эти запросы POST и их подтверждения. Но на распространенных платформах служб HTTP такие запросы обычно являются вложенными, то есть исходящий HTTP-запрос выполняется ботом в процессе обработки входящего HTTP-запроса. Этот шаблон показан на более ранней схеме. Поскольку здесь есть два раздельных HTTP-подключения с откликами на них, модель безопасности должна учитывать оба из них.

#### NOTE

У программы-робота есть 15 секунд для подтверждения вызова со статусом 200 на большинстве каналов. Если программа-робот не ответила в течение 15 секунд, возникает ошибка HTTP Гатевайтимеаут (504).

## Стек обработки действия

Давайте вернемся к предыдущей схеме последовательностей, чтобы сосредоточиться на поступлении действия сообщения.



В приведенном выше примере в ответ на действие сообщения бот отправляет другое действие сообщения, помещая в него текст из первого сообщения. Обработка начинается с поступления на веб-сервер запроса HTTP POST, в котором сведения о действии передаются в формате полезных данных JSON. В C# обычно это проект ASP.NET, в проекте Node.js JavaScript это может быть одна из популярных платформ, таких как Express или restify.

Встроенный компонент пакета SDK, который называется *адаптером*, является ядром среды выполнения SDK. Действие передается в тексте запроса HTTP POST в формате JSON. Этот JSON десериализуется для создания объекта *действия*, который затем передается адаптеру через его метод *действия Process*. При получении действия адаптер создает *turn context* (контекст шага) и вызывает ПО промежуточного слоя.

Как упоминалось выше, контекст шага позволяет ботам отправлять исходящие действия, чаще всего в ответ на входящее действие. Для этого контекст «переключить» предоставляет методы ответа на действия *отправки*, *обновления* и *удаления*. Каждый метод ответа выполняется в асинхронном процессе.

#### IMPORTANT

Поток, обрабатывающий первое включение бота, связан с удалением объекта контекста после окончания работы. **Обязательно ожидайте параметр `await` любых вызовов активности**, чтобы первичный поток дождался сгенерированного действия до завершения его работы и утилизации контекста включения. В противном случае, если ответ (или какой-либо из его обработчиков) после длительной паузы попытается воздействовать на объект контекста, может возникнуть ошибка *Контекст был удален*.

## Шаблоны Bot

Необходимо выбрать уровень приложения, используемый для приложения. Однако у платформы Bot есть шаблоны и примеры для ASP.NET (C#), restify (JavaScript) и aiohttp (Python). Документация написана, предполагая, что вы используете одну из этих платформ, но пакет SDK не требует этого. Инструкции по доступу к шаблонам и их установке см. в разделе краткие руководства ([C#](#), [JavaScript](#), [Python](#)).

Bot — это веб-приложение, а шаблоны предоставляются для каждой языковой версии пакета SDK. Все шаблоны предоставляют реализацию и адаптер конечной точки по умолчанию. Каждый шаблон

включает:

- Подготовка ресурсов
- Реализация конечной точки HTTP, зависящей от языка, которая направляет входящие действия на адаптер.
- Объект адаптера
- Объект Bot

Основное различие между различными типами шаблонов заключается в объекте Bot. Шаблоны:

- **Пустой робот**
  - Включает обработчик действий, который приветствует пользователя в диалоге, отправляя сообщение "Hello World" в первый ход диалога.
- **Бот Echo**
  - Использует обработчик действий для приветствия пользователей и вывода пользовательского ввода.
- **Основной робот**
  - Объединяет множество функций пакета SDK и демонстрирует рекомендации для программы-робота.
  - Использует обработчик действий для приветствия пользователей.
  - Использует диалоговое окно компонента и дочерние диалоговые окна для управления диалогом.
  - В диалогах используются функции Language Understanding (LUIS) и QnA Maker.

## Дополнительные сведения

### Управление ресурсами Bot

Ресурсами ботов, такими как идентификаторы приложений, пароли, ключи или секреты для подключенных служб, необходимо управлять соответствующим образом. Дополнительные сведения о том, как это сделать, см. в [руководствах по безопасности на Bot Framework](#) и об [управлении ресурсами Bot](#).

### Адаптеры каналов

Кроме того, пакет SDK позволяет использовать адаптеры каналов, в которых адаптер также выполняет задачи, которые служба соединителя Bot обычно использует для канала.

Пакет SDK предоставляет несколько адаптеров каналов на некоторых языках. Дополнительные адаптеры каналов доступны в репозиториях Botkit и сообщества. Дополнительные сведения см. в таблице "[каналы и адаптеры](#)" репозитория SDK для Bot Framework.

### Соединитель Bot REST API

Пакет SDK для Bot Framework упаковывает и строится на основе соединителя Bot REST API. Сведения об основных HTTP-запросах, поддерживающих пакет SDK, см. в разделе [Проверка подлинности](#) соединителя и связанные статьи. Действия, которые отправляет и получает Bot, соответствуют [схеме действия "Bot Framework"](#).

## Дальнейшие действия

- Общие сведения о роли состояния для ботов см. в статье [об управлении состоянием](#).
- Основные понятия разработки ботов для Microsoft Teams вам поможет понять статья [How Microsoft Teams bots work](#) (Принцип работы ботов Microsoft Teams).

# Управление состоянием

27.03.2021 • 16 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

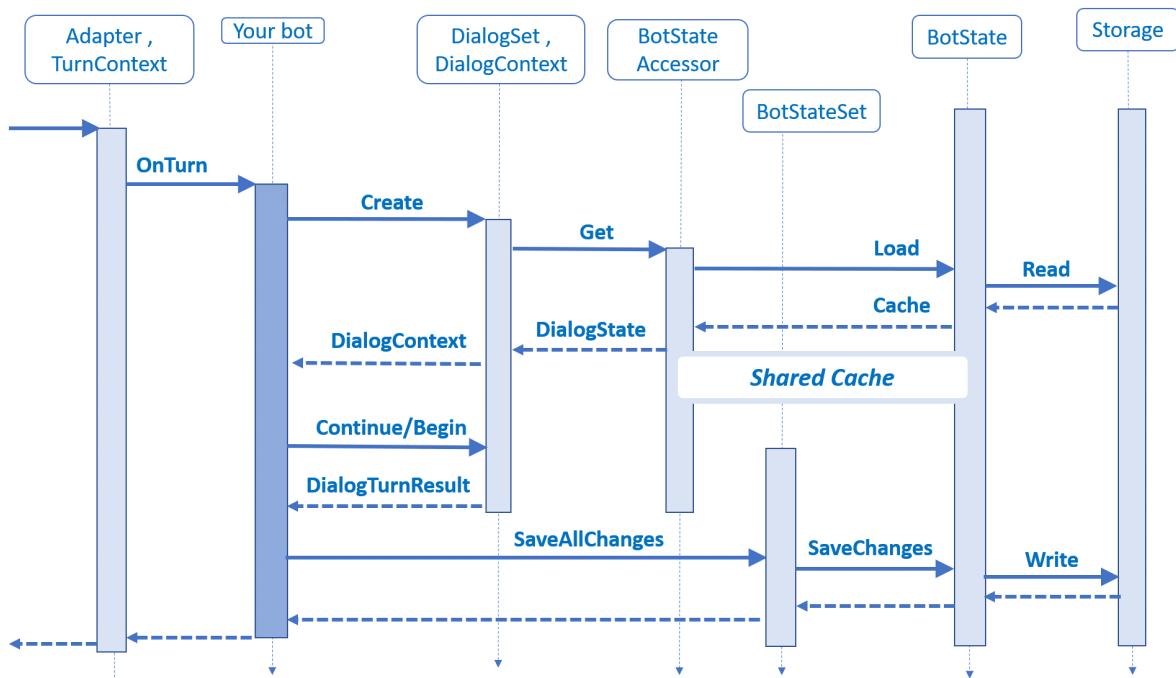
Состояние бота подчиняется тем же принципам, что и современные веб-приложения, и пакет SDK для Bot Framework предоставляет некоторые абстракции, позволяющие упростить управление состоянием.

Как и веб-приложения, бот по своей природе не учитывает состояния. Любой экземпляр бота может обработать любой шаг беседы. Для некоторых программы-роботы такая простота является предпочтительной, так — как Bot может работать без дополнительной информации, или необходимая информация гарантированно находится в входящем сообщении. Для других пользователей состояние (например, место, в котором отключен диалог или данные, полученные ранее) требуется для того, чтобы программа Bot получила полезную беседу.

## Зачем мне нужно состояние?

Поддержание состояния позволяет боту повысить информативность бесед путем запоминания некоторых данных о пользователе и (или) диалогах. Например, если вы уже общались ранее с некоторым пользователем, вы можете сохранить информацию о нем и не запрашивать ее повторно. Кроме того, состояние сохраняет данные не только о последнем шаге, что позволяет боту хранить сведения о ходе многоэтапного диалога.

Так как он относится к программы-роботы, существует несколько уровней использования состояния: уровень хранилища, управление состоянием (содержится в Bot-состоянии на схеме ниже) и методы доступа к свойствам состояния. На этой схеме показаны части последовательности взаимодействий этих уровней. Сплошные стрелки обозначают вызов метода, а пунктирными представлены ответы (с возвратом значения или без).



Поток на этой схеме описан в следующих разделах с подробными сведениями по каждому уровню.

## Уровень хранения

Начиная с серверной части, где фактически хранятся сведения о состоянии, это *уровень хранилища*. Это можно рассматривать как физическое хранилище, например в памяти, Azure или на сервере стороннего производителя.

Пакет SDK для Bot Framework содержит несколько реализации для уровня хранилища.

- **Хранилище в памяти** создает хранилище в памяти для целей тестирования. Хранение данных в памяти следует использовать только для локального тестирования, так как это хранилище является временным и ненадежным. Данные очищаются при каждом перезапуске бота.
- **Хранилище BLOB-объектов** подключается к базе данных хранилища больших двоичных объектов Azure.
- **Секционированное хранилище Azure Cosmos DB** подключается к секционированной базе данных Cosmos DB NoSQL.

#### IMPORTANT

Класс хранилища *Cosmos DB* является устаревшим. Контейнеры, изначально созданные с помощью Космосдбстораже, не имели набора ключей секций и получили ключ секции по умолчанию " / \_partitionKey".

Контейнеры, созданные с помощью хранилища *Cosmos DB*, можно использовать с *Cosmos DB секционированного хранилища*. См. сведения о [секционировании в Azure Cosmos DB](#).

Также обратите внимание, что, в отличие от устаревшего *Cosmos DB* хранения, *Cosmos DB* секционированного хранилища не создает базу данных в учетной записи *Cosmos DB* автоматически. Необходимо [создать новую базу данных вручную](#), но пропустить создание контейнера вручную, так как *космосдбартитионедстораже* создаст контейнер.

Инструкции по подключению к другим системам хранения можно найти в статье [Запись данных напрямую в хранилище](#).

## Управление данными о состоянии

Управление состоянием автоматизирует получение и сохранение состояния бота на уровне хранилища. Состояние хранится в виде *свойств состояния*, то есть пар "ключ — значение", которые бот может считывать и записывать через объект управления состоянием, не беспокоясь о конкретной реализации. Свойства состояния определяют, как хранятся эти данные. Например, извлекая свойство со значением определенного класса или объекта, вы заранее знаете структуру этих данных.

Эти свойства состояния размещаются в "контейнерах" с определенной областью видимости, которые представляют собой коллекции для упорядочивания этих свойств. Пакет SDK поддерживает три таких "контейнера":

- состояние пользователя;
- состояние беседы;
- личное состояние беседы.

Все эти контейнеры являются подклассами класса *bot state* (состояние бота), на основе которого вы можете определить и другие типы контейнеров с другими областями действия.

Эти предустановленные контейнеры имеют определенные области видимости.

- Состояние пользователя доступно на любом шаге, который бот выполняет в любой беседе с определенным пользователем на определенном канале.
- Состояние беседы доступно на любом шаге конкретной беседы для любого пользователя (например, в групповой беседе).
- Для личного состояния беседы область ограничивается как конкретным диалогом, так и конкретным

пользователем.

#### TIP

Область действий для состояний беседы и пользователя ограничивается определенным каналом. Один человек, взаимодействующий с ботом через несколько каналов связи, будет восприниматься как несколько разных пользователей с разными состояниями, по числу используемых каналов.

В каждом из стандартных контейнеров используются ключи, определенные для пользователя, для беседы или для обоих этих объектов. При настройке значения для свойства состояния определяется внутренний ключ с информацией в контексте шага, которая позволяет правильно соотносить пользователей и беседы с контейнерами и свойствами. Определяются следующие ключи:

- Состояние пользователя создает ключ на основе идентификатора канала (*Channel Id*) и идентификатора источника (*From Id*). Например:  
*{Activity.ChannelId}/users/{Activity.From.Id}#ИмяСвойства.*
- Состояние беседы создает ключ на основе идентификатора канала (*Channel Id*) и идентификатора общения (*Conversation Id*). Например:  
*{Activity.ChannelId}/conversations/{Activity.Conversation.Id}#ИмяСвойства.*
- Личное состояние беседы создает ключ на основе идентификатора канала (*Channel Id*), идентификатора источника (*From Id*) и идентификатора общения (*Conversation Id*). Например:  
*{Activity.ChannelId}/conversations/{Activity.Conversation.Id}/users/{Activity.From.Id}#ИмяСвойства*

#### Использование разных типов состояния

С помощью сведений о состоянии беседы можно отслеживать контекст беседы, например:

- задавал ли бот пользователю вопрос, и какой именно;
- на какую тему ведется текущая беседа или о чем была последняя завершенная.

С помощью сведений о состоянии пользователя можно отслеживать сведения о пользователе, например:

- некритические сведения о пользователе, например имя и персональные предпочтения, настройки будильников и (или) оповещений;
- сведения о последней беседе этого пользователя с ботом.
  - Например, бот службы поддержки может отслеживать список продуктов, о которых пользователь задавал вопросы.

Личное состояние беседы хорошо подходит для каналов, где поддерживаются групповые беседы, если в них есть смысл отслеживать определенные сведения о беседе и пользователе одновременно. Например, в боте для школьного класса можно выполнять следующие действия:

- собирать и отображать данные об ответах каждого учащегося на определенный вопрос;
- собирать сведения о результатах каждого учащегося и передавать их ему в частном порядке после каждого сеанса.

Дополнительные сведения о предопределенных контейнерах см. в [практическом руководстве по работе с состоянием](#).

#### Подключение к нескольким базам данных

Если бот должен подключаться к нескольким базам данных, создайте слой хранилища для каждой из них. Вы можете использовать несколько баз данных, если программа-робот собирает сведения, имеющие различные требования к безопасности, параллелизму или расположению данных.

Для каждого слоя хранилища создайте объекты управления состоянием, которые потребуются для хранения свойств состояния.

## Методы доступа к свойству состояния

Методы доступа к свойству состояния позволяют считывать и сохранять из текущего шага свойства состояния через предоставляемые методы *get*, *set* и *delete*. Чтобы создать метод доступа, укажите имя свойства. Обычно это происходит при инициализации бота. Позже вы сможете использовать этот метод доступа для получения и (или) изменения свойства в состоянии бота.

Методы доступа позволяют пакету SDK получать состояние из базового хранилища и самостоятельно обновляют *кэш состояний* бота. Кэш состояний представляет собой локальный кэш, который бот поддерживает для хранения объекта состояния и обработки чтения и записи без фактического доступа к базовому хранилищу. Если нужного состояния нет в кэше, вызванный метод доступа *get* получает его из хранилища и помещает в кэш. Полученное свойство состояния можно изменять так же, как и обычную локальную переменную.

Метод доступа *delete* удаляет свойство из кэша, а также из базового хранилища.

### IMPORTANT

При первом вызове метода доступа *get* ему необходимо предоставить фабричный метод, позволяющий создать объект, если он отсутствует в нужном состоянии. Если фабричный метод не предоставлен, создается исключение. Сведения об использовании фабричного метода можно найти в [практическом руководстве по работе с состоянием](#).

Чтобы сохранить изменения, внесенные в полученное свойство состояния, обновите свойство в кэше состояния. Для этого можно вызвать метод доступа *set*, который устанавливает значение свойства в кэше, т. е. делает новое значение доступным для чтения и (или) обновления далее на том же шаге. Чтобы сохранить эти данные в базовом хранилище (т. е. сделать их доступными на следующем шаге бота), необходимо [сохранить состояние](#).

### Как работают методы доступа к свойству состояния

Методы доступа являются основным средством для взаимодействия бота с состоянием. Ниже описаны принцип работы и взаимодействие базовых уровней для каждого из них.

- Метод доступа *get* выполняет следующее:
  - запрашивает данные из кэша состояний;
  - Если нужное свойство находится в кэше, оно сразу возвращается. В противном случае оно извлекается из объекта управления состоянием. (Если в состоянии еще нет этого объекта, применяется фабричный метод, предоставленный при вызове метода доступа *get*.)
- Метод доступа *set* выполняет следующее:
  - Сохраняет в кэше состояний новое значение свойства.
- Метод *save changes* в объекте управления состоянием выполняет следующее:
  - Проверяет изменения свойства в кэше состояний.
  - Сохраняет нужное свойство в хранилище.

## Состояние в диалоговых окнах

Библиотека диалоговых окон использует метод доступа к свойству состояния диалогового окна, определенный в состоянии диалога Bot, для сохранения диалогового окна диалога. Свойство состояния диалогового окна также позволяет каждому диалоговому окну сохранять временные данные между переходами.

Адаптивные диалоговые окна имеют более сложную структуру области памяти, что упрощает доступ к результатам настройки и распознавания, а также к другим элементам. *Диспетчер диалоговых окон* использует объекты управления пользователя и состояния диалога для предоставления этих областей

памяти.

Дополнительные сведения о библиотеке диалогов см. в статье [Библиотека диалоговых окон](#).

- Сведения о типах диалоговых окон см. в разделе [о диалоговых окнах компонентов и каскадных типов](#).
- Сведения, относящиеся к адаптивным диалоговым окнам, см. в статьях [Введение в адаптивные диалоговые окна](#) и [Управление состоянием в адаптивных диалоговых окнах](#).

## Сохранение состояния

Если вы вызываете метод доступа `set` для сохранения обновленного состояния, свойство состояния не сохраняется в постоянное хранилище, а обновляется только в кэше состояний вашего бота. Чтобы сохранить в постоянное хранилище все внесенные в кэше изменения, следует вызвать метод `save changes` в объекте управления состоянием, который доступен через вышеупомянутую реализацию класса состояния бота (например, состояния пользователя или состояния беседы).

Вызов метода сохранения изменений в объекте управления состоянием (например, для вышеупомянутых контейнеров) сохраняет в выбранный контейнер все свойства, обновленные на текущий момент в кэше состояний, но не затрагивает другие контейнеры в состоянии бота.

### TIP

Состояние бота реализует подход "приоритет имеет последняя запись", т. е. последняя операция записи отменяет все ранее сохраненные состояния. Это допустимо для многих приложений, но имеет свои побочные эффекты, особенно в сценариях горизонтального масштабирования с определенным уровнем параллелизма и (или) задержек.

Если у вас есть пользовательское ПО промежуточного слоя, которое может обновлять состояние после завершения обработчика шага, есть смысл [управлять состоянием в этом ПО промежуточного слоя](#).

## Дополнительные ресурсы

- [Непосредственная запись в хранилище](#)
- [Сохранение данных пользователя и диалога](#)

# Управляемые событиями диалоги с помощью обработчика действий

27.03.2021 • 20 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

*Обработчик действий* — это управляемый событиями способ организации логики диалога для робота. Каждый другой тип или подтип действия представляет собой событие другого типа. На самом деле *обработчик пошагового выполнения* программы-робота вызывает отдельный обработчик действия для любого типа полученного действия.

Например, если программа Bot получает действие сообщения, обработчик «поверх» будет видеть это входящее действие и отправить его обработчику действия *On Message*. При создании программы-робота для обработки сообщений и реагирования на них в обработчике *действия сообщения* будет отправлена логика Bot. Аналогичным образом, логика для обработки членов, добавляемых в диалог, будет находиться в обработчике *добавленных членов*, который вызывается при каждом добавлении элемента в диалог.

Другие способы организации логики Bot см. в разделе " [логика Bot](#) " в статье **принцип работы программы-роботы**.

- [C#](#)
- [JavaScript](#)
- [Python](#)

Чтобы реализовать логику для этих обработчиков, необходимо переопределить эти методы в роботе, как показано в разделе [пример обработчика действий](#) ниже. Базовой реализации для каждого из этих обработчиков нет, поэтому просто добавляйте логику, используемую при переопределении.

В некоторых ситуациях требуется переопределить основной обработчик шагов, например, для [сохранения состояния](#) в конце шага. Перед этим обязательно вызовите

```
await base.OnTurnAsync(turnContext, cancellationToken);
```

, чтобы убедиться, что базовая реализация `OnTurnAsync` выполняется перед дополнительным кодом. Эта базовая реализация, помимо прочего, отвечает за вызов остальных обработчиков действий, таких как `OnMessageActivityAsync`.

## Обработка действий

Логика бота обрабатывает входящие действия из одного или нескольких каналов и создает исходящие действия в ответ.

- [C#](#)
- [JavaScript](#)
- [Python](#)

Основная логика Bot определена в коде Bot. Чтобы реализовать робот в качестве обработчика действия, создайте производный класс Bot из `ActivityHandler`, который реализует `IBot` интерфейс.

`ActivityHandler` Определяет различные обработчики для различных типов действий, таких как `OnMessageActivityAsync`, И `OnMembersAddedAsync`. Эти методы защищены, но могут быть переопределены, так как мы производим от `ActivityHandler`.

Ниже описаны обработчики, определенные в `ActivityHandler`:

СОБЫТИЕ	ОБРАБОТЧИК	ОПИСАНИЕ
Любой тип полученного действия	OnTurnAsync	Вызывает один из других обработчиков на основе типа полученного действия.
Полученное действие сообщения	OnMessageActivityAsync	Переопределите для обработки действия <code>message</code> .
Полученное действие обновления диалога	OnConversationUpdateActivityAsync	В действии <code>conversationUpdate</code> вызывает обработчик, если участники не добавлены в бота или покидают беседу.
Не добавленные в бота участники, которые присоединяются к беседе	OnMembersAddedAsync	Переопределите для обработки участников, присоединяющихся к беседе.
Не добавленные в бота участники, которые покидают беседу	OnMembersRemovedAsync	Переопределите для обработки участников, покидающих беседу.
Полученное действие события	OnEventActivityAsync	В действии <code>event</code> вызывает обработчик для события определенного типа.
Полученное действие события ответа маркера	OnTokenResponseEventAsync	Переопределите для обработки событий ответа маркера.
Полученное действие события другого ответа	OnEventAsync	Переопределите для обработки событий других типов.
Получение действия реакции на сообщение	OnMessageReactionActivityAsync	При получении действия <code>messageReaction</code> вызывает обработчик, если в сообщении были добавлены или удалены одна или несколько реакций.
Добавление реакции на сообщение в сообщение	OnReactionsAddedAsync	Переопределите, чтобы обрабатывать добавление реакций в сообщение.
Удаление реакции на сообщение из сообщения	OnReactionsRemovedAsync	Переопределите, чтобы обрабатывать удаление реакций из сообщения.
Получено действие обновления установки	OnInstallationUpdateActivityAsync	В <code>installationUpdate</code> действии вызывает обработчик на основе того, был ли установлен или удален объект Bot.
Bot установлен	OnInstallationUpdateAddAsync	Переопределите этот параметр, чтобы добавить логику для установки программы-робота в подразделении.

СОБЫТИЕ	ОБРАБОТЧИК	ОПИСАНИЕ
Удален Bot	OnInstallationUpdateRemoveAsync	Переопределите этот параметр, чтобы добавить логику для удаления Bot в подразделении.
Другой тип полученного действия	OnUnrecognizedActivityTypeAsync	Переопределите для обработки действия любого типа, которое иначе не будет обработано.

Эти разные обработчики используют `turnContext` для получения сведений о входящем действии, которое соответствует входящему HTTP-запросу. Действия могут быть разных типов, поэтому каждый обработчик предоставляет действием со строгой типизацией в параметре контекста шага. В большинстве случаев `OnMessageActivityAsync` является наиболее распространенным и всегда обрабатывается.

Как и в предыдущих версиях 4.x этой платформы, в этой версии также можно реализовать открытый метод `OnTurnAsync`. Сейчас базовая реализация этого метода выполняет проверку ошибок, а затем вызывает каждый из определенных обработчиков (например те два, которые мы определяем в этом примере) в зависимости от типа входящего действия. В большинстве случаев можно не изменять этот метод и использовать отдельные обработчики. Но при необходимости вы также можете работать с пользовательской реализацией `OnTurnAsync`.

#### IMPORTANT

При переопределении метода `OnTurnAsync` вам нужно вызвать `base.OnTurnAsync`, чтобы получить базовую реализацию для вызова всех остальных обработчиков `On<activity>Async`. Но вы также можете вызывать эти обработчики самостоятельно. В противном случае эти обработчики не будут вызваны и код не будет выполняться.

## Пример обработчика действия

Например, вы можете работать с членами, добавленными для приветствия в беседу, и обменять сообщения для вывода сообщений, отправляемых им в Bot.

- [C#](#)
- [JavaScript](#)
- [Python](#)

```
public class EchoBot : ActivityHandler
{
    protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
    {
        var replyText = $"Echo: {turnContext.Activity.Text}";
        await turnContext.SendActivityAsync(MessageFactory.Text(replyText, replyText), cancellationToken);
    }

    protected override async Task OnMembersAddedAsync(IList<ChannelAccount> membersAdded,
ITurnContext<IConversationUpdateActivity> turnContext, CancellationToken cancellationToken)
    {
        var welcomeText = "Hello and welcome!";
        foreach (var member in membersAdded)
        {
            if (member.Id != turnContext.Activity.Recipient.Id)
            {
                await turnContext.SendActivityAsync(MessageFactory.Text(welcomeText, welcomeText),
cancellationToken);
            }
        }
    }
}
```

## Дальнейшие действия

- Канал Microsoft Teams представляет некоторые действия, связанные с командами, которые должны поддерживаться программой Bot для правильной работы с командами. Основные понятия разработки ботов для Microsoft Teams вам поможет понять статья [How Microsoft Teams bots work](#) (Принцип работы ботов Microsoft Teams).
- Обработчик действий — это хороший способ разработки робота, который не должен отслеживать интерактивное состояние между операциями включения. [Библиотека диалогов](#) предоставляет способы управления долго выполняющимся диалогом с пользователем.

# Библиотека диалогов

27.10.2020 • 15 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Диалоговые окна — это центральная концепция в пакете SDK, предоставляющая способы управления долго выполняющимся диалогом с пользователем. Диалоговое окно выполняет задачу, которая может представлять часть или полный поток диалога. Он может охватывать только один раз или несколько, и может охватывать короткий или длительный период времени.

В этой статье описываются основные классы и функции библиотеки диалогов. Вы должны быть знакомы с тем, [как работает программы-роботы](#) (включая что-то [делать](#)) и [управлять состоянием](#).

Каждое диалоговое окно представляет собой задачу, которая может выполняться до завершения и возвращать собранные сведения. Каждое диалоговое окно представляет собой базовую единицу потока управления: оно может начинаться, продолжаться и заканчиваться. Приостановка и возобновление; или отменить.

Диалоговые окна похожи на метод или функцию в языке программирования. Можно передать аргументы или параметры при запуске диалогового окна, после чего диалоговое окно может получить возвращаемое значение по завершении.

## Состояние в Dialog

Диалоговые окна — это способ реализации *множественной беседы*, и поэтому они полагаются на *постоянное состояние* во всех случаях. Без состояния в диалоговых окнах робот не сможет понять, где находился в беседе или какая информация уже собрана.

Чтобы сохранить диалоговое окно в диалоге, состояние диалогового окна должно быть извлечено из и сохранено в память каждый ход. Это осуществляется через метод доступа свойства состояния диалогового окна, определенного в состоянии диалога Bot. Это диалоговое окно собирает сведения для всех активных диалоговых окон и потомков активных диалоговых окон. Это позволяет элементу Bot выбрать место, где она была оставлена последней, и управлять множеством моделей диалога.

Во время выполнения свойство состояния диалогового окна содержит сведения о том, где диалоговое окно находится в его логическом процессе, включая все внутренние собранные сведения в форме объекта *экземпляра диалогового окна*. Опять же, он должен быть считан в Bot и сохранен в памяти каждый из них.

## Инфраструктура диалоговых окон

Наряду с различными типами диалоговых окон в разработке и управлении диалогами участвуют следующие классы. Хотя обычно не требуется взаимодействовать с этими классами напрямую, они знают о них и их назначение полезно при проектировании диалоговых окон для программы-робота.

КЛАСС	ОПИСАНИЕ
<i>Набор диалоговых окон</i>	Определяет коллекцию диалоговых окон, которые могут ссылаться друг на друга и работать совместно.
<i>Контекст диалога</i>	Содержит сведения обо всех активных диалоговых окнах.

**КЛАСС****ОПИСАНИЕ**

<i>Экземпляр диалогового окна</i>	Содержит сведения об одном активном диалоговом окне.
<i>Результат диалогового окна</i>	Содержит сведения о состоянии из активного или недавно активного диалогового окна. Если активное диалоговое окно завершилось, оно содержит возвращаемое значение.

**NOTE**

В рамках появления адаптивных диалоговых окон версия 4,9 пакета SDK для C# предоставила класс *диспетчера диалоговых окон*, который автоматизирует многие задачи управления диалоговыми окнами. (Адаптивные диалоговые окна доступны в предварительной версии пакета SDK для JavaScript версия 4,10.) Дополнительные сведения см. в статье [Введение в адаптивные диалоговые окна](#).

## Типы диалогов

Библиотека диалоговых окон предоставляет несколько типов диалоговых окон, облегчающих управление беседами в Bot.

ТИП	ОПИСАНИЕ
<i>Откроется</i>	Базовый класс для всех диалоговых окон.
<i>диалоговое окно контейнера</i>	Базовый класс для всех диалоговых окон <i>контейнера</i> , таких как компонент и адаптивные диалоговые окна. Он поддерживает внутренний набор диалоговых окон и позволяет рассматривать коллекцию диалоговых окон как единое целое.
<i>диалоговое окно компонента</i>	Тип общего назначения контейнера, который инкапсулирует набор диалоговых окон, позволяющий повторно использовать набор в целом. Когда запускается диалоговое окно компонента, оно начинается с назначенного диалогового окна в его коллекции. По завершении внутреннего процесса диалоговое окно компонента завершается.
<i>каскадное диалоговое окно</i>	Определяет последовательность шагов, позволяя программе-роботу перебрать пользователя с помощью линейного процесса. Обычно они предназначены для работы в контексте диалогового окна компонента.
<i>диалоговые окна командной строки</i>	Запросите у пользователя ввод и возвратите результат. Запрос будет повторяться до тех пор, пока не получит допустимый ввод или не будет отменен. Они предназначены для работы с каскадными диалогами.

ТИП	ОПИСАНИЕ
<i>Адаптивное диалоговое окно</i>	Тип диалогового окна контейнера, позволяющего гибкому потоку диалога. Он включает встроенную поддержку для распознавания языка, создания языка и функций области памяти. Чтобы выполнить адаптивный диалог (или другой диалог, содержащий адаптивный диалог), его нужно запустить из диспетчера диалогов.
<i>диалоговые окна действий</i>	Представление программных структур. Они позволяют проектировать поток бесед практически так же, как выражения и инструкции в традиционном языке программирования, позволяющие проектировать процедурные потоки в приложении. Они работают только в адаптивном диалоговом окне.
<i>диалоговые окна ввода</i>	Запросите у пользователя ввод, похожий на диалоговые окна запроса, но с другими структурами в адаптивном диалоговом окне. Они работают только в адаптивном диалоговом окне.
<i>диалоговое окно навыка</i>	Автоматизирует управление одним или несколькими навыками, программы-роботы от потребителя навыков.
<i>Диалоговое окно QnA Maker</i>	Автоматизирует доступ к базе знаний QnA Maker.

## Шаблоны диалоговых окон

Существует два основных шаблона запуска диалоговых окон и управления ими из программы-робота.

1. Для адаптивных диалоговых окон или любого набора диалоговых окон, содержащих Адаптивное диалоговое окно, необходимо создать экземпляр диспетчера диалоговых окон для корневого диалогового окна и сделать состояние диалога (и, при необходимости, пользовательское состояние) доступным для руководителя. Дополнительные сведения см. в статьях [Введение в адаптивные диалоговые окна](#) и [Создание программы-робота с помощью адаптивных диалоговых окон](#).
2. Для других диалоговых окон можно использовать диспетчер диалоговых окон или просто использовать метод расширения *Run* для корневого диалогового окна. Сведения об использовании метода Run с диалоговым окном компонента см. в разделах [о диалоговых окнах Component](#) и [каскадного взаимодействия](#) и как [реализовать последовательный поток диалога](#).

### Стек диалоговых окон

Контекст диалогового окна содержит сведения обо всех активных диалоговых окнах и включает *стек диалоговых окон*, который выступает в качестве *стека вызовов* для всех активных диалоговых окон. Каждое диалоговое окно контейнера имеет внутренний набор диалоговых окон, управление которыми осуществляется, и поэтому в каждом активном диалоговом окне контейнера отображается внутренний контекст диалогового окна и стек диалоговых окон в рамках своего состояния.

Хотя вы не будете обращаться к стеку напрямую, понимание того, что он существует и его функция поможет понять, как работают различные аспекты библиотеки диалогов.

## Диалоговые окна контейнера

Диалоговое окно контейнера может быть частью большего набора диалоговых окон. Каждый контейнер имеет внутренний набор диалоговых окон, который также управляет.

- Каждый набор диалоговых окон создает область для разрешения идентификаторов диалоговых окон.
- В настоящее время пакет SDK реализует два типа диалоговых окна контейнеров: диалоговые окна компонентов и адаптивные диалоговые окна. В то время как концептуальная структура двух весьма различается, их можно использовать вместе.

## Идентификаторы диалоговых окон

При добавлении диалогового окна в набор диалоговых окон ему назначается уникальный идентификатор в пределах этого набора. Диалоговые окна в наборе ссылаются друг на друга по идентификаторам.

Когда одно диалоговое окно ссылается на другое диалоговое окно во время выполнения, оно делает это в ИДЕНТИФИКАТОРе диалогового окна. Контекст диалогового окна пытается разрешить идентификатор на основе других диалоговых окон в диалоговом окне немедленного набора. Если совпадений нет, он ищет совпадение в содержащем или внешнем наборе диалоговых окон и т. д. Если совпадений не найдено, создается исключение или ошибка.

## Компонентные диалоги

В диалоговых окнах компонента используется модель последовательностей для диалогов, и каждое диалоговое окно в контейнере отвечает за вызов других диалоговых окон в контейнере. При пустом стеке внутреннего диалогового окна компонента Компонент завершается.

Рассмотрите возможность использования диалоговых окон компонентов и каскадов, если у программы-робота есть сравнительно простой поток управления, для которого не требуется более динамичный поток диалога.

[О диалоговых окнах компонентов и каскадов](#). подробное описание диалоговых окон компонентов, каскадов и запросов.

## Адаптивные диалоги

Адаптивные диалоговые окна используют гибкую модель для диалогов для более широкого спектра действий пользователя. Адаптивное диалоговое окно можно разрабатывать, чтобы завершить или оставить активным, если внутренний стек диалоговых окон пуст. Они предлагают несколько встроенных возможностей, включая обработку прерываний, распознавание языка, создание языка и многое другое. Адаптивные диалоги позволяют уделять больше внимания разработке беседы, не акцентируя внимание на механизме диалога.

Адаптивный диалог входит в библиотеку диалогов и работает с диалогами любых других типов. Вы можете легко создать бот, который использует несколько типов диалогов.

Адаптивные диалоги можно использовать в следующих сценариях:

- Бот требует ветвлений или циклов в потоке беседы на основе бизнес-логики, вводимых пользователем данных или других факторов.
- Бот должен изменять контекст, принимать данные в произвольном порядке или допускать приостановку одного потока беседы для создания побочной беседы с последующим возвратом к главному потоку.
- Бот требует моделей распознавания речи с учетом контекста или распознавания сущностей в пользовательских данных.
- Бот может пользоваться преимуществами пользовательской обработки входных данных или создания ответа.

В статье [Общие сведения о адаптивных диалоговых окнах](#) и других адаптивных диалоговых окнах описаны функции, поддерживаемые адаптивными диалогами: распознавание языка и поддержка формирования языка, использование триггеров и действий для моделирования потока обсуждений, доступ к областям памяти и т. д. Дополнительные сведения об использовании диспетчера диалоговых

окон для запуска адаптивного диалогового окна см. в статье [Создание программы-робота с помощью адаптивных диалоговых окон](#).

## Другие диалоговые окна

Диалоги QnA Maker и навыка можно использовать в качестве изолированных диалоговых окон или в составе коллекции диалоговых окон в контейнере.

### Диалоговое окно QnA Maker

Диалоговое окно QnA Maker обращается к базе знаний QnA Maker и поддерживает QnA Maker запросы к исполнению и активные средства обучения.

- Запросы к исполнению, также известные как многострочные запросы, позволяют базе знаний запрашивать у пользователя дополнительные сведения, прежде чем ответить на свой вопрос.
- Активные предложения по обучению позволяют усовершенствовать базу знаний со временем. Диалог QnA Maker поддерживает явные отзывы для функции активного обучения.

Сведения о QnA Maker см. в разделе [использование QnA Maker для ответа на вопросы](#).

### Диалоговое окно навыка

Диалоговое окно навыка обращается к одному или нескольким навыкам и управляет ими. Диалог навыка отправляет действия из родительского робота в бот навыка и возвращает пользователю ответы от этого навыка.

Сведения о навыках см. в [обзоре навыков](#).

## Дальнейшие действия

[О по промежуточного слоя](#)

# О диалоговых окнах Component и каскадных компонентов

27.03.2021 • 22 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Диалоговые окна бывают разных типов. В этой статье описываются диалоговые окна компонентов, каскадов и запросов. Дополнительные сведения о диалоговых окнах см. в статье [Библиотека диалогов](#). Сведения о адаптивных диалоговых окнах см. в статье [Введение в адаптивные диалоговые окна](#).

*Каскадное диалоговое окно* (или каскадное) определяет последовательность шагов, позволяя программе-роботу перебрать пользователя с помощью линейного процесса. Обычно они предназначены для работы в контексте [диалогового окна компонента](#).

Диалоговое окно компонента — это тип диалогового окна контейнера, который позволяет диалоговым окнам в наборе вызывать другие диалоги в наборе, такие как диалоговые окна диалогового окна каскадного вызова или другое диалоговое окно каскада. Диалоговые окна компонента управляют набором *дочерних* диалоговых окон, таких как каскадные диалоговые окна, приглашения и т. д. Можно разработать диалоговое окно компонента для выполнения конкретных задач и его повторного использования в той же программе-роботе или в нескольких программах-роботах.

*Диалоговые окна командной строки* (приглашения) — это диалоговые окна, предназначенные для того, чтобы запрашивать у пользователя определенные типы информации, такие как число, Дата, имя и т. д. Запросы предназначены для работы с каскадными диалогами в диалоговом окне компонента.

## Компонентные диалоги

Иногда вы будете создавать диалог для многократного использования, который применим для разных сценариев. В качестве примера можно представить диалог, который предлагает пользователю указать название улицы, город и почтовый индекс.

*Компонентный диалог* позволяет создавать независимые диалоги для обработки определенных сценариев, разбивая большие наборы диалогов на более управляемые компоненты. Каждый из этих компонентов имеет отдельный набор диалогов, что позволяет избежать конфликтов имен с родительским набором диалогов. Подробнее об этом см. в [руководстве по компонентным диалогам](#).

## Каскадные диалоги

Каскадным диалогом называют особую реализацию диалога, которая часто используется для сбора информации от пользователя или предоставления пользователю инструкций по выполнению ряда задач. Каждый шаг диалога реализуется как асинхронная функция, которая принимает параметр *контекста каскадного шага* (`step`). На каждом шаге бот *запрашивает у пользователя входные данные* (или запускает дочерний диалог — обычно это еще один запрос), ожидает ответа и передает результат в следующий шаг. Результат выполнения первой функции передается в виде аргумента следующую функцию и т. д.

На следующей диаграмме показана последовательность шагов каскада и выполняемых операций стека. Сведения об использовании стека диалогов см. ниже в разделе об [использовании диалогов](#).

## DialogContext Begin Waterfall

## Push Waterfall

### Waterfall Step 1

- Begin Prompt 1

Push Prompt 1

### Waterfall Step 2

- Process result from Prompt 1
- Begin Prompt 2

Pop Prompt 1

Push Prompt 2

### Waterfall Step 3

- Process result from Prompt 2
- Begin Prompt 3

Pop Prompt 2

Push Prompt 3

### Waterfall Step 4

- Process result from Prompt 3
- EndDialog (stack entry goes away)

Pop Prompt 3

Pop Waterfall

При выполнении шагов каскадного диалога сохраняется соответствующий *контекст*. Он применяется так же, как контекст диалога, то есть предоставляет сведения о контексте и состоянии текущего шага.

Контекстный объект шага каскада используется для взаимодействия с набором диалогов из шага каскада.

Значение, возвращаемое диалогом, можно обрабатывать в другом каскадном шаге этого же диалога или в основном обработчике шагов бота. Обычно достаточно только проверить состояние результата шага диалога, полученное от логики обработчика шагов. Внутри шага каскада диалог возвращает значение в контексте шага каскада в виде свойства *result*.

### Свойства контекста для каскадного шага

Контекст каскадного шага содержит следующие элементы:

- *Параметры* — входные данные для диалога.
- *Значения* — сведения, которые можно добавить в контекст и передать в последующие шаги.
- *Результат* — результат, полученный от предыдущего шага.

Кроме того, следующий метод (**некстасинк** в C#, **Next** в JS и Python) переходит к следующему шагу диалогового окна каскада в той же самой очереди, позволяя роботу пропустить определенный шаг при необходимости.

## Запросы

Запросы из библиотеки диалогов — это простой способ запросить у пользователя определенные сведения и оценить ответы. Например, для запроса числа вы определяете вопрос или подсказку о нужных сведениях, и запрос автоматически проверяет, является ли полученный ответ (число) допустимым. Если это так, диалог можно продолжить. В противном случае пользователю повторно предлагается ввести допустимый ответ.

Запросы данных, по сути, являются диалогами из двух этапов. Сначала запрос предлагает ввести данные, а затем возвращает допустимое значение или повторяет цикл запроса.

Запросы имеют *параметры запроса*, которые предоставляются при вызове запроса. Они позволяют

указать текст для строки приглашения, строку повторного запроса при неудачной проверке и варианты ответа на запрос. Как правило, аргументы запроса и повторной попытки являются действиями, но в разных языках программирования они обрабатываются немного по-разному.

Кроме того, вы можете добавить в запрос пользовательские проверки при его создании. Предположим, что мы хотим узнать размер компании с помощью запроса числа, но этот размер будет ограничен значениями от 2 до 12. Такой запрос сначала проверяет, получил ли он допустимое число, а затем выполняет пользовательскую проверку, если она настроена. Если пользовательская проверка не будет пройдена успешно, запрос выведет повторное обращение к пользователю, как описано выше.

Завершенный запрос явным образом возвращает значение, которое было запрошено. Получив это значение, мы можем быть уверены, что оно удовлетворяет как встроенным параметрам проверки, так и любой настроенной пользовательской проверке.

Примеры разных запросов и их использования см. в руководстве по [сбору вводимых пользователем данных с помощью библиотеки диалогов](#).

### Типы запросов

Запросы данных, по сути, являются диалогами из двух этапов. На первом запрашиваются входные данные, а на втором возвращается допустимое значение или цикл запроса запускается заново. Библиотека диалогов содержит несколько простых запросов, каждый из которых возвращает разные типы ответов. Базовые запросы могут интерпретировать входные данные на естественном языке, например числа "ten" (десять) или "a dozen" (дюжина) и указания времени "tomorrow" (завтра) или "Friday at 10am" (в 10 вечера в пятницу).

PROMPT	ОПИСАНИЕ	РЕЗУЛЬТАТЫ
<i>Запрос вложений</i>	Предложение передать одно или несколько вложений, например документов или изображений.	Коллекция объектов <i>вложений</i> .
<i>Запрос выбора</i>	Предложение выбрать один параметр из набора.	Объект <i>найденного выбора</i> .
<i>Запрос подтверждения</i>	Запрашивается подтверждение.	Значение типа Boolean.
<i>Запрос даты и времени</i>	Предложение ввести дату и (или) время.	Коллекция объектов <i>разрешения даты и времени</i> .
<i>Запрос числа</i>	Предложение ввести число.	Числовое значение.
<i>Запрос текста</i>	Предложение ввести входные данные в простом текстовом виде.	Строка.

Чтобы запросить ввод данных от пользователя, определите запрос на основе одного из встроенных классов, например [текстового запроса](#), и добавьте этот запрос в набор диалогов. Запросы имеют фиксированное идентификаторы, уникальные в пределах набора диалогов. Для каждого запроса вы можете создать пользовательский проверяющий элемент управления, а для некоторых — еще и указать [языковой стандарт по умолчанию](#).

### Языковой стандарт для запроса

Языковой стандарт используется для определения поведения, зависящего от языка, в запросах **выбора, подтверждения, даты и времени и числа**. Для каждого полученного от пользователя сообщения действует следующее правило: если канал передает в сообщении [языковой стандарт](#), то используется именно он. Если для запроса задан [языковой стандарт по умолчанию](#) при вызове конструктора строки

или позднее, то используется именно он. Если языковый стандарт не указан, в качестве языкового стандарта используется английский язык (en-us). Примечание. Языковой стандарт определяется кодом ISO 639 из 2, 3 или 4 символов, который указывает определенный язык или языковую группу.

## Параметры запроса

Второй параметр для метода *prompt* контекста шага принимает объект *аргументов запроса* с указанными ниже свойствами.

Свойство	Описание
<i>Prompt</i>	Исходное действие, отправляемое пользователю для получения входных данных.
<i>Retry Prompt</i>	Действие, отправляемое пользователю при несоблюдении формата для первого введенного значения.
<i>Choices</i>	Список вариантов, из которых может выбрать пользователь при использовании запроса выбора.
<i>Validations</i>	Дополнительные параметры для пользовательского проверяющего элемента управления.
<i>Style</i>	Определяет, как варианты запроса выбора или подтверждения будут отображаться пользователю.

Всегда следует указывать действие исходного запроса для отправки пользователю, а также запрос повторной попытки для случаев, когда введенные пользователем данные не проверяются.

Если введенные пользователем данные не подходят, пользователю отправляется запрос повторный попытки. Если повторная попытка не указана, повторно отправляется исходный запрос. Тем не менее если действие отправляется пользователю из проверяющего элемента управления, запрос повторной попытки не отправляется.

## Проверка запроса

Вы можете проверить ответ на запрос, прежде чем значение будет возвращено на следующем шаге каскада. Функция проверяющего элемента управления принимает параметр *контекста проверяющего элемента управления запроса* и возвращает логическое значение, которое подтверждает успешное прохождение проверки для входных данных. Контекст проверяющего элемента управления запроса поддерживает указанные ниже свойства.

Свойство	Описание
<i>Контекст</i>	Текущий контекст реплики для бота.
<i>Recognized</i>	<i>Результат распознавателя запроса</i> , который содержит сведения о входных данных после обработки распознавателем.
<i>Параметры</i>	Содержит <i>варианты выбора для запроса</i> , предоставленные в вызове, который запускал этот запрос.

Результат распознавателя запроса имеет описанные ниже свойства.

Свойство	Описание
<i>Успешно</i>	Указывает, смог ли распознаватель успешно выполнить синтаксический анализ входных данных.
<i>Значение</i>	Возвращаемое распознавателем значение. При необходимости это значение можно изменять в коде проверки.

## Использование диалогов

Диалоги можно рассматривать как структуру программного стека, который называется *стеком диалогов*. Управление стеком выполняет обработчик шагов диалога, и он же вызывается при пустом стеке. Самый верхний элемент в этом стеке считается *активным диалоговым окном*, а контекст диалогового окна направляет все входные данные активному диалоговому окну.

Когда начинается новый диалог, он становится активным диалогом, перемещаясь на вершину стека. Он остается активным, пока не завершится, не будет удален с помощью метода [replace dialog](#), или в стек не будет помещен новый диалог (из обработчика шагов или самим активным диалогом), который станет новым активным диалогом. Когда новый активный диалог завершится, он будет извлечен из стека, и активным снова станет диалог, расположенный на предыдущем уровне стека. Это позволяет [повторять](#) или [разветвлять](#) диалог, как описано ниже.

Вы можете начать или продолжить работу с корневым диалоговым окном, используя метод расширения диалогового окна *Run*. Из кода Вот вызов диалогового окна выполнение метода расширения либо возобновляет существующее диалоговое окно, либо запускает новый экземпляр диалогового окна, если стек в данный момент пуст. Элементы управления и ввод пользователя переходят в активное диалоговое окно в стеке.

Для доступа к состоянию диалогового окна метод *Run* требует наличия *метода доступа к свойству состояния*. Этот метод доступа создается и используется так же, как другие методы доступа к состоянию, но является свойством самого себя, основанным на состоянии диалога. См. дополнительные сведения об [управлении состоянием](#), а также о [последовательном потоке диалога](#).

В диалоговом окне у вас есть доступ к контексту диалогового окна, и его можно использовать для запуска других диалоговых окон, завершения текущего диалогового окна и выполнения других операций.

### Начало диалога

В диалоговом окне каскада передайте *идентификатор диалогового окна*, который требуется запустить, в диалоговом окне *Begin диалогового окна начало работы, запросили Замена*.

- Методы диалогового окна *Prompt* и *Begin* будут отправлять новый экземпляр диалогового окна, на который указывает ссылка, в верхнюю часть стека.
- С помощью метода *replace dialog* текущий диалог извлекается из стека и в стек помещается новый диалог. Замененный диалог будет отменен, а все сведения, содержащиеся в этом экземпляре, удалены.

Используйте параметр *options* для передачи сведений в новый экземпляр диалога. Параметры, передаваемые в новый диалог, можно получить из контекста шага с помощью свойства *options* на любом шаге диалога. См. дополнительные сведения о [создании сложного потока беседы с использованием ветвлений и циклов с примером кода](#).

### Продолжение диалога

В диалоговом окне каскада используйте свойство *Values* контекстного действия, чтобы сохранить

состояние между переходами. Любое значение, добавляемое в эту коллекцию на предыдущем шаге, будет доступно на следующих шагах. См. дополнительные сведения о [создании сложного потока беседы с использованием ветвлений и циклов с примером кода](#).

## Завершение диалога

В диалоговом окне каскада используйте метод *End Dialog* для завершения диалогового окна путем его освобождения из стека. *Конечный метод диалогового окна* может возвращать необязательный результат в родительский контекст (например, в диалоговом окне, вызвавшем его, или в обработчике включения Bot). Чаще всего этот метод вызывается из диалога для того, чтобы завершить текущий экземпляр этого диалога.

Вы можете вызвать метод *end* из любого места в контексте диалогов, но для бота этот вызов всегда будет выглядеть как поступивший из текущего активного диалога.

### TIP

Мы рекомендуем явным образом вызывать метод *end dialog* в конце диалога.

## Очистка всех диалогов

Если нужно извлечь все диалоги из стека, можно очистить стек диалога, вызвав метод *cancel all dialogs* для контекста диалога.

## Повторение диалога

Диалог можно заменить им же, создав цикл с помощью метода *replace dialog*. Это отличный способ обработки [сложных взаимодействий](#) и один прием для управления меню.

### NOTE

Если вам нужно сохранить внутреннее состояние текущего диалога, передайте сведения новому экземпляру диалога в вызове метода *replace dialog*, а затем соответствующим образом инициализируйте этот диалог.

## Ветвление диалога

Контекст диалога поддерживает стек диалогов и помнит следующий шаг для каждого диалога в стеке. Его метод *begin dialog* создает дочерний диалог и помещает его на вершину стека, а метод *end dialog* извлекает из стека верхний диалог. *End dialog* обычно вызывается из диалога, работу которого нужно завершить.

Диалог может начать новый диалог внутри того же набора диалогов, вызвав метод *begin dialog* контекста диалога и указав идентификатор нового диалога. При этом новый диалог становится текущим активным диалогом. Исходный диалог по-прежнему находится в стеке, но вызовы метода *continue dialog* для контекста диалога отправляются только тому диалогу, который находится на вершине стека, т. е. *активному диалогу*. Когда диалог извлекается из стека, контекст диалога продолжает выполнение со следующего шага в каскадной последовательности, хранящейся в стеке, на котором остановился исходный диалог.

Таким образом, можно создать ветвь в процессе общения, добавив шаг в один диалог, который на основе условия может выбрать другой диалог из набора доступных диалогов и начать его.

## Дополнительные сведения

- См. сведения об [адаптивных диалогах](#).
- См. сведения о [навыках](#).

# ПО промежуточного слоя

27.03.2021 • 12 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

ПО промежуточного слоя — это просто класс, который находится между адаптером и логикой бота, добавленный в коллекцию промежуточного программного обеспечения адаптера во время инициализации. Пакет SDK позволяет писать собственное или добавлять созданное другими пользователями ПО промежуточного слоя. Любые входящие или исходящие действия вашего бота будут передаваться через ПО промежуточного слоя.

Адаптер обрабатывает и направляет входящие действия через конвейер по промежуточного слоя в логику Bot, а затем обратно. Каждый поток действий входа и выхода бота, каждый компонент ПО промежуточного слоя может проверять или выполнять действие до и после запуска логики бота.

Прежде чем приступить к работе с ПО промежуточного слоя, важно понять суть [ботов в целом](#) и [как они обрабатывают действия](#).

## Использование ПО промежуточного слоя

Часто возникает следующий вопрос: "В каких случаях для реализации действий лучше выбрать ПО промежуточного слоя вместо обычной логики бота?". ПО промежуточного слоя расширяет интерактивные возможности ведения диалога с пользователями как перед, так и после обработки каждой *реплики*. Кроме того, ПО промежуточного слоя позволяет сохранять и извлекать сведения, касающиеся диалога, а также при необходимости вызывать дополнительную логику обработки. Ниже описаны некоторые распространенные сценарии применения ПО промежуточного слоя.

### Просмотр каждого действия и его выполнение

Есть множество ситуаций, которые требуют от бота соответствующего поведения для всех действий или только для отдельных действий определенного типа. Например, нам необходимо зарегистрировать каждое действие сообщения, которое получает бот, или предоставить резервный ответ, если бот еще не создал реплику ответа. ПО промежуточного слоя — это отличное место для этого со способностью действовать как до, так и после выполнения остальной части логики бота.

### Изменение или расширение контекста включения

Некоторые общения могут быть намного плодотворнее, если бот имеет больше информации, чем предусмотрено в этом действии. ПО промежуточного слоя в этом случае может просмотреть текущие сведения о состоянии беседы, запросить источник внешних данных и добавить это к объекту [контекста реплик](#) перед передачей выполнения логике бота.

В пакете SDK определено ПО промежуточного слоя для ведения журнала, которое позволяет записывать входящие и исходящие действия. Но вы можете определить собственное ПО промежуточного слоя.

## Конвейер ПО промежуточного слоя бота

Адаптер вызывает ПО промежуточного слоя для каждого действия в том порядке, в котором оно было добавлено. Адаптер передает объект контекста для возвращения и *следующего* делегата, а ПО промежуточного слоя вызывает делегат для передачи управления следующему промежуточному программному обеспечению в конвейере. ПО промежуточного слоя также имеет возможность выполнять действия после возвращения *следующего* делегата до завершения метода. Можно представить, что каждый объект ПО промежуточного слоя имеет первую и последнюю возможность

действовать в отношении объектов ПО промежуточного слоя, которые следуют за ним в конвейере.

Пример:

- обработчик выполнения переворачивания объекта первого по промежуточного слоя выполняет код перед вызовом *Next*:
  - обработчик «вращение» объекта по промежуточного слоя выполняет код перед вызовом *Next*.
  - Обработчик пошагового выполнения программы-бота выполняет функцию и возвращает.
  - обработчик «вращение» объекта по промежуточного слоя выполняет оставшийся код перед возвратом.
- обработчик переворачивания объекта первого по промежуточного слоя выполняет оставшийся код перед возвратом.

Если по промежуточного слоя не вызывает следующий делегат, адаптер не будет вызывать ни одно из последующего по промежуточного слоя или обработчиков последовательного выполнения, а также короткие цепи конвейера.

После завершения работы конвейера ПО промежуточного слоя бота возвращение завершается и его контекст выходит за пределы области.

ПО промежуточного слоя или бот могут создавать ответы и регистрировать обработчики событий ответа, но учитывайте, что ответы обрабатываются в отдельных процессах.

## Порядок выполнения ПО промежуточного слоя

Поскольку порядок добавления ПО промежуточного слоя определяет порядок, в котором оно обрабатывает действие, важно определить последовательность добавления ПО промежуточного слоя.

### NOTE

Это дает общий шаблон, который работает для большинства ботов, но имейте ввиду, что каждая часть ПО промежуточного слоя будет взаимодействовать с другими частями.

Первыми действиями в конвейере ПО промежуточного слоя должны быть те, которые берут на себя задачи самого низкого уровня, которые используются постоянно. Например, ведение журнала, обработка исключений и перевод. Их порядок зависит от потребностей. Скажем, вам нужно перевести входящее сообщение перед сохранением или же сообщение требуется сохранить до перевода (что может означать, что сохраненные сообщения не будут переведены).

Последним в конвейере ПО промежуточного слоя должен быть специальный бот ПО промежуточного слоя, который является программным обеспечением, реализующим выполнение обработки каждого сообщения, отправленного вашему боту. Если ПО промежуточного слоя использует информацию о состоянии или другую информацию, заданную в контексте бота, добавьте ее в конвейер программного обеспечения после того, как оно изменит состояние или контекст.

## Сокращение каналов

*Сокращение каналов* — важный фактор работы ПО промежуточного слоя и обработчиков ответов. Если выполнение должно продолжаться через последующие слои, нужно, чтобы ПО промежуточного слоя (или обработчик ответов) передало данные выполнения, вызвав *следующий* делегат. Если следующий делегат не вызывается в рамках ПО промежуточного слоя (или обработчика ответов), соответствующий конвейер сокращает канал и последующие слои не выполняются. Это означает, что вся логика ботов и любое ПО промежуточного слоя, определенное далее в конвейере, пропускается. Есть небольшое

различие между способами сокращения каналов для реплик в ПО промежуточного слоя и в обработчике ответов.

Когда промежуточное ПО сокращает канал реплики, обработчик реплик бота не вызывается, а весь код ПО промежуточного слоя, выполненный до этого момента в конвейере, продолжает выполняться до завершения.

Пропуск вызова *следующего* делегата в обработчике событий означает, что событие отменяется и значительно отличается от логики пропуска промежуточного программного обеспечения. Не обработав остальную часть события, адаптер никогда не отправит его.

#### TIP

Если вы сокращаете канал события ответа, например `SendActivities`, убедитесь, что такое поведение действительно требуется. Иначе возникнут трудности с исправлением этой ошибки.

## Обработчики событий ответа

В дополнение к логике приложения и ПО промежуточного слоя в контекстный объект могут быть добавлены обработчики ответов (иногда называемые обработчиками событий или обработчиками событий активности). Эти обработчики вызываются, когда связанный ответ происходит в текущем объекте контекста перед выполнением фактического ответа. Эти обработчики полезны, когда вы знаете, что хотите сделать до или после фактического события по каждому действию этого типа для остальной части текущего ответа.

#### WARNING

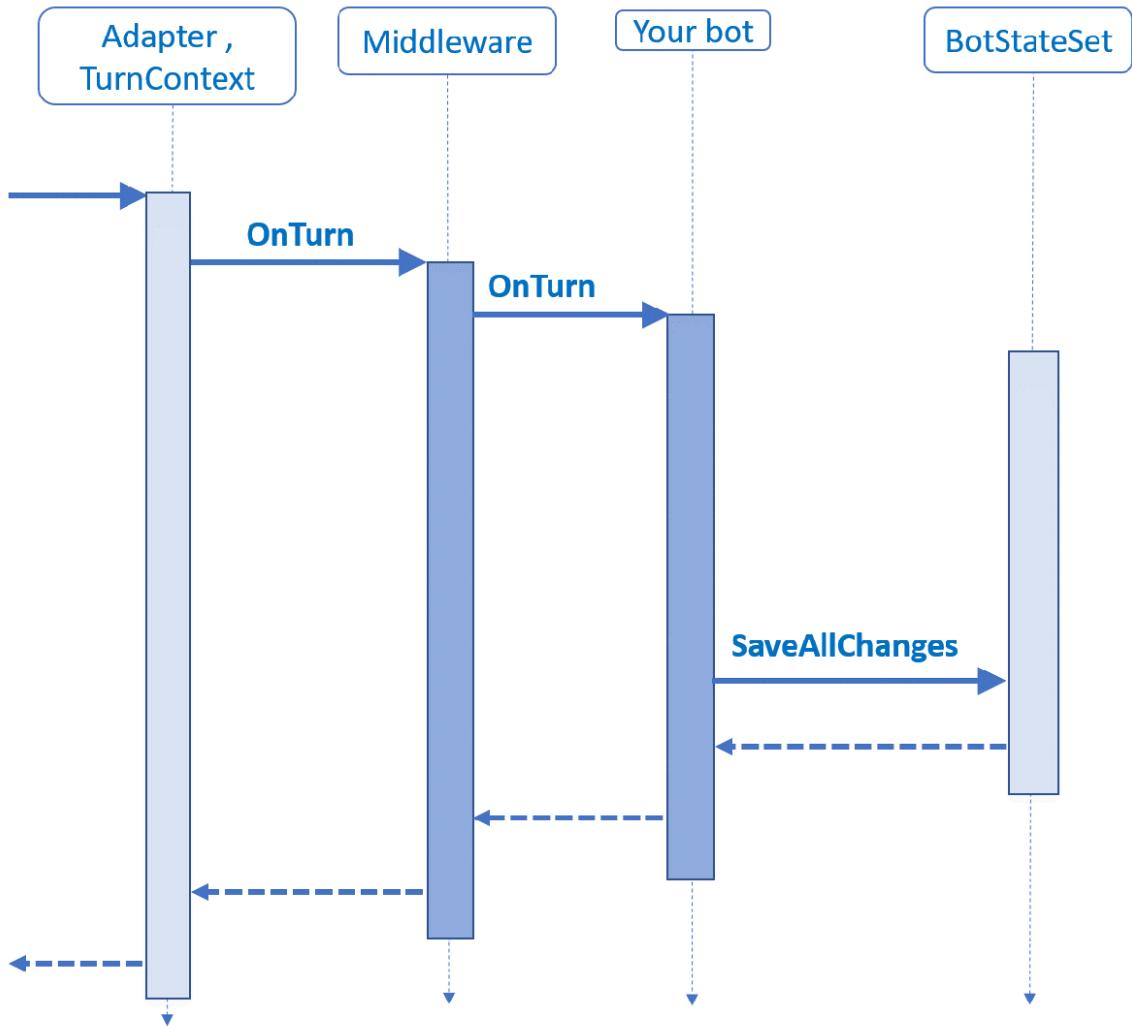
Ни в коем случае не вызывайте метод ответа на действие из своего соответствующего обработчика событий ответа (например, вызвав из обработчика отправки действий метод отправки действия). Таким образом можно создать бесконечный цикл.

Как вы помните, каждое новое действие получает новый поток для выполнения. Когда создается поток для обработки действия, список обработчиков для этого действия копируется в этот новый поток. Если нет добавленных обработчиков, то после этой точки будет выполняться определенное действие.

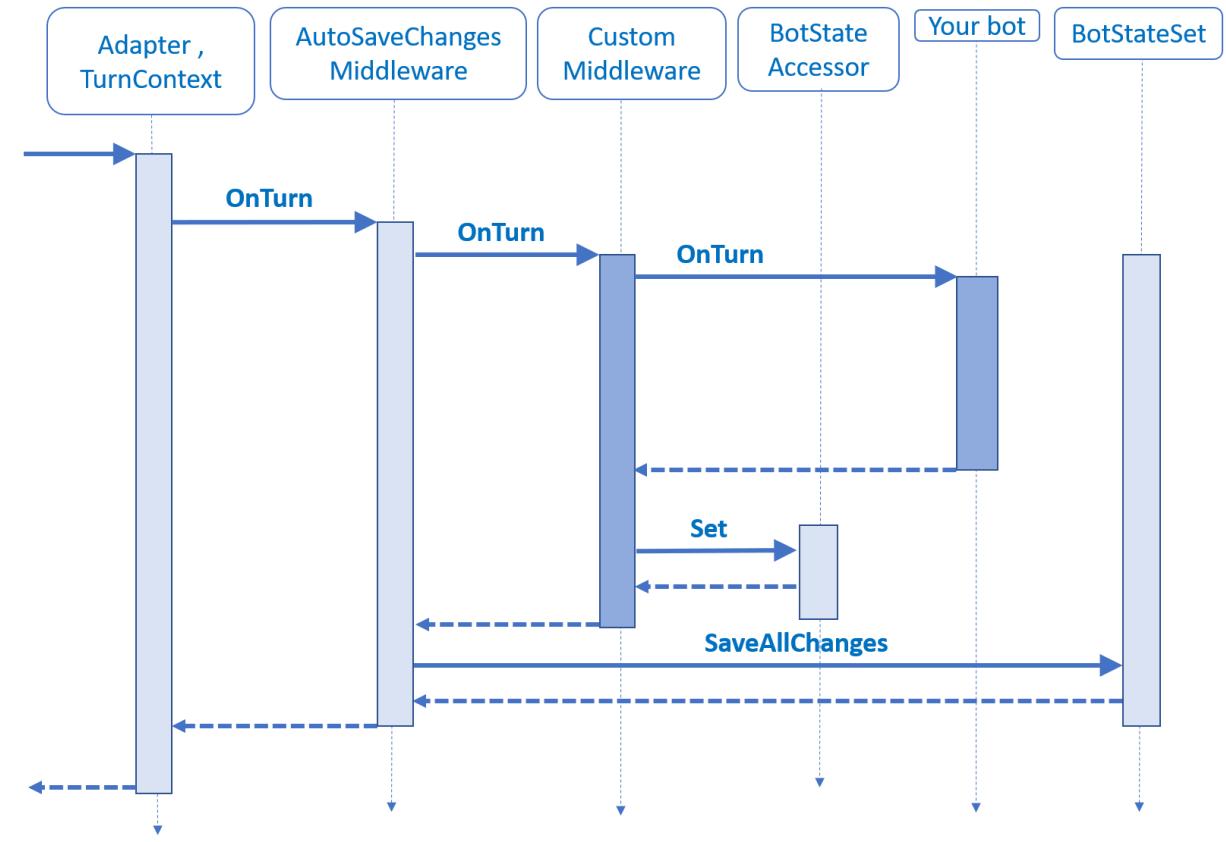
Адаптер управляет обработчиками, зарегистрированными в объекте контекста, почти так же, как конвейером ПО промежуточного слоя. Обработчики вызываются в том порядке, в котором добавлены. При команде "Далее" делегат передает управление следующему зарегистрированному обработчику событий. Если обработчик не вызывает следующий делегат, никакие последующие обработчики событий не вызываются, короткие цепи событий и адаптер не отправляют ответ в канал.

## Обработка состояний в ПО промежуточного слоя

Обычно для сохранения состояния в конце обработчика шагов выполняется вызов, сохраняющий изменения. Вы видите схему, на которой выделен вызов.



Проблема с этим подходом заключается в том, что любые обновления состояния, внесенные из какого-либо пользовательского промежуточного по, которое происходит после возврата обработчика поступающего, не будут сохранены в долговременное хранилище. Чтобы решить эту проблему, переместите вызов для сохранения изменений так, чтобы он выполнялся после пользовательского ПО промежуточного слоя. Для этого добавьте экземпляр `auto-save changes` в начало стека ПО промежуточного слоя или по меньшей мере до любых обращений к ПО промежуточного слоя, приводящих к изменению состояния. Такая реализация представлена в следующем примере.



Добавьте объекты управления состоянием, которым потребуется обновлять объект *набора состояний бота*, и примените их при создании ПО промежуточного слоя для автоматического сохранения изменений.

## Дополнительные ресурсы

Ознакомьтесь с реализацией ПО промежуточного слоя для ведения журнала расшифровки в пакете SDK Bot Framework [[C#](#) | [JS](#)].

# Описание структуры эхо-бота

27.03.2021 • 16 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Шаблоны и примеры для платформы Bot написаны для ASP.NET (C#), restify (JavaScript) и aiohttp (Python). Однако функции веб-службы не входят в пакет SDK для Bot Framework, но часть веб-платформы, которую вы решили использовать.

Все приложения-роботы совместно используют некоторые общие функции.

ФУНКЦИЯ	ОПИСАНИЕ
Подготовка ресурсов	Программа Bot в качестве веб-приложения должна создать веб-службу, интерфейс Bot и объект Bot.
Messaging endpoint (Конечная точка обмена сообщениями)	Веб-приложению необходимо реализовать конечную точку обмена сообщениями для получения действий и переадресации действий в адаптер Bot.
Адаптер Bot	Адаптер получает действия от конечной точки обмена сообщениями, пересыпает их обработчику передачи сообщений Bot и перехватывает все ошибки или исключения, которые логика Bot не перехватывает.
Объект Bot	Объект Bot обрабатывает причину или логику программы-робота для включения.

Можно создать эхо-робот из шаблонов, как описано в кратком руководстве (для [C#](#), [JavaScript](#) или [Python](#)), или скопировать проект ECHO Bot из репозитория [Microsoft/BotBuilder-Samples](#).

Шаблоны C# и JavaScript имеют встроенную поддержку потоковой передачи. В этой статье рассматриваются функции потоковой передачи. Сведения о потоковых подключениях см. в статье [Подключение программы-робота к прямой речи](#).

## Предварительные требования

- Базовые знания о [ботах](#).
- Копия образца echo Bot в [C#](#), [JavaScript](#) или [Python](#).

## Шаблоны Bot

Бот — это веб-приложение, а шаблоны для него доступны для каждого из трех языков.

- [C#](#)
- [JavaScript](#)
- [Python](#)

В состав платформы Bot входят шаблоны VSIX и DotNet.

Шаблоны создают веб-приложение [ASP.NET MVC Core](#). Основные компоненты для [ASP.NET](#) содержат похожий код в таких файлах, как `Program.cs` и `Startup.cs`. Эти файлы являются обязательными для всех веб-приложений и не зависят от конкретного бота.

## NOTE

Пакет [VSIX](#) включает версии .NET Core 2.1 и .NET Core 3.1 шаблонов C#. При создании ботов в Visual Studio 2019 следует использовать шаблоны .NET Core 3.1. В текущих примерах ботов используются шаблоны .NET Core 3.1. Примеры, использующие шаблоны .NET Core 2.1, можно найти в ветви [4.7-archive](#) репозитория BotBuilder-Samples. Сведения о развертывании .NET Core 3.1 программы-роботы в Azure см. в статье [развертывание программы Bot в Azure](#).

`appsettings.js` в файле указывает сведения о конфигурации для программы-робота, такие как идентификатор приложения и пароль, помимо прочего. Если вы применяете некоторые технологии или используете этот бот в рабочей среде, нужно добавить в эту конфигурацию определенные ключи или URL-адрес. Однако для этого эхо-робота сейчас вам не нужно ничего делать. в настоящее время идентификатор приложения и пароль могут остаться неопределенными.

В файле `echobot.csproj` указаны зависимости и связанные с ними версии для программы Bot. Все они настраиваются в системе и согласно шаблону. Дополнительные зависимости можно установить с помощью диспетчера пакетов NuGet или `dotnet add package` команды.

## Подготовка ресурсов

Программа Bot в качестве веб-приложения должна создать веб-службу, интерфейс Bot и объект Bot.

Многие программы-роботы также создают объекты уровня хранилища и управления памятью для Bot, но для вывода эха не требуется состояния. Другие программы-роботы также создают объекты, которые являются внешними по отношению к объекту Bot или адаптеру, который либо должен использовать.

- [C#](#)
- [JavaScript](#)
- [Python](#)

В ASP.NET вы регистрируете объекты и методы создания объектов в файле `Startup.cs`.

`ConfigureServices` Метод загружает подключенные службы, а также их ключи из `appsettings.js` или Azure Key Vault (если они есть), соединения и т. д. Здесь адаптеры и роботы определены для доступа посредством внедрения зависимостей. Затем `Configure` метод завершит настройку приложения.

`ConfigureServices` И `Configure` вызываются средой выполнения при запуске приложения.

## Messaging endpoint (Конечная точка обмена сообщениями)

Шаблон реализует веб-службу с конечной точкой обмена сообщениями. Служба извлекает заголовок проверки подлинности и полезные данные запроса и пересыпает их адаптеру.

Пакеты SDK для C# и JavaScript поддерживают потоковую передачу соединений. Хотя программа Echo Bot не использует функции потоковой передачи, адаптер в шаблоне предназначен для их поддержки.

Каждый входящий запрос представляет начало новой очереди.

- [C#](#)
- [JavaScript](#)
- [Python](#)

## Контроллеры \ ботконтроллер. CS

```
// This ASP Controller is created to handle a request. Dependency Injection will provide the Adapter and
IBot
// implementation at runtime. Multiple different IBot implementations running at different endpoints can be
// achieved by specifying a more specific type for the bot constructor argument.
[Route("api/messages")]
[ApiController]
public class BotController : ControllerBase
{
    private readonly IBotFrameworkHttpAdapter Adapter;
    private readonly IBot Bot;

    public BotController(IBotFrameworkHttpAdapter adapter, IBot bot)
    {
        Adapter = adapter;
        Bot = bot;
    }

    [HttpPost,HttpGet]
    public async Task PostAsync()
    {
        // Delegate the processing of the HTTP POST to the adapter.
        // The adapter will invoke the bot.
        await Adapter.ProcessAsync(Request, Response, Bot);
    }
}
```

## Адаптер бота

Адаптер получает действия от конечной точки обмена сообщениями, пересыпает их обработчику передачи сообщений Bot и перехватывает все ошибки или исключения, которые логика Bot не перехватывает.

Адаптер позволяет добавить собственный обработчик *ошибок включения*.

- [C#](#)
- [JavaScript](#)
- [Python](#)

`Startup.cs`.

Используемый адаптер определяется в `ConfigureServices` методе.

```
// Create the Bot Framework Adapter with error handling enabled.
services.AddSingleton<IBotFrameworkHttpAdapter, AdapterWithErrorHandler>();
```

`AdapterWithErrorHandler.cs`

```

public class AdapterWithErrorHandler : BotFrameworkHttpAdapter
{
    public AdapterWithErrorHandler(IConfiguration configuration, ILogger<BotFrameworkHttpAdapter> logger)
        : base(configuration, logger)
    {
        OnTurnError = async (turnContext, exception) =>
        {
            // Log any leaked exception from the application.
            // NOTE: In production environment, you should consider logging this to
            // Azure Application Insights. Visit https://aka.ms/bottelemetry to see how
            // to add telemetry capture to your bot.
            logger.LogError(exception, $"[OnTurnError] unhandled error : {exception.Message}");

            // Send a message to the user
            await turnContext.SendActivityAsync("The bot encountered an error or bug.");
            await turnContext.SendActivityAsync("To continue to run this bot, please fix the bot source
code.");

            // Send a trace activity, which will be displayed in the Bot Framework Emulator
            await turnContext.TraceActivityAsync("OnTurnError Trace", exception.Message,
"https://www.botframework.com/schemas/error", "TurnError");
        };
    }
}

```

## Логика Bot

Программа Echo Bot использует *обработчик действий* и реализует обработчики для типов действий, которые он распознает и реагирует на, в данном случае на действия *обновления диалога и сообщений*.

- Действие обновления диалога включает сведения о том, кто присоединился к беседе или оставил ее. Для бесед, отличных от групп, как Bot, так и пользователь присоединяются к беседе при запуске. При обмене данными с группами происходит обновление диалога, когда кто-то присоединяется или оставляет диалог, будь это Bot или пользователь.
- Действие Message представляет сообщение, которое пользователь отправляет в Bot.

Программа Echo Bot приветствует пользователя, когда он присоединяется к беседе и возвращает сообщения, отправляемые им в Bot.

- C#
- JavaScript
- Python

### Startup.cs.

Используемый робот определяется в `ConfigureServices` методе.

```

// Create the bot as a transient. In this case the ASP Controller is expecting an IBot.
services.AddTransient<IBot, EchoBot>();

```

### Bots\EchoBot.cs

```
public class EchoBot : ActivityHandler
{
    protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
    {
        var replyText = $"Echo: {turnContext.Activity.Text}";
        await turnContext.SendActivityAsync(MessageFactory.Text(replyText, replyText), cancellationToken);
    }

    protected override async Task OnMembersAddedAsync(IList<ChannelAccount> membersAdded,
ITurnContext<IConversationUpdateActivity> turnContext, CancellationToken cancellationToken)
    {
        var welcomeText = "Hello and welcome!";
        foreach (var member in membersAdded)
        {
            if (member.Id != turnContext.Activity.Recipient.Id)
            {
                await turnContext.SendActivityAsync(MessageFactory.Text(welcomeText, welcomeText),
cancellationToken);
            }
        }
    }
}
```

## Дальнейшие действия

- Сведения о том, как [отправлять и получать текстовые сообщения](#)
- Сведения о том, как [Отправить приветственные сообщения пользователям](#)

# Общие сведения об адаптивных диалогах

27.03.2021 • 14 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Адаптивные диалоговые окна предлагают новое добавление на основе событий в [библиотеку диалоговых окон](#), которое позволяет легко разложить сложные методы управления диалогами, такие как обработка [прерываний](#), диспетчеризация и многое другое.

## IMPORTANT

В настоящее время адаптивные диалоговые окна доступны только в версии .NET пакета SDK для Bot Framework. Образец программы-роботы, созданный с помощью адаптивных диалоговых окон, можно найти в [репозитории BotBuilder-Samples](#) в GitHub.

## Предварительные требования

- Представление об использовании [диалогов](#) в пакете SDK для Bot Framework версии 4.
- Общее представление о [запросах](#) в пакете SDK для Bot Framework версии 4.

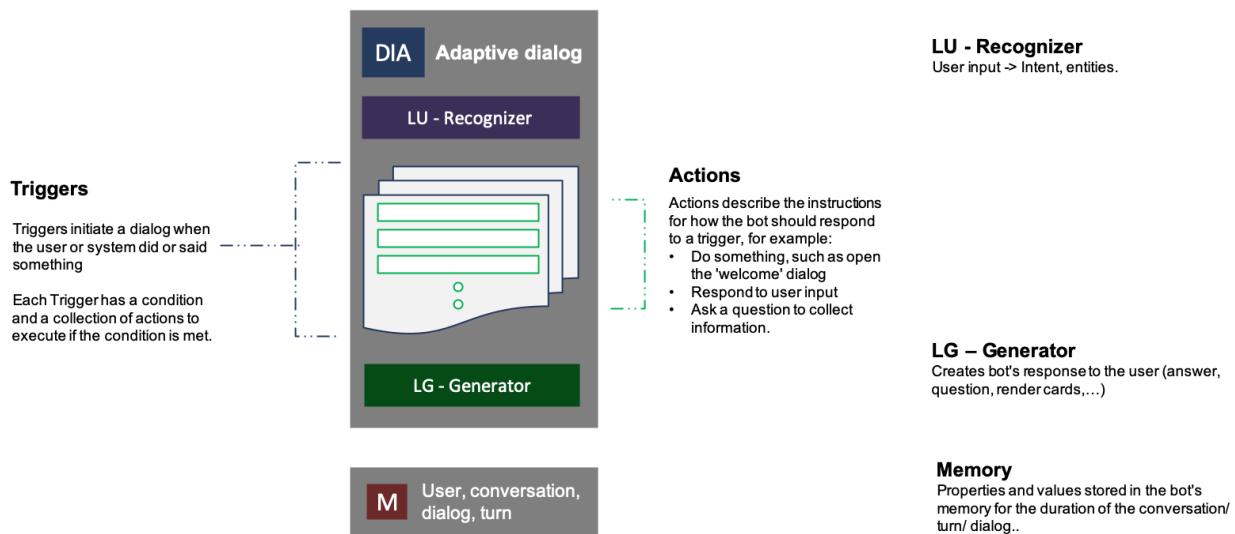
## Определение адаптивных диалогов

### Почему адаптивные диалоговые окна

Адаптивные диалоговые окна имеют много преимуществ для [вательфалдиалогс](#). В основном они:

- Обеспечить гибкость, позволяющую динамически обновлять поток бесед на основе контекста и событий. Это особенно удобно при работе с переключениями контекста диалога и [прерываниями](#) в середине диалога.
- Поддержка и направляются поверх обширной системы обработки событий для диалоговых окон, поэтому можно гораздо проще описать и контролировать семантику [моделирования](#), отмены и семантическое планирование выполнения.
- Перенос входных данных и обработка событий на основе правил
- Объедините модель диалога (диалоговое окно) и формирование выходных данных в единую единую автономную единицу.
- Поддержка точек расширения для распознавания, правил событий и машинного обучения.
- Изначально были разработаны как декларативные объекты. Это позволяет использовать различные инструменты, включая такие продукты, как [Bot Framework](#), которые предоставляют визуальные полотна для моделирования общения.

## Анатомия адаптивного диалога



## Триггеры

Все адаптивные диалоги содержат список, содержащий один или несколько обработчиков событий, называемых *триггерами*. Каждый триггер содержит необязательный **Condition** и список из одного или нескольких **действий**. Триггеры позволяют перехватывать события и реагировать на них. Если **Condition** выполняет условия **Actions**, то триггер активируется, а после обработки события дальнейшие действия с этим событием не выполняются. Если условия активации триггеров **Condition** не выполняются, событие передается следующему обработчику событий для вычисления.

Дополнительные сведения об использовании *триггеров* в адаптивных диалогах см. в разделе [События и триггеры в адаптивных диалогах](#).

## Действия

*Действия* определяют поток общения, когда триггер перехватывает определенное событие. В отличие от каскадного диалога, в котором каждый шаг является функцией, каждое действие в адаптивном диалоге само по себе является диалогом. Это делает адаптивные диалоговые окна как мощными, так и гибкими и позволяет адаптивным диалоговым окнам легко управлять [прерываниями](#) и условиями ветвления на основе контекста или текущего состояния.

Пакет SDK для Bot Framework предоставляет множество встроенных действий, позволяющих выполнять различные операции, такие как работа с памятью, управление диалогами и управление потоком общения бота. Так как действия — это, по сути, диалоги, они являются расширяемыми, что позволяет создавать собственные пользовательские действия.

Дополнительные сведения об использовании *действий* в адаптивных диалогах см. в разделе [Действия в адаптивных диалогах](#).

## Входные данные

*Входные данные* относятся к адаптивным диалогам, в которых используются [запросы](#) к базовому классу диалога. Входные данные — это специализированные действия, которые можно использовать в адаптивном диалоге для запроса и проверки сведений от пользователя, а затем, если проверка пройдена, для передачи этих входных данных в память. Все классы входных данных в пакете SDK для Bot Framework предназначены для следующих задач.

- Выполнение проверок существования перед запросом, чтобы избежать запроса сведений, которые уже есть у бота.
- Сохранение входных данных в указанное свойство, если они соответствуют ожидаемому типу сущности.
- Принятие ограничений — минимум, максимум и т. д.

Дополнительные сведения о *входных данных* в адаптивных диалоговых окнах см. в статье [запрос ввода пользователя с помощью адаптивных диалоговых окон](#).

## Распознаватели

*Распознаватели* позволяют боту понять и извлечь значимые части информации из входных данных пользователя. Все распознаватели выдают события, такие как событие `recognizedIntent`, которое возникает, когда распознаватель выбирает намерение (или извлекает сущности) из речевого фрагмента пользователя. Использовать распознаватели в адаптивном диалоге необязательно, но если их не использовать, то события `recognizedIntent` никогда не возникнут, а вместо них будет срабатывать событие `unknownIntent`.

Дополнительные сведения об использовании *распознавателей* в адаптивных диалогах см. в разделе [Распознаватели в адаптивных диалогах](#).

## Генератор

*Генератор* привязывает определенную систему создания речи к адаптивному диалогу. Наряду с распознавателем это позволяет четко отделить и инкапсулировать ресурсы распознавания речи и создания речи определенного диалога. Используя функцию [создания речи](#), можно связать генератор с *LG*-файлом или задать для генератора экземпляр `TemplateEngine`, в котором вы явным образом управляете одним или несколькими *LG*-файлами, обеспечивающими работу адаптивного диалога.

Дополнительные сведения об использовании *генераторов* в адаптивных диалогах см. в разделе [Создание речи в адаптивных диалогах](#).

## Области памяти и управление состоянием

Адаптивные диалоги дают возможность обращаться к памяти и управлять ею. По умолчанию все адаптивные диалоги используют эту модель, так что все компоненты, использующие память или участвующие в ее работе, применяют общий способ чтения и записи информации в соответствующей области. Все свойства во всех областях — это контейнеры свойств, которые дают возможность динамически изменять перечень свойств, которые сохраняются.

Дополнительные сведения об *областях памяти и управлении состоянием* в адаптивных диалогах см. в разделе [Области памяти и управление состоянием](#).

## Декларативные ресурсы

Адаптивные диалоги позволяют определить диалог как класс, создав новый объект `AdaptiveDialog` и определив триггеры и действия в исходном файле `classes`. Можно также создать диалог, используя декларативный подход, при котором вы определяете все атрибуты диалога в `JSON`-файле с расширением "DIALOG". Для определения диалогов исходный код не требуется, и в одном боте может быть несколько диалогов, созданных обоими способами. Во время выполнения бот создаст и выполнит код диалога в соответствии с определением в этих декларативных файлах диалога.

Дополнительные сведения об использовании *декларативных ресурсов* в адаптивных диалоговых окнах см. в статье [Использование декларативных ресурсов](#).

# Сборка

## Поведение среды выполнения адаптивного диалога

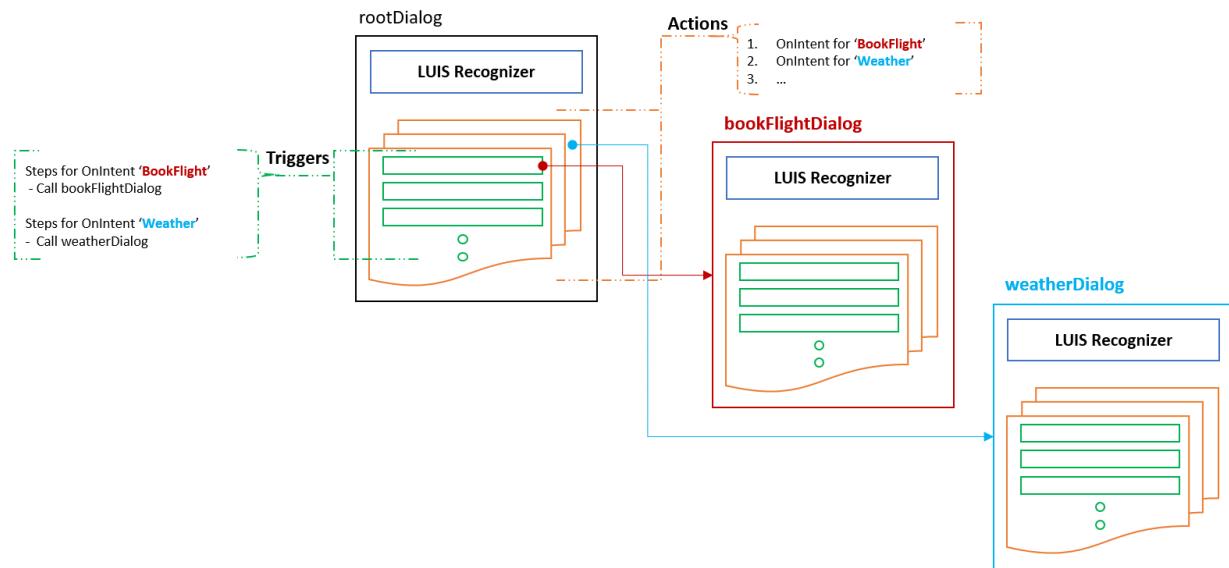
Приведенный ниже фиктивный *бот туристического агентства* поможет проиллюстрировать поведение среды выполнения адаптивных диалогов. Реальное приложение будет иметь несколько возможностей, таких как возможность поиска и бронирования авиабилетов, номеров в отелях, автомобилей и даже проверка погоды, и для каждой из этих возможностей будет использоваться собственный специальный диалог.

Что происходит, когда пользователь делает что-то непредвиденное, общаясь с ботом?

Рассмотрим следующий сценарий.

```
User: I'd like to book a flight  
Bot: Sure. What is your destination city?  
User: How's the weather in Seattle?  
Bot: Its 72 and sunny in Seattle  
...
```

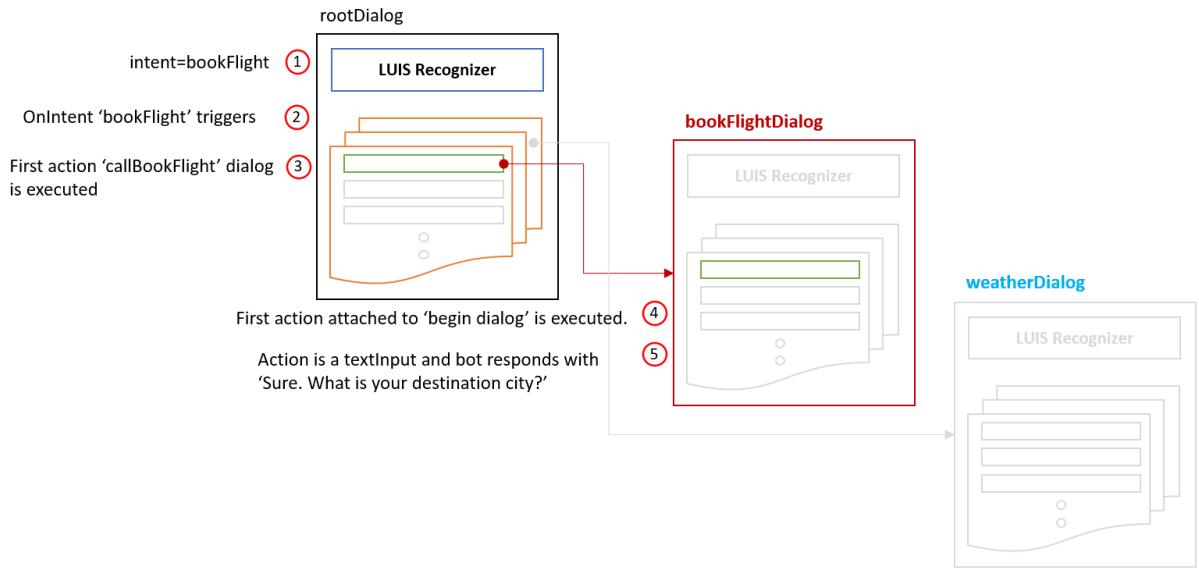
Пользователь не только не ответил на вопрос, но и полностью сменил тему, что потребует совершенно другого кода (действия), который существует в другом диалоге. Адаптивные диалоги позволяют справиться с этим сценарием, как показано на следующей схеме.



Этот бот содержит три приведенных ниже адаптивных диалога.

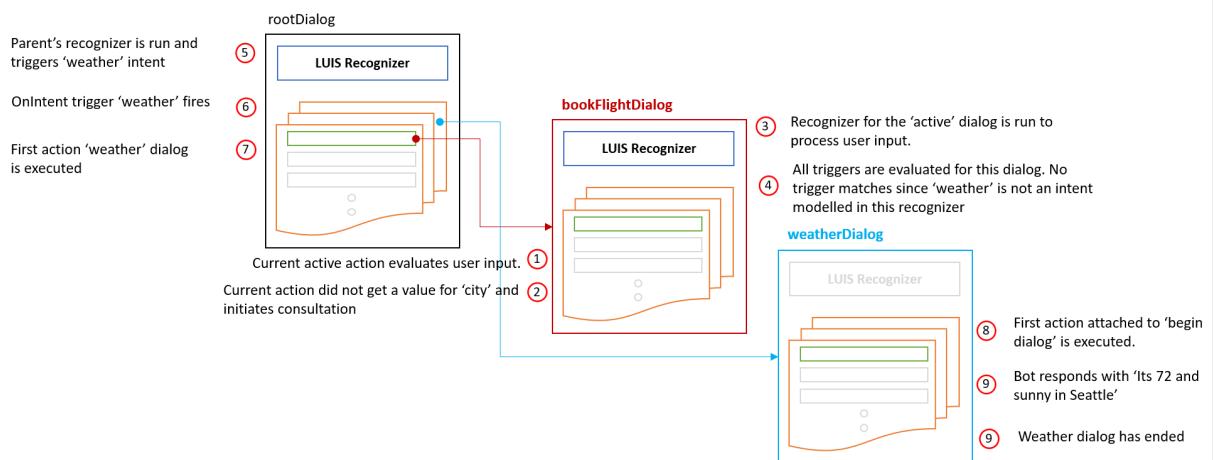
1. Диалог **rootDialog** с собственной моделью LUIS, набором триггеров и действий, некоторые из которых будут вызывать дочерний диалог для обработки определенных запросов пользователей.
2. Диалог **bookFlightDialog** с собственной моделью LUIS, набором триггеров и действий, которые обрабатывают беседы, связанные с бронированием авиабилетов.
3. Диалог **weatherDialog** с собственной моделью LUIS, набором триггеров и действий, которые обрабатывают беседы о получении сведений о погоде.

Ниже приведен поток общения, когда пользователь говорит: *I'd like to book a flight* (Я хочу зарезервировать авиабилеты)



Распознаватель активного диалога (`rootDialog`) создает событие `recognizedIntent`, которое можно реализовать с помощью триггера `OnIntent`. В этом случае пользователь сказал: «*мне хотелось бы зарезервировать рейс*», который соответствует намерению, определенному в `rootDialog`, и приводит к тому `OnIntent`, что триггер содержит действие `BeginDialog` для выполнения, которое вызывает диалоговое окно `bookFlightDialog`. Диалог бронирования авиабилетов выполняет свои действия, одним из них является вопрос о городе, в который вы хотите полететь.

Пользователь может ввести в ответ что угодно. В некоторых случаях ответ может быть вообще не связан с вопросом, и в данном случае пользователь ответил *How's the weather in Seattle?* (Какая погода в Сиэтле?)



Так как в диалоге `bookFlightDialog` нет триггера `OnIntent` для обработки запроса пользователя, бот перемещает входные данные этого пользователя вверх по стеку диалога, минуя все диалоги вызова и возвращаясь к корневому диалогу. В данном случае это всего на один диалог выше, так как в `rootDialog` имеется триггер `OnIntent` для обработки намерения `weather`. Этот триггер выполняет свое действие `BeginDialog`, вызывающее диалог `weatherDialog`, в который передается вопрос пользователя. После того как `weatherDialog` завершает работу и отвечает на вопрос пользователя, бот возвращает управление исходному диалогу, а поток общения продолжается оттуда, где он ранее *прервался*, и пользователю снова предлагается ввести город назначения.

Итог:

Распознаватель каждого диалога анализирует входные данные пользователя, чтобы определить его намерение. После определения намерения распознаватель создает событие `IntentRecognized`, обрабатываемое диалогом с помощью триггера `OnIntent`. Если в активном диалоге нет триггера

`onIntent`, который может обработать это намерение, то бот отправляет это намерение в родительский диалог. Если в родительском диалоге нет триггера для обработки намерения, оно будет передано выше, пока не достигнет корневого диалога. После того как триггер, обрабатывающий это намерение, будет выполнен, он вернет управление диалогу, запустившему этот процесс, и в нем можно будет продолжить поток общения с того самого момента, на котором он был прерван.

## Дополнительные сведения

### Принципы адаптивности

- События и триггеры в адаптивных диалогах
- Действия в адаптивных диалогах
- Запрос ввода данных пользователем с помощью адаптивных диалогов
- Распознаватели в адаптивных диалогах
- Создание текста в адаптивных диалогах
- Области памяти и управление состоянием в адаптивных диалогах
- Использование декларативных ресурсов

### Разработка программы-бота с помощью адаптивных диалоговых окон

- Создание бота с использованием адаптивных диалогов
- Создание бота с использованием адаптивных, компонентных, каскадных и настраиваемых диалогов

# События и триггеры в адаптивных диалогах.

27.03.2021 • 12 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Адаптивные диалоги предлагают новый подход на основе событий для моделирования общения. Любая подсистема в Bot может выдавать события, а все адаптивные диалоговые окна содержат один или несколько обработчиков событий, называемых *триггерами*, которые позволяют реагировать на эти события. При каждом порождении события вычисляются триггеры активного адаптивного диалога и, если какой-либо триггер соответствует текущему событию, выполняются [действия](#), связанные с этим триггером. Если событие не удалось обработать в активном диалоге, оно передается в родительский диалог для вычисления. Этот процесс продолжится до тех пор, пока событие не будет обработано или не достигнет корневого диалога бота. Если обработчик событий (*триггер*) не найден, событие будет пропущено и никакие действия не будут выполнены.

## Предварительные требования

- [Принципы работы бота](#)
- [Общие сведения об адаптивных диалогах](#)
- [Библиотеки диалогов](#)

## Структура триггера

Триггер состоит из условия и одного или нескольких действий. Пакет SDK для Bot Framework предлагает несколько триггеров, каждый из которых содержит набор предопределенных условий, которые проверяют либо eventName, либо eventValue. В триггер можно добавить дополнительные условия, обеспечив расширенный контроль над его выполнением.

Пакет SDK для Bot Framework предоставляет различные предопределенные триггеры, предназначенные для обработки распространенных типов событий. Например, триггер `OnIntent` срабатывает, когда распознаватель обнаруживает [намерение](#). При использовании распознавателя [LUIS](#) также возвращается [оценка прогноза](#), которая указывает степень достоверности результатов прогноза. Чтобы повысить надежность и точность бота, может потребоваться выполнять триггер `OnIntent`, только если степень достоверности составляет не меньше 80 %. Это можно сделать, добавив [условие](#). Все триггеры содержат необязательное свойство `Condition`. Если оно определено, то оно должно иметь значение `true`, чтобы триггер выполнился. Свойство `Condition` является строкой, но чтобы оно работало, это свойство должно содержать допустимое [адаптивное выражение](#). Пример приведенного выше свойства `Condition` будут выглядеть примерно так: `Condition = "#<IntentName>.Score >= 0.8"`. Адаптивные выражения позволяют задавать сложные условия, которые делают возможной реализацию практически любого сценария.

Все триггеры также содержат список *действий*. Действия представляют операции, выполняемые ботом в ответ на срабатывание триггера. Это основа триггера. Узнать больше о действиях и основанных на них элементах в пакете SDK для Bot Framework можно в разделе [Действия в адаптивных диалогах](#).

## Типы триггеров

*Триггеры* позволяют перехватывать события и реагировать на них. Самым широким триггером, от которого наследуются все другие триггеры `OnCondition`, является триггер, который позволяет перехватывать и присоединять список действий, выполняемых при порождении определенного события

любой из подсистем программы-роботы.

Список триггеров приведен в следующих разделах, где они классифицированы и сгруппированы по типу.

В таблицах в следующих разделах перечислены все триггеры, поддерживаемые адаптивными диалогами, которые в настоящее время включены в пакет SDK, а также события, на которых они основаны.

## Базовый триггер

Триггер `OnCondition` является базовым триггером, от которого наследуют все остальные триггеры. При определении триггеров в адаптивном диалоговом окне они определяются как список `OnCondition` объектов.

Дополнительные сведения и пример см. в разделе [базовый триггер](#) в статье о встроенных триггерах адаптивных диалогов.

## Триггеры событий распознавателя

Распознаватели извлекают значимые фрагменты информации из входных данных пользователя в виде *намерений* и *сущностей*, после чего они порождают события. Например, событие `recognizedIntent` порождается, когда распознаватель выбирает намерение (или извлекает сущности) из заданного речевого фрагмента пользователя. Вы обрабатываете эти события с помощью `OnIntent` триггера.

В следующем списке показаны некоторые *триггеры событий распознавателя*, доступные в пакете SDK для Bot Framework:

- **Выберите намерение.** `OnChooseIntent` Триггер выполняется при неоднозначности между определениями из нескольких распознавателей в [кросstrainedреконизерсет](#).
- **Распознано намерение.** `OnIntent` Триггер выполняется при распознавании указанной цели.
- **QnA совпадение.** `OnQnAMatch` Триггер выполняется, когда [кнамакерреконизер](#) возвращает `QnAMatch` намерение.
- **Не удалось распознать цель.** `OnUnknownIntent` Триггер выполняется, когда входные данные пользователя не распознаны или не найдены совпадения ни в одном из `OnIntent` триггеров. Его также можно использовать в качестве первого триггера в корневом диалоге вместо `OnBeginDialog`, чтобы выполнить все необходимые задачи при первом запуске диалога.

Подробные сведения и примеры см. в разделе [триггеры событий распознавателя](#) в статье о встроенных триггерах адаптивных диалогов.

## Триггеры событий диалога

Триггеры диалогового окна обрабатывают события, относящиеся к диалоговому окну, которые связаны с жизненным циклом диалогового окна. В настоящее время в пакете SDK для Bot Framework доступны 6 триггеров диалога, и все они являются производными от класса `OnDialogEvent`.

### TIP

Они отличаются от обычных обработчиков событий прерывания, в которых действия дочерних элементов продолжают выполняться после завершения действий обработчиков. Для каждого события в боте выполняется новый набор действий, по завершении которого завершается и этап диалога.

Для триггера диалогового окна:

- Выполнить действие немедленно при запуске диалогового окна, даже перед вызовом распознавателя, используйте `OnBeginDialog` триггер.
- Запретить отмену диалога, когда любое из его дочерних диалоговых окон выполняет `CancelAllDialogs` действие, используйте `OnCancelDialog` триггер.
- Предпринимать действия, если все действия и события неоднозначности обработаны, используйте

`OnEndOfActions` триггер.

- Обработайте условие ошибки, используя `OnError` триггер.

Подробные сведения и примеры см. в разделе [события диалога](#) в статье о встроенных триггерах адаптивных диалогов.

### Триггеры событий действия

Триггеры действия позволяют связать действия с любым входящим действием клиента. Например, когда новый пользователь присоединяется и бот начинает новую беседу. Дополнительные сведения о действиях можно найти в разделе [Схема действия в Bot Framework](#).

Все события действия имеют базовое событие `ActivityReceived`, которое дополнительно уточняется с помощью *типа действия*. Базовый класс, от которого наследуются все триггеры действия, — `OnActivity`.

- Обновление диалога.** Используйте этот параметр для управления событиями, возникающими, когда пользователь начинает новый диалог с Bot.
- Диалог завершен.** Действия, выполняемые при получении действия типа `EndOfConversation`.
- Событие получено.** Действия, выполняемые при получении действия типа `Event`.
- Передачи вам образа для человека.** Действия, выполняемые при получении действия типа `HandOff`.
- Вызван диалог.** Действия, выполняемые при получении действия типа `Invoke`.
- Пользователь вводит.** Действия, выполняемые при получении действия типа `Typing`.

Подробные сведения и примеры см. в разделе [действия триггеров событий](#) в статье о встроенных триггерах адаптивных диалогов.

### Триггеры событий сообщения

Триггеры **событий сообщения** позволяют реагировать на любое событие сообщения, например, когда сообщение изменено (`MessageUpdate`) либо удалено (`MessageDeletion`), или когда кто-либо реагирует (`MessageReaction`) на сообщение (например, к распространенным реакциям на сообщение относятся такие действия, как отметка "Нравится", "Сердечко", "Смех", "Удивление" и "Злость").

События сообщения относятся к типу события действия, поэтому все события сообщения имеют базовое событие `ActivityReceived`, которое дополнительно уточняется с помощью *типа действия*. Базовый класс, от которого наследуются все триггеры сообщения, — `OnActivity`.

- Сообщение получено.** Действия, выполняемые при получении действия типа `MessageReceived`.
- Сообщение удалено.** Действия, выполняемые при получении действия типа `MessageDelete`.
- Реакция на сообщение.** Действия, выполняемые при получении действия типа `MessageReaction`.
- Сообщение Обновлено.** Действия, выполняемые при получении действия типа `MessageUpdate`.

Подробные сведения и примеры см. в разделе [сообщения о триггерах событий](#) в статье о встроенных триггерах адаптивных диалогов.

### Триггер настраиваемого события

Можно порождать собственные события, добавив действие `EmitEvent` в любой триггер. Затем можно будет обработать это пользовательское событие в любом триггере любого диалога бота, определив триггер *настраиваемого события*. Триггер настраиваемого события — это триггер `OnDialogEvent`, который фактически становится настраиваемым триггером, если для свойства `Event` задано то же значение, что для свойства `EventName` действия `EmitEvent`.

**TIP**

Можно разрешить другим диалогам бота работать с настраиваемым событием, задав для свойства `BubbleEvent` действия `EmitEvent` значение `true`. Дополнительные сведения и пример см. в разделе [пользовательские события] [Custom-events] статьи о встроенных триггерах адаптивных диалогов.

Подробные сведения и пример см. в разделе [пользовательские триггеры событий] [Custom-Event-Triggers] статьи о встроенных триггерах адаптивных диалогов.

## Дополнительные сведения

- [Общие сведения об адаптивных диалогах](#)
- [Библиотеки диалогов](#)
- [Действия в адаптивных диалогах](#)
- Дополнительные сведения о триггерах в адаптивных диалоговых окнах, включая примеры, см. в статье [триггеры в адаптивных диалоговых окнах — справочное руководство](#).

# Действия в адаптивных диалогах

27.03.2021 • 11 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Действия помогают создавать и обслуживать поток беседы бота после получения события [триггером](#). Адаптивные диалоги содержат списки триггеров, а триггеры, в свою очередь, содержат списки действий, которые будут выполняться после срабатывания триггера для решения необходимых задач, например для обработки запроса пользователя. Помимо создания и обслуживания потока беседы бота, действия можно применить для отправки сообщений, предоставления пользователям ответов на вопросы из [базы знаний](#), математических или любых других вычислений для пользователя. В адаптивных диалогах путь развития беседы бота может ветвиться и зацикливаться. Бот может задавать вопросы и отвечать на них, проверять входные данные пользователя, обрабатывать и хранить в [памяти](#) значения и принимать решения на основе введенных пользователем данных.

## IMPORTANT

Действия по сути являются диалогами, и в качестве действия можно применить любой диалог, что дает вам мощные и гибкие средства для создания полнофункционального и надежного бота. В пакет SDK для Bot Framework уже включен обширный набор действий, но вы можете дополнить их собственными настраиваемыми действиями, что позволит выполнять практически любые специализированные задачи и процессы.

## Предварительные требования

- [Общие сведения об адаптивных диалогах](#)
- [События и триггеры в адаптивных диалогах](#)

## Действия

Действия, которые входят в состав пакета SDK для Bot Framework, позволяют использовать условное выполнение кода:

- ветвление и циклы;
- задачи управления диалогами, например запуск нового диалога, завершение, отмена или повтор уже запущенного диалога;
- задачи управления памятью, например создание, удаление или изменение свойства, сохраненного в [памяти](#);
- доступ к внешним ресурсам, например отправка [HTTP-запросов](#);
- выполнение [запросов OAuth на вход](#) и многое другое.

## TIP

В отличие от каскадного диалога, в котором каждый шаг является функцией, в адаптивном диалоге каждое действие сохраняет все функциональные возможности, мощь и гибкость полноценного диалога. Благодаря такой архитектуре адаптивные диалоги:

- Предоставляют более простой способ обработки [прерываний](#).
- поддерживают условное ветвление по параметрам контекста или состояния.

Адаптивные диалоги поддерживают выполнение следующих действий.

## Действия

Используется для отправки любых действий, например реагирования на пользователя.

- **Действие Send.** Отправка любых действий, например ответов пользователю.
- **Обновление действия.** Позволяет обновить отправленное действие.
- **Делетеактивити.** Позволяет удалить отправленное действие.
- **Получение элементов действия.** Позволяет получить список элементов действия и сохранить их в свойство в [памяти](#).

Подробные сведения и примеры см. в разделе [действия](#) в статье [действия в адаптивных диалоговых окнах — справочное руководство](#).

## Запрос данных пользователя

Пакет SDK для Bot Framework определяет различные действия, используемые для сбора и проверки вводимых пользователем данных. Эти важные и специализированные действия описаны в статье [запрос вводимых пользователем данных с помощью адаптивных диалоговых окон](#).

## Создание условия

Условные операторы, такие как операторы If и циклы for, позволяют программам-роботам принимать решения и управлять потоком диалога. Эти условия задаются набором условных операторов с логическими выражениями, которые возвращают логическое значение true или false.

Ниже перечислены условные действия, предоставляемые пакетом SDK для Bot Framework.

- **If/else.** Используется для создания инструкций If и If-Else, которые используются для выполнения кода, только если указанное условие имеет значение true.
- **Коммутатор.** Позволяет создать меню с множественным выбором.
- **Цикл for each.** Циклический перебор набора значений, хранящихся в массиве.
- **Для каждого цикла страницы.** Постстраничный циклический перебор набора значений, хранящихся в массиве.
  - **Выход из цикла.** Используйте, чтобы прервать цикл.
  - **Продолжить цикл.** Используется для продолжения цикла.
- **Goto.** Немедленно переходит к указанному действию и возобновляет выполнение. Действие определяется значением actionId.

Подробные сведения и примеры см. в разделе [условные операторы](#) раздела [действия в адаптивных диалоговых окнах — справочное руководство](#).

## Управление диалогами

Действия по управлению диалоговыми окнами позволяют управлять любым действием, связанным с диалоговым окном. действия по управлению диалоговыми службами, предоставляемые пакетом SDK для Bot Framework, включают:

- **Начать новое диалоговое окно.** Это запустит выполнение другого диалогового окна. Когда диалог завершится, будет возобновлено выполнение текущего триггера.
- **Отмена диалогового окна.** Отменяет активный диалог. Используется, когда нужно немедленно закрыть диалог, даже не завершая текущий процесс.
- **Отмена всех диалоговых окон.** Отменяет все активные диалоги, включая родительские. Используйте это действие, если нужно извлечь все диалоги из стека, а очистить стек диалогов можно с помощью метода отмены всех диалогов в контексте диалога. Создает событие `CancelAllDialogs`.
- **Завершение этого диалогового окна.** Завершает активный диалог. Используйте это действие, если перед выходом нужно завершить работу диалога и возвратить результаты. Создает событие

`EndDialog`.

- **Завершение диалогового окна.** Завершает текущий шаг диалога, не закрывая сам диалог.
- **Повторите это диалоговое окно.** Используется для перезапуска родительского диалога.
- **Замените это диалоговое окно.** Заменяет текущее диалоговое окно новым диалоговым окном.
- **Жетконверсатионмемберс.** Позволяет получить список участников беседы и сохранить их в свойство в [памяти](#).
- **Едитактионс.** Позволяет изменять в режиме реального времени текущую последовательность действий на основе введенных пользователем данных. Особенно полезно при обработке [прерываний](#).

Подробные сведения и примеры см. в разделе " [Управление диалогами](#)" статьи "**действия в адаптивных диалоговых окнах**" — [Справочное руководство](#).

#### **Управление свойствами**

Действия управления свойствами позволяют создавать, обновлять и удалять свойства. Дополнительные сведения о свойствах см. в статьях [Управление состоянием](#) и управление состоянием пакета SDK Bot Framework [в адаптивных диалоговых окнах](#).

- **Изменение массива.** Это действие позволяет выполнять с массивом операции редактирования.
- **Удаление свойства.** Позволяет удалить свойство из [памяти](#).
- **Удаление свойств.** Позволяет удалять одним действием сразу несколько свойств.
- **Создание или обновление свойства.** Позволяет задать значение для свойства в [памяти](#).
- **Создание или обновление свойств.** Позволяет инициализировать одним действием одно или несколько свойств.

Подробные сведения и примеры см. в разделе [Управление свойствами](#) статьи "**действия в адаптивных диалоговых окнах**" — [справочное руководство](#)".

#### **Доступ к внешним ресурсам**

Эти действия позволяют получить доступ к внешним ресурсам, например навыкам, отправке HTTP-запроса, выдаче пользовательского события или вызову пользовательского кода и т. д.

- **Начало диалогового окна навыка.** Используйте адаптивный диалог навык для выполнения навыков.
- **Отправка HTTP-запроса.** Позволяет отправить HTTP-запросы к любой конечной точке.
- **Выдача пользовательского события.** Позволяет выполнить пользовательское событие, на которое бот будет реагировать через механизм [пользовательского триггера](#).
- **Выйдите из учетной записи пользователя.** Позволяет текущему пользователю выйти из системы.
- **Вызов пользовательского кода.** Позволяет вызвать произвольный пользовательский код.

Подробные сведения и примеры см. в разделе [доступ к внешним ресурсам](#) статьи Справочник по **действиям в адаптивных диалоговых окнах**.

#### **Параметры отладки**

- **Выполните вход в консоль.** Записывает сообщение в консоль и (необязательно) отправляет его как действие трассировки.
- **Выдает событие трассировки.** Используется для отправки действия трассировки с заданными данными в виде TE.

Подробные сведения и примеры см. в разделе [параметры отладки](#) раздела **действия в адаптивных диалоговых окнах** — [справочное руководство](#).

## **Дополнительные сведения**

- Сведения о действиях, связанных со сбором данных от пользователя, см. в статье [Запрос ввода данных пользователем с помощью адаптивных диалогов](#).
- Дополнительные сведения об адаптивных выражениях см. [в этой статье](#).
- Подробные сведения и примеры по всем действиям, описанным в этой статье, см. в статье [действия в адаптивных диалоговых окнах — справочник](#).

# Запрос ввода данных пользователем с помощью адаптивных диалогов

27.03.2021 • 3 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В пакете SDK для Bot Framework определены разнообразные диалоги ввода для сбора и проверки входных данных, вводимых пользователем. Диалоговые окна ввода — это тип адаптивного диалогового действия.

## IMPORTANT

Действия (*действия триггера*) — это диалоги, поэтому они предоставляют мощные и гибкие возможности для создания полнофункционального и надежного потока общения. В пакет SDK для Bot Framework уже включен обширный набор действий, но вы можете дополнить их собственными настраиваемыми действиями, что позволит выполнять практически любые специализированные задачи и процессы.

## Предварительные требования

- [Общие сведения об адаптивных диалогах](#)
- [События и триггеры в адаптивных диалогах](#)
- [Действия в адаптивных диалогах](#)
- [Области памяти и управление состоянием в адаптивных диалогах](#)
- Знакомство с [шаблонами создания языка](#)
- Знакомство с [адаптивными выражениями](#)

## TIP

Этот синтаксис определяется в шаблонах [создания речи](#), которые включают в себя [адаптивные выражения](#). Он используется в объекте `ActivityTemplate`, который необходим для нескольких параметров, используемых в большинстве входных действий, предоставленных в пакете SDK для Bot Framework.

## Входные данные

Как и [запросы](#), [входные данные](#) можно использовать в адаптивных диалогах, чтобы запрашивать и получать входные данные пользователя, проверять их и принимать в память. Входные данные:

- Привязывают результат запроса к свойству в области [управления состоянием](#).
- Запрашиваются у пользователя, только если у свойства Result еще нет значения.
- Сохраняются в указанное свойство, если они соответствуют ожидаемому типу сущности.
- Допускают ограничения проверки, такие как минимум, максимум и т. д.
- Позволяют использовать в качестве входных данных контекстно значимые намерения в диалоге, а также использовать прерывание для отображения всплывающего ответа пользователю для соответствующего родительского диалога, который может его обработать.

В библиотеке адаптивных диалогов определены следующие типы входных данных.

- [Входной базовый класс](#). Базовый класс, от которого наследуются все входные классы.

- **Text.** Для запроса на ввод данных, вводимых пользователем, введите **текст**. **Число.** Для запроса **\*числовых** вводимых пользователем данных \*. **\_Подтверждение.** Для запроса **подтверждения\*** от пользователя. **\_Множественный вариант.** Чтобы запросить выбор из **\*набора параметров**. **Файл или вложение.** Для запроса или предоставления пользователю возможности **отправки файла**.
- **Дата или время.** Чтобы запросить **\*дату и время** от пользователя. **\_Имя входа OAuth.** Чтобы пользователи могли входить на **защищенный сайт**.

## Дополнительные сведения

- Более подробные сведения о входных данных, включая примеры кода, см. в статье [адаптивные диалоговые окна с предварительно созданными входными данными](#).
- Чтобы узнать больше о выражениях, ознакомьтесь с разделом [Адаптивные выражения](#).

# Распознаватели в адаптивных диалогах

27.03.2021 • 17 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Распознаватели обеспечивают распознавание речи в адаптивных диалогах.

Распознаватели дают возможность понимать входные данные пользователя. В адаптивном диалоге можно настроить один или несколько распознавателей для преобразования входных данных пользователя в конструкции, которые может обработать бот. Например, можно настроить [распознавание речи и вопросы и ответы](#).

## Предварительные требования

- Общее представление о [принципе работы ботов](#) в пакете SDK для Bot Framework версии 4.
- Общее представление о принципах работы адаптивных диалогов в пакете SDK для Bot Framework версии 4. Дополнительные сведения см. в разделах [Общие сведения об адаптивных диалогах](#) и [Библиотеки диалогов](#).

## Общие сведения об обработке естественного языка в адаптивных диалогах

Распознаватели, как и [генераторы](#), обеспечивают возможности обработки естественного языка (NLP) в адаптивных диалогах. Обработка естественного языка — это технический процесс, который позволяет компьютерным приложениям, например ботам, извлекать смысл входных данных пользователем. Для этого предпринимается попытка определить ценные сведения, содержащиеся в беседах, путем интерпретации потребностей пользователей (намерений) и извлечения ценных сведений (сущностей) из предложения. После чего формируется ответ на языке, который будет понятен пользователю.

Возможности бота будут весьма скучны без обработки естественного языка. Она позволяет боту понимать сообщения пользователя и реагировать соответствующим образом. Когда пользователь отправляет сообщение с помощью "Hello", это возможность обработки естественного языка Bot, которая позволяет ему понять, что пользователь разместил стандартное приветствие, который, в свою очередь, позволяет использовать возможности AI для получения надлежащего ответа. В этом случае бот может ответить приветствием.

Без NLP вы не сможете различать роботов, когда пользователь вводит "Hello" или "прощание" или что-либо еще. Для программы-робота без NLP слова «Привет» и «прощание» не будут отличаться от других строк, объединенных в случайном порядке. Обработка естественного языка помогает получить контекст и смысл введенного текстового или голосового сообщения, чтобы бот мог выбрать наилучший ответ.

Одна из самых значительных проблем обработки естественного языка в боте заключается в том, что пользователи совершенно не имеют представления о том, что можно сказать боту. Хотя можно попытаться предсказать, что пользователи будут говорить, а что не будут, всегда возможны непредвиденные беседы. К счастью, пакет SDK для Bot Framework предоставляет инструменты, необходимые для постоянного улучшения возможностей обработки естественного языка бота.

Два основных компонента обработки естественного языка в адаптивных диалогах — это **распознаватели** (распознавание речи), которые обрабатывают и преобразовывают [входные данные пользователя](#) (именно им посвящена данная статья), и [генераторы](#) (создание речи), которые формируют [ответы бота](#), то есть создают осмысленные фразы и предложения на естественном языке.

Попросту говоря, бот отвечает пользователю на понятном для человека языке.

#### TIP

Хотя пользователи часто общаются с ботов, вводя или произнося сообщение, распознаватель — это подсистема, которую можно использовать для обработки любого вида входных данных пользователя. Можно распознавать речь, текст, щелчки (например, при ответе в [адаптивных картах](#)) и даже другие модальности, например, можно использовать распознаватель геолокации или распознаватель взгляда. Уровень распознавателя позволяет абстрагировать сложность обработки входных данных пользователя от триггеров и действий. Таким образом триггерам и действиям не нужно интерпретировать различные типы входных данных пользователя, вместо них этим занимаются распознаватели.

## Распознавание речи

**Распознавание речи** (LU) — это подраздел обработки естественного языка, который посвящен обработке входных данных пользователя и их преобразованию в то, что бот сможет понять и на что он сможет интеллектуально отреагировать. Этот процесс реализуется путем настройки распознавателей и предоставления обучающих данных для диалога, чтобы можно было получать **намерения и сущности**, содержащиеся в сообщении пользователя. Когда распознаватель распознает намерение, он выдает событие `IntentRecognized`, содержащее это намерение. Затем срабатывает триггер `OnIntent`, определенный для этого намерения, и выполняются действия, содержащиеся в данном триггере.

## Основные принципы распознавания речи в адаптивных диалогах

### Намерения

Намерения позволяют классифицировать ожидаемые намерения пользователя, выражаемые в сообщениях для бота. Намерение можно считать представлением действия, которое пользователь хочет выполнить, то есть это цель, выраженная в его входных данных. Это может быть бронирование авиабилетов, оплата счета или поиск новостной статьи. Вы определяете и именуете намерения, соответствующие этим действиям. Например, любой бот может определить намерение *Greeting* (Приветствие), а приложение для путешествий может создать намерение *BookFlight* (Бронирование авиабилетов). Намерения определяются в файле шаблона распознавания речи. Это текстовые файлы с расширением "LU", которые обычно находятся в одном каталоге и имеют то же имя, что и диалог. Например, корневой диалог будет содержать файл шаблона распознавания речи `RootDialog.lu`.

Ниже приведен пример LU-файла, который понимает простое намерение *Greeting* и использует список примеров речевых фрагментов, соответствующих разным способам, которыми пользователь может выразить это намерение. Используйте знак `-`, `+` или `*` для обозначения списков. Нумерованные списки не поддерживаются.

```
# Greeting
- Hi
- Hello
- How are you?
```

`#<intent-name>` описывает новый раздел определения намерений в LU-файле шаблона. Каждая строка после определения намерения является примером речевого фрагмента, описывающего это намерение. В LU-файле можно создать несколько определений намерений. Каждый раздел обозначается нотацией `#<intent-name>`. При анализе файла пустые строки пропускаются.

### Высказывания

Речевые фрагменты (*фразы-триггеры*) — это входные данные, вводимые пользователями. Поэтому они могут содержать почти бесконечное число возможных вариантов. Так как речевые фрагменты не всегда

сформированы правильно, необходимо предоставить несколько примеров речевых фрагментов для конкретных намерений, которые фактически позволят обучить бот распознавать намерения из разных речевых фрагментов. Таким образом бот сможет более "интеллектуально" понимать человеческую речь.

#### TIP

Речевые фрагменты также называют *фразами-триггерами*, так как это *фразы, выраженные пользователем*, который могут активировать *триггер* `OnIntent`.

## Сущности

Сущности — это коллекция объектов, каждый из которых состоит из данных, извлеченных из речевого фрагмента, которые добавляют дополнительные сведения, описывающие намерение. Например, это могут быть места, время и люди. Сущности и намерения — это важные фрагменты данных, извлекаемые из речевого фрагмента. Речевые фрагменты, как правило, содержат намерение и могут включать в себя сущности, предоставляющие важные сведения, связанные с этим намерением.

Сущности в формате LUIS-файла определяются в следующем формате: `{<entityName>=<labelled value>}`, например, `{toCity=seattle}` (EntityName `toCity`, а значение с меткой — `seattle`). Пример:

```
# BookFlight
- book a flight to {toCity=seattle}
- book a flight from {fromCity=new york} to {toCity=seattle}
```

В приведенном выше примере показано определение намерения `BookFlight` с двумя примерами речевых фрагментов и двумя определениями сущностей: `toCity` и `fromCity`. Если при активации распознаватель сможет определить город назначения, то название города будет доступно как `@toCity` в активированных действиях или город отбытия будет доступен в `@fromCity` как значения сущностей. Значения сущностей можно использовать непосредственно в выражениях и шаблонах создания речи или сохранить в свойстве в [памяти](#) для последующего использования.

## Типы распознавателей

Пакет SDK для Bot Framework предоставляет несколько различных распознавателей, включая возможность создавать собственные.

Адаптивные диалоги поддерживают следующие распознаватели:

- [распознаватель RegexRecognizer](#);
- [Распознаватель для LUIS](#)
- [распознаватель QnA Maker](#);
- [многоязычный распознаватель](#);
- [набор взаимно-обучаемых распознавателей](#);
- [RecognizerSet](#).

### RegexRecognizer

`RegexRecognizer` дает возможность извлекать данные намерений и сущностей из речевого фрагмента на основе шаблонов регулярных выражений.

`RegexRecognizer` состоит в основном из:

- `Intents`. Объект `Intents` содержит список объектов `IntentPattern`, а эти объекты `IntentPattern` содержат свойство `Intent`, которое является именем намерения, и свойство `Pattern`, содержащее регулярное выражение, используемое для анализа речевого фрагмента для распознания намерения.

- `Entities`. Каждый объект `Entities` содержит список объектов `EntityRecognizer`. Пакет SDK для Bot Framework определяет несколько `EntityRecognizer` классов, которые помогут определить сущности, содержащиеся в utteranceх пользователей.

Подробные сведения и примеры см. в разделе [режексрекогнайзер](#) в справочном руководстве по использованию распознавателей в адаптивных диалоговых окнах.

### Распознаватель LUIS

Интеллектуальная служба распознавания речи (LUIS) — это облачная служба API, которая применяет пользовательскую аналитику машинного обучения к тексту пользователя в разговорном стиле и на естественном языке, чтобы предсказать общий смысл и извлечь соответствующую подробную информацию. Распознаватель LUIS позволяет извлекать намерения и сущности из речевых фрагментов пользователей с помощью определенного приложения LUIS, которое было предварительно обучено.

Подробные сведения и пример создания распознавателя LUIS см. в разделе [распознаватель Luis](#) в справочном руководстве распознаватели в адаптивных диалоговых окнах.

Подробные инструкции по созданию приложения LUIS и развертыванию моделей LUIS с помощью интерфейса командной строки Bot см. в разделе [развертывание ресурсов Luis с помощью команд CLI для пакета SDK для Bot Framework](#).

### Распознаватель QnA Maker

[QnAMaker.ai](#) — это одна из служб [Microsoft Cognitive Services](#), которая позволяет создавать пары "вопрос-ответ" на основе существующего содержимого — документов, URL-адресов, документов в формате PDF и т. д. Для интеграции с этой службой можно использовать распознаватель QnA Maker.

Подробные сведения и пример создания распознавателя QnA Maker см. в разделе [распознаватель QnA Maker](#) в справочном руководстве распознаватели в адаптивных диалоговых окнах.

### Многоязычный распознаватель

При создании сложного многоязычного бота, как правило, один распознаватель привязывается к конкретному языку и языковому стандарту. Многоязычный распознаватель позволяет легко указать распознаватель для использования на основе свойства `locale` входящего действия пользователя.

Подробные сведения и пример создания многоязыкового распознавателя см. в разделе [многоязыковой распознаватель](#) в справочном руководстве распознаватели в адаптивных диалоговых окнах.

### Набор распознавателей

Иногда может потребоваться запускать несколько распознавателей на каждом этапе беседы. Именно для этого предназначен набор распознавателей. Все распознаватели выполняются на каждом этапе диалога, а результат представляет собой объединение всех результатов распознавания.

Подробные сведения и пример создания набора распознавателя см. в разделе " [набор распознавателей](#) " в справочном руководстве распознаватели в адаптивных диалоговых окнах.

### Набор взаимно-обучаемых распознавателей

Набор расученных распознавателей сравнивает результаты распознавания от нескольких распознавателей, чтобы принять решение о победителях. Для заданной коллекции распознавателей взаимно-обучаемый распознаватель сделает следующее.

- Повысит приоритет результата распознавания одного из распознавателей, если все другие распознаватели переносят распознавание на отдельный распознаватель. Чтобы перенести распознавание, распознаватель может вернуть намерение `None` или явное намерение `DeferToRecognizer_recognizerId`.
- Породит событие `onChooseIntent`, чтобы позволить коду выбрать, какой результат распознавания следует использовать. Результаты каждого распознавателя возвращаются с помощью свойства

`turn.recognized.candidates`. Это позволяет выбрать наиболее подходящий результат.

#### Дополнительные сведения о перекрестном обучении

- Подробные сведения о перекрестном обучении программы Bot см. в разделе [перекрестное обучение программы-робота для использования Luis и QnA Maker распознаватели](#).
- Технические сведения о перекрестном обучении программы-робота см. в разделе [перекрестно обученный набор распознавателей](#) в окне справочника по адаптивным диалоговым окнам.
- Дополнительные сведения о перекрестном обучении программы Bot с помощью существующего робота в качестве примера см. в разделе [Создание перекрестной ОБЛUIСки для использования как распознавателя, так и QnA Maker](#).

## Дополнительные сведения

- [Что такое LUIS?](#)
- [Распознавание речи](#)
- [.lu File Format](#) (Формат файлов LU)
- [Адаптивные выражения](#)
- [Извлечение данных из текста речевого фрагмента с помощью намерений и сущностей](#)
- [Добавление возможности распознавания естественного языка в бот](#)
- [Добавление возможности создания естественного языка в бот](#)
- Дополнительные сведения о распознавателях в адаптивных диалоговых окнах, включая примеры, см. в разделе «[распознаватели в адаптивных диалоговых окнах — справочное руководство](#)».

# Генераторы в адаптивных диалогах

27.03.2021 • 6 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Генератор привязывает определенную систему [создания речи](#) (LG) к адаптивному диалогу. В этой статье вы узнаете о шаблонах LG, которые добавляют разнообразные и индивидуальные данные в ответы Bot, а также о том, как вызывать эти шаблоны в корневом диалоговом окне с помощью `TemplateEngineLanguageGenerator`.

## Предварительные требования

- Знания об [адаптивных диалогах](#).
- Представление о роли [распознавателей](#) в адаптивных диалогах.
- Знания об [адаптивных выражениях](#).

## Создание речи

Создание языка (LG) позволяет разработчикам извлекать внедренные строки из файлов кода и ресурсов и управлять ими с помощью среды выполнения LG и формата файлов. С помощью LG разработчики могут создавать более естественные диалоги, определяя несколько вариантов фраз, выполняя простые выражения на основе контекста и ссылаясь на сеансы памяти.

Наряду с [распознавателями](#), создание речи позволяет четко отделить и инкапсулировать ресурсы распознавания речи и создания речи диалога. Распознаватели дают боту возможность понять входные данные, а создание речи позволяет боту интеллектуально реагировать на них.

## Шаблоны создания речи

Шаблоны можно считать основным понятием в системе создания речи. Варианты, заданные в шаблонах, используются ботом для реагирования на вводимые пользователем данные. Каждый шаблон имеет имя и содержит что-то одно из следующего:

- список текстовых значений для однократных вариантов;
- определение структурированного содержимого;
- коллекция условий, в каждом из которых есть:
  - [адаптивное выражение](#);
  - список текстовых значений для однократных вариантов по условиям.

Шаблоны определяются в LG-файлах, которые могут содержать один или несколько шаблонов.

Существуют три типа шаблонов:

- [простой ответ](#);
- [условный ответ](#);
- [структурированный ответ](#).

### Шаблон простого ответа

Шаблоны простого ответа могут содержать один или несколько вариантов текста, используемых для составления и расширения ответов. При вычислении этого шаблона среда выполнения создания речи случайным образом выбирает один из этих вариантов.

Ниже приведен пример шаблона простого ответа с двумя вариантами.

```
> Greeting template with 2 variations.  
# GreetingPrefix  
- Hi  
- Hello
```

## Шаблон условного ответа

Шаблоны условного ответа позволяют создавать содержимое на основе условия. Все условия выражаются с помощью [адаптивных выражений](#), как предварительно [созданных](#), так и [пользовательских](#).

Существуют два типа шаблонов условного ответа: [IF-ELSE](#) и [SWITCH](#).

### Шаблон IF-ELSE

Шаблон IF-ELSE позволяет создать шаблон, который выбирает коллекцию на основе каскадной структуры условий. Как и шаблоны простого ответа, этот шаблон может содержать внутренние варианты в блоках IF или ELSE. Вложенность реализуется с помощью [композиции](#).

Ниже приведен пример шаблона условного ответа IF-ELSE.

```
> time of day greeting reply template with conditions.  
# timeOfDayGreeting  
- IF: ${timeOfDay == 'morning'}  
  - Good morning!  
  - Wake up.  
- ELSE:  
  - Good evening  
  - Sleep well!  
  - Goodnight.
```

### Шаблон SWITCH

Шаблон SWITCH позволяет создать шаблон, который сравнивает значение некоторого выражения с предложениями CASE и формирует результат по правилам для этого варианта.

Ниже приведен пример шаблона условного ответа SWITCH.

```
# TestTemplate  
- SWITCH: ${condition}  
- CASE: ${case-expression-1}  
  - output1  
- CASE: ${case-expression-2}  
  - output2  
- DEFAULT:  
  - final output
```

## Шаблон структурированного ответа

Шаблон структурированного ответа позволяет определить сложную структуру с поддержкой большого набора функций из системы создания речи, таких как использование шаблонов, компоновка и подстановка, передавая задачу составления структурированного ответа объекту, вызывающему библиотеку создания речи. Определения действий и карт в настоящее время поддерживаются для приложений ботов.

Ниже приведено описание шаблона структурированного ответа.

```
# TemplateName
> this is a comment
[Structure-name
  Property1 = <plain text> .or. <plain text with template reference> .or. <expression>
  Property2 = list of values are denoted via '|'. e.g. a | b
> this is a comment about this specific property
  Property3 = Nested structures are achieved through composition
]
```

Дополнительные сведения и примеры сложных шаблонов доступны в разделе [Шаблон структурированного ответа](#).

## Вызов шаблонов в корневом диалоге

После создания шаблонов для бота их можно добавить в адаптивный диалог. Можно задать для генератора LG-файл или экземпляра `TemplateEngineLanguageGenerator`, в котором вы явно управляете одним или несколькими LG-файлами. В приведенном ниже примере показано применение второго подхода.

Предположим, что требуется добавить шаблоны из `RootDialog.lg` в адаптивный диалог. Добавьте в код приведенные ниже пакеты.

```
using Microsoft.Bot.Builder.LanguageGeneration;
using Microsoft.Bot.Builder.Dialogs.Adaptive.Templates;
using Microsoft.Bot.Builder.Dialogs.Adaptive.Generators;
```

Затем необходимо разрешить путь к LG-файлу.

```
string[] paths = { ".", "Dialogs", "RootDialog.lg" };
string fullPath = Path.Combine(paths);
```

Это обеспечит вызов правильных файлов шаблонов для программы-робота.

Теперь можно создать `TemplateEngineLanguageGenerator` в адаптивном диалоге, который управляет шаблонами в `RootDialog.lg`.

```
var rootDialog = new AdaptiveDialog(nameof(AdaptiveDialog)){
  .
  .
  .
  Generator = new TemplateEngineLanguageGenerator(Templates.ParseFile(fullPath))
}
```

Теперь можно вызывать шаблоны в боте по имени, используя формат  `${<template-name>}` .

```
new SendActivity("${FinalUserProfileReadOut()}")
```

В приведенном выше примере бот вызывает шаблон `FinalUserProfileReadOut` и реагирует с помощью его содержимого.

## Дополнительные сведения

- [Формат файлов LG](#)
- [Шаблон структурированного ответа](#)

- Встроенные функции адаптивных выражений

# Управление состоянием в адаптивных диалогах.

27.10.2020 • 6 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Понятия *с отслеживанием состояния* и *без отслеживания состояния* описывают, предназначено ли приложение для запоминания одного или нескольких предшествующих событий в заданной последовательности взаимодействия с пользователем (или любым другим действием). С отслеживанием состояния приложение отслеживает состояние его взаимодействия, обычно путем сохранения значений в памяти в виде свойства. "Без отслеживания состояния" означает, что приложение не отслеживает состояние своих взаимодействий. Это означает, что память каких-либо предыдущих взаимодействий отсутствует и все входящие запросы должны содержать всю информацию, необходимую для выполнения запрошенного действия. Можно представить *состояние* как текущий набор значений или содержимое бота, например идентификатор беседы или имя активного пользователя.

Пакет SDK для Bot Framework следует тем же принципам, что и современные веб-приложения, и не управляет состоянием. Однако пакет SDK для Bot Framework предоставляет некоторые абстракции, которые значительно упрощают внедрение управления состоянием в бот. Эта тема подробно описана в статье об [управлении состоянием](#) в пакете SDK для Bot Framework. Рекомендуется прочитать и изучить сведения, изложенные в данной статье, прежде чем читать текущую статью.

## Предварительные требования

- Общее представление о том, [как работают боты](#).
- Общее представление об адаптивных диалогах. Дополнительные сведения см. в разделах [Общие сведения об адаптивных диалогах](#) и [Библиотеки диалогов](#).
- Общие сведения об управлении состоянием см. в статье об [управлении состоянием](#) в пакете SDK для Bot Framework.

## Управление состоянием

Бот по своей природе не учитывает состояния. Для некоторых ботов это удобно — либо они могут работать без дополнительных сведений, либо все нужные сведения гарантированно присутствуют во входящем действии. Другим ботам сведения о состоянии (например, текущий этап беседы или полученный ответ пользователе) необходимы, чтобы поддерживать полезную беседу.

Пакет SDK для Bot Framework определяет области памяти, в которых разработчики могут сохранять значения из памяти бота и затем извлекать их. Эти значения могут использоваться при обработке циклов и ветвей, а также при создании динамических сообщений и других процессов в боте.

Это дает ботам, созданным с помощью пакета SDK для Bot Framework, возможность:

- передавать сведения между диалогами;
- сохранять профили и предпочтения пользователей;
- запоминать важную информацию между сессиями, например последний поисковый запрос или последний выбор, сделанный пользователем.

Адаптивные диалоги позволяют обращаться к памяти и управлять ею. Эту модель используют все адаптивные диалоги, то есть все компоненты, считывающие или записывающие данные в память, применяют согласованный способ обработки информации в соответствующих областях.

#### NOTE

Все свойства памяти, хранящиеся во всех областях памяти, являются контейнерами свойств, то есть при необходимости в них можно добавлять свойства.

Ниже приведены различные области памяти, доступные в адаптивных диалоговых окнах, в каждом из которых содержится ссылка на дополнительные сведения, содержащиеся в справочной статье о состояниях памяти:

- **Область пользователя** — это постоянные данные, ОГРАНИЧЕННЫЕ идентификатором пользователя, которым выполняется обращение.
- **Область диалога** — это сохраняемые данные, областью действия которых является идентификатор диалога.
- **Область диалогового окна** сохраняет данные в течение жизненного цикла связанного диалогового окна, предоставляя пространство памяти для каждого диалога для внутреннего постоянного бухгалтерского учета.
- **Включение области** предоставляет место для совместного использования данных в течение времени существования текущей очереди.
- **Область параметров** представляет все параметры, которые становятся доступными для программы-робота через систему конфигурации параметров, характерных для платформы.
- **Эта область** сохраняет данные за время существования связанного действия. Ее удобно использовать для действий ввода, так как их жизненный цикл обычно превышает один этап беседы.
- **Область класса** содержит свойства экземпляра активного диалогового окна.

## Сокращенные нотации памяти

Для доступа к конкретным областям памяти существует несколько сокращенных нотаций.

СИМВОЛ	ИСПОЛЬЗОВАНИЕ	РАСШИРЕНИЕ	ПРИМЕЧАНИЯ
\$	\$userName	dialog.userName	Сокращенная нотация, представляющая область диалога.
#	#intentName	turn.recognized.intents.intentName	Сокращение, используемое для обозначения именованного намерения, возвращенного распознавателем.
@	@entityName	turn.recognized.entities.entityName	возвращает первое и только первое значение, найденное для сущности, вне зависимости от кратности значения.
@@	@@entityName	turn.recognized.entities.entitiesName	вернет фактическое значение сущности с учетом кратности.

СИМВОЛ	ИСПОЛЬЗОВАНИЕ	РАСШИРЕНИЕ	ПРИМЕЧАНИЯ
%	<code>%propertyName</code>	<code>class.propertyName</code>	Используется для ссылки на свойства экземпляра (например, <code>MaxTurnCount</code> , <code>DefaultValue</code> и т. д.).

## Дополнительные сведения

- Более подробные сведения об управлении состоянием, включая примеры, см. в статье [Управление состоянием в адаптивных диалоговых окнах — справочное руководство](#).

# Управление прерываниями в адаптивных диалогах

27.03.2021 • 19 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

При создании потока диалогового окна, запрашивающего ввод данных пользователем, необходимо решить, что должно произойти, когда пользователь отвечает, который принимает диалог в другом направлении. Это естественно, если речь идет о том, чтобы временно обсудить другую тему перед возвратом к первоначальной теме, а также возможность беспрепятственно работать с сценариями, в которых пользователь прерывает поток беседы, является важным аспектом надежного робота.

## IMPORTANT

Обработка прерываний в адаптивных диалоговых окнах становится возможной с помощью механизма "консультации" пакета SDK для Bot Framework, который позволяет диалоговому окну обращаться к его родительскому диалоговому окну. Этот механизм консультации также обеспечивает [гибкое извлечение сущностей](#) и [подтверждение и исправление](#), описанные далее в этой статье.

## Предварительные требования

- [Общие сведения об адаптивных диалогах](#)
- [Запрос ввода данных пользователем с помощью адаптивных диалогов](#)
- Базовое понимание концепций [понимания языка](#), особенно концепций [цели и сущности](#).

## Создание потока диалога

Предположим, вы создаете робот для выполнения заказов в местном кафе. Вы можете создать поток сообщений, который выглядит примерно так:

**Пользователь:** я хочу заказать кофе.

**Bot:** какой тип кофе вам хотелось?

**Пользователь:** эспрессо.

**Bot:** какой размер вам нравится?

**Пользователь:** сделайте его большим..

**Bot.** нет проблем, большая эспрессо будет готова через 5 минут.

Что произойдет, если вы запросили тип кофе, вы не получаете ответ, а на вопрос? Это один из способов, который пользователь может прервать в потоке. Рассмотрим следующий пример:

**Пользователь:** я хочу заказать кофе.

**Bot:** какой тип кофе вам хотелось?

**Пользователь:** какие типы кофе у вас есть?

Если вы не ожидали прерывания работы пользователей, предполагая, что основной уровень обработки ошибок, диалог может выглядеть примерно так:

**Пользователь:** я хочу заказать кофе.

Bot: какой тип кофе вам хотелось?

**Пользователь:** какие типы кофе у вас есть?

Bot: я не понял вашего ответа.

Bot: какой тип кофе вам хотелось?

Адаптивные диалоговые окна дают ответ на этот дилеммой, используя *прерывания*. Перерывы — это метод, доступный в адаптивном диалоговом окне, позволяющий роботу понять и реагировать на ввод пользователя, который не относится непосредственно к конкретному фрагменту информации, для которой пользователь помещает запрос.

### Примеры прерываний в потоке диалога

В отличие от формы с заранее заданным набором полей, которые пользователь заполнит, Bot позволяет пользователю вводить данные в поле ввода, и надежному роботу придется понять, как правильно реагировать на различные входные данные пользователя. Прерывания — это одно средство, которое позволяет программе-роботу реагировать на неожиданные входные данные пользователя, или ввод, который не помещается непосредственно в диалоговом потоке данного диалогового окна. Ниже перечислены сценарии, которые могут оказаться полезными при обработке прерываний.

- Программа-робот, которая выполняет вылетающий бюллетень.
  - Запрашивается прогноз температуры для назначения в процессе резервирования рейса.
  - Пользователь, запрашивающий Добавление аренды автомобиля в окончательное подтверждение подтверждения на перелет.
- Программа-робот, которая запрашивает время открытия магазина в середине заказа кофе.
- Запрашивается медицинское или юридическое определение, а затем возвращается для выполнения серии вопросов.
- Запрос на *справку* в середине любого диалога.
- Запрос на *отмену* в середине любого диалога.
- Запуск процесса с начала, когда в середине набора шагов.

## Обработка прерываний

Перерывы могут обрабатываться локально в диалоговом окне, а также глобально путем повторной маршрутизации в другое диалоговое окно. Перерывы можно использовать в качестве методики:

- Обнаружение и обработка ответа пользователя в качестве локально соответствующей цели в области активного диалогового окна. Это означает, что Адаптивное диалоговое окно, содержащее триггер, содержащий действие ввода, предлагающее пользователю, является диалоговым окном, которое обрабатывает *локальные* прерывания.
- Определение того, что другое диалоговое окно лучше подходит для работы с вводом данных пользователя, а затем с помощью механизма консультаций по адаптивному диалогу, чтобы позволить другому диалоговому окну обрабатывать вводимые пользователем данные.

Если входное действие активно, при получении ответа от пользователя адаптивные диалоговые окна выполняют следующие действия.

- Запустите распознаватель, настроенный в адаптивном диалоговом окне, содержащем Входное

действие.

- Оцените значение свойства *Разрешить прерывания*.
  - При **значении true**: вычислять триггеры в адаптивном диалоговом окне, содержащем Входное действие. Если триггеры не срабатывают, вызывается родительское Адаптивное диалоговое окно и выполняется его распознаватель, а затем оцениваются его триггеры. Если триггеры не срабатывают, этот процесс будет продолжен до тех пор, пока не будет достигнуто корневое диалоговое окно.
  - При **значении false**: оцените свойство *value* и присвойте его значение свойству, привязанному к входным данным. Если значение равно null, запустите распознаватель внутренних сущностей для этого входного действия, чтобы разрешить значение для этого действия ввода. Если внутренний распознаватель был возвращен без результатов, выдайте запрос повторно.

### Свойство "разрешить прерывания"

*Разрешить прерывания* — это свойство класса *диалогового окна входных данных*, для которого все входные данные являются производными от, поэтому они доступны при **запросе ввода пользователя в адаптивных диалоговых окнах**. Свойство *Разрешить перерывы* принимает **логическое выражение**, которое может быть либо логическим значением (*true/false*), либо **адаптивным выражением**, которое разрешается в логическое значение.

Свойство *Разрешить прерывания* определяется при создании **входных данных** и значения по умолчанию *true*. Некоторые примеры.

РАЗРЕШИТЬ ЗНАЧЕНИЕ ПРЕРЫВАНИЯ	ОБЪЯСНЕНИЕ
True	Разрешить прерывания во входных данных.
"false"	Не разрешать прерывания во входных данных.
"поверните. Score <= 0,7"	Разрешите прерывания только в том случае, если у нас нет классификации с высокой достоверностью.
"Turn. распознаваемое. Score >= 0,9    ! @personName "	Разрешите прерывания только при наличии классификации с высокой достоверностью или если вы не получаете значение для <code>personName</code> сущности.

#### TIP

`personName` — Это предварительно созданная сущность, предоставляемая *Luis*, которая обнаруживает имена людей. Это очень полезно, так как оно уже обучено, поэтому вам не нужно добавлять пример фразы продолжительностью, содержащийся `personName` в программы-роботы. Дополнительные сведения об использовании готовых сущностей в файле Bot см. в [формате Lu](#).

### Локальная обработка прерываний

Прерывания можно выполнять локально, добавляя триггеры в соответствии с возможными способами. Можно добавить любой триггер, который подписывается на `recognition` событие, например `OnIntent` или `OnQnAMatch`. Рассмотрим следующий пример.

**Пользователь:** я хочу заказать кофе.

**Bot:** какой тип кофе вам хотелось?

**Пользователь:** эспрессо.

**Bot:** какой размер вам нравится?

**Пользователь:** какие размеры у вас есть?

Bot. Мы предлагаем следующие размеры эспрессо:

Высокий \$2,95  
Гранди \$3,65  
Венти \$4,15

Bot: какой размер вам нравится?

**Пользователь:** Гранди.

Bot. нет проблем, Гранди эспрессо будет готов за 5 минут.

В этом примере Bot запускается в корневом диалоговом окне. Когда пользователь запрашивает порядок кофе, распознаватель корневого диалогового окна вернул `order` намерение, что привело бы к запуску диалогового окна `заказа` для запуска и обработки заказа. Когда пользователь прервал поток сообщений о заказе с вопросом, распознаватель диалогового окна `заказа` вернул `sizes` намерение, которое было обработано локально, то есть в диалоговом окне `заказа`.

Что происходит в тех случаях, когда активному диалоговому окну не удается обнаружить намерение из ответа пользователей? Механизм консультации предлагает решение, позволяя диалоговому окну программы-робота отправить ответ на активный родительский диалог диалогов, который обсуждается далее.

### Глобальное управление прерываниями

*Глобальные прерывания* — это прерывания, которые не обрабатываются активным адаптивным диалоговым окном. Если в активном адаптивном диалоговом окне нет триггеров, способных справиться с намерением, который включает в себя любой триггер, который подписывается на событие `распознавания`, например `OnIntent` `OnQnAMatch` триггеры или, программа-робот отправляет его в родительское диалоговое окно, используя механизм *консультации* адаптивных диалоговых окон. Если в родительском диалоговом окне нет триггера, обрабатывающего намерение, он переходит в состояние, пока не достигнет корневого диалогового окна.

Типичные варианты использования глобальных прерываний включают создание базовых функций управления диалогами, таких как *Справка* или *Отмена* в корневом диалоговом окне, которые затем доступны для любого из его дочерних диалоговых окон.

После обработки прерываний поток диалога продолжается там, где он вышел, с двумя возможными исключениями. Первое исключение возникает при использовании действия `едитактионс` для изменения последовательности действий. Второе исключение возникает, когда прерывание является запросом отмены. в этом случае можно использовать действие `канцелалдиалог` для завершения потоковой передачи и активного адаптивного диалогового окна, как показано в следующем примере:

Bot: хорошее утро, как я могу помочь вам?

Bot запускается в *корневом* диалоговом окне, дружелюбны пользователю.

**Пользователь:** я хочу заказать кофе.

Пользователь utterance *хочу заказать кофе*. отправляется распознавателю LUIS, который возвращает `order` намерение, а также `coffee` сущность. Триггер, связанный с `order` намерением, срабатывает и вызывает диалоговое окно `заказа` для обработки заказа.

Bot: какой тип кофе вам хотелось?

**Пользователь:** не забудьте отменить заказ.

**Bot:** нет проблем, попрекрасный день!

Это завершает поток диалога между Bot и пользователем. В этом случае `order` диалоговое окно закрывается и управление возвращается к корневому диалоговому окну.

## Извлечение гибких сущностей

При каждом запуске адаптивного диалогового окна все параметры, передаваемые в диалоговое окно через *диалоговое окно Begin*, становятся свойствами этого диалогового окна и доступны при условии, что оно находится в *области*. Доступ к этим свойствам можно получить по имени: `dialog.<propertyName>`.

### NOTE

Эта же концепция применима и к сущностям, доступным в *области действия "включить"*.

Вы можете спроектировать робота, чтобы использовать эти значения при их наличии, и запрашивать пользователя, когда они отсутствуют. Если диалоговое окно ввода может принимать входные данные из нескольких различных источников, можно использовать `coalesce` предварительно созданную функцию для использования первого значения, отличного от NULL. `coalesce` доступен в составе [адаптивных выражений](#).

Существуют ситуации, когда может потребоваться запросить у пользователя эти сведения, даже если свойство не имеет значение `null`. В таких случаях можно задать параметр *всегда запрашивать true*.

Другой способ извлечения гибких сущностей может быть очень полезен, когда пользователь отвечает на запрос информации и, помимо ответа на свой вопрос, также предоставляет дополнительные важные сведения. Например, при регистрации нового пользователя можно запросить имя пользователя и ответить на него, а также их возраст:

**Bot:** что такое имя?

**Пользователь:** мое имя — вишвак, а я — 36 лет назад

Или Bot-робот, запрашивающий о уходе пользователей в аэропорту, а также ответ пользователя с названием города отправления вместе с конечным городом.

**Bot:** что такое ваш город отправления?

**Пользователь:** мне нужно полет с Детройт на Сиэтл

Гибкое извлечение сущностей позволяет правильно обработать эти ситуации. Для этого необходимо определить цели с определенными фразами продолжительностью в файлах. Iu, которые предполагается ввести пользователю. в первом примере приведенный ниже шаблон определяет `getUserProfile` намерение со списком возможных фразы продолжительностью, которые присваивают введенные пользователем значения в `@firstName` `@userAge` переменные и.

```
# getUserProfile
- {userName=vishwac}, {userAge=36}
- I'm {userName=vishwac} and I'm {userAge=36} years old
- call me {userName=vishwac}, I'm {userAge=36}.
- my name is {userName=vishwac} and I'm {userAge=36}
- {userName=vishwac} is my name and I'm {userAge=36} years of age.
- you can call me {userName=vishwac}, I'm {userAge=36}
```

## Перекрестное обучение языковых моделей для обработки прерываний

Перекрестное обучение по моделям языка — это подход к полной обработке прерываний. При перекрестном обучении моделей LUIS в Bot каждое Адаптивное диалоговое окно может быть осведомлено о возможностях других диалоговых окон и переносить поток диалога в диалоговое окно, предназначенное для обработки любого конкретного запроса пользователя. Перекрестное обучение также может упростить использование нескольких распознавателей в адаптивном диалоговом окне, а также в нескольких диалоговых окнах, использующих различные технологии, например LUIS и QnA Maker, чтобы позволить роботу определить наилучшую технологию для реагирования на пользователя. Дополнительные сведения см. в статье [перекрестное обучение робота для использования распознавателей Luis и QnA Maker](#).

## Подтверждение и исправление

*Подтверждение и исправление* позволяет использовать сценарий, в котором вы запрашиваете у пользователя подтверждение, и он не только выдает подтверждение, но также включает входные данные, содержащие дополнительные пользовательские намерения в ответ на подтверждение.

## Дополнительные сведения

- Как [управлять прерываниями пользователей в адаптивных диалоговых окнах](#).
- [Перекрестное обучение программы-робота для использования распознавателей Luis и QnA Maker](#).
- [Адаптивные выражения](#).
- [Формат файлов LU](#)
- [Целей в приложении LUIS](#)
- [Выясните, какие фразы лучше подходят для вашего приложения LUIS](#)

# Использование декларативных ресурсов в адаптивных диалогах

27.03.2021 • 25 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье объясняются принципы работы декларативных ресурсов и ботов, в том числе адаптивные диалоги на основе декларативного подхода. Декларативные адаптивные диалоги состоят из файлов на основе JSON, описывающих все атрибуты адаптивных диалогов, включая их [триггеры](#) и [действия](#). Эти декларативные файлы загружаются во время выполнения с помощью диспетчера ресурсов для создания адаптивных диалогов.

## Предварительные требования

- Понимание принципов работы [ботов, библиотек диалогов и адаптивных диалогов](#).

## Декларативные файлы

В настоящее время декларативные файлы — это *DIALOG*-файлы, описывающие все атрибуты адаптивного диалога, и *LG*-файлы, которые состоят из шаблонов создания речи, определяющих аспекты [создания речи \(LG\)](#) бота.

### DIALOG-файлы

Декларативные файлы адаптивных диалогов с расширением DIALOG содержат следующие элементы.

- Значение `$schema` содержит универсальный код ресурса (URI), указывающий на схему, описывающую формат этого декларативного файла. Это схема компонента Bot Framework, которая соответствует [черновику 7](#) словаря схемы JSON. Этот файл схемы позволяет [IntelliSense](#) работать с декларативными элементами. Сведения о том, как создать этот файл, см. в разделе [команды merge](#) ниже. Имя файла схемы может быть любым допустимым именем, но обычно он называется `app.schema`.
- Поле `$kind` определяет тип компонента, описываемого в этом файле. Для адаптивного диалога значением `$kind` должно быть `Microsoft.AdaptiveDialog`. Во вспомогательных объектах `$kind` определяет триггер или действие, которое является частью диалогового окна. Это поле сопоставляется с атрибутом класса `[JsonProperty("$kind")]`, который связан с каждым классом в пакете SDK для Bot Framework, предназначенным для применения декларативного подхода.
- Значение `recognizer` содержит [тип распознавателя](#) и массив из одного или нескольких [намерений](#), а также, при необходимости, массив из одной или нескольких [сущностей](#).
- Значение `generator` содержит ссылку на LG-файл, связанный с адаптивным диалогом, который определяет данный DIALOG-файл.
- Значение `triggers` содержит массив из одного или нескольких [триггеров](#). Тип триггера объявляется с помощью ключевого слова `$kind`. Каждый триггер содержит массив из одного или нескольких действий.
- Значение `actions` содержит массив из одного или нескольких [действий](#). С каждым действием могут быть связаны свойства.

Пример простого DIALOG-файла приведен ниже.

```
{
    "$schema": "../app.schema",
    "$kind": "Microsoft.AdaptiveDialog",
    "generator": "multiTurnPrompt.lg",
    "recognizer": {
        "$kind": "Microsoft.RegexRecognizer",
        "intents": [
            {
                "intent": "CancelIntent",
                "pattern": "(?i)cancel"
            }
        ]
    },
    "triggers": [
        {
            "$kind": "Microsoft.OnUnknownIntent",
            "actions": [
                {
                    "$kind": "Microsoft.SendActivity",
                    "activity": "You said '${turn.activity.text}'"
                }
            ]
        }
    ]
}
```

#### NOTE

Файл `$schema` позволяет использовать IntelliSense. Если ключевое слово `$schema` отсутствует или значение `$schema` ссылается на файл схемы, который не удается найти, то предупреждение или ошибка не возникнет, и все, кроме IntelliSense, будет работать должным образом.

Элементы, определяемые в DIALOG-файле.

```
{
    "$schema": "../app.schema", ← Reference to the schema file
    "$kind": "Microsoft.AdaptiveDialog", ← This defines what type of .dialog file this is
    "generator": "multiTurnPrompt.lg", ← Reference to the language generation (.lg) file
    "recognizer": { ← Beginning of the recognizer section which specifies the type of
        "$kind": "Microsoft.RegexRecognizer", ← recognizer used and an array of intents and optional entities
        "intents": [ ← Begins the array of intents
            {
                "intent": "CancelIntent", ← Begins the first element of the array of intents
                "pattern": "(?i)cancel" ← The intent name of the first (and only) intent in the array
            }
        ] ← Ends the first element in the array of intents
    }, ← Ends the recognizer section
    "triggers": [ ← Begins the array of triggers
        {
            "$kind": "Microsoft.OnUnknownIntent", ← Begins the first element of the array of triggers
            "actions": [ ← Begins the array of actions that are part of this trigger
                {
                    "$kind": "Microsoft.SendActivity", ← Begins the first element of the array of actions
                    "activity": "You said '${turn.activity.text}'" ← This is a SendActivity action
                }
            ] ← Ends the first element in the array of actions
        } ← Ends the array of actions
    ] ← Ends the first element in the array of triggers
}
```

Сведения о том, как создавать `.dialog` файлы автоматически с помощью команды CLI `luis:build` при создании приложения Luis, см. в разделе "файл диалогового окна" статьи **развертывание ресурсов Luis с помощью команд интерфейса командной строки Bot Framework Luis**.

Сведения о том, как создавать `.dialog` файлы автоматически с помощью команды CLI `qnamaker:build` при

создании QnA Maker базы знаний, см. в разделе "файл диалогового окна" статьи **развертывание QnA Maker базы знаний с помощью команд интерфейса командной строки Bot Framework qnamaker**.

## LG-файлы

Декларативные файлы диалога, имеющие расширение LG, подробно описаны в статье [Формат файлов LG](#).

## Обозреватель ресурсов

Обозреватель ресурсов предоставляет инструменты, позволяющие импортировать декларативные файлы адаптивных диалогов в бот и использовать эти адаптивные диалоги, как будто они были определены в исходном коде бота.

С помощью обозревателя ресурсов можно создать объекты ресурсов, содержащие все необходимые сведения о декларативных файлах, необходимых для создания адаптивных диалогов во время выполнения. Для этого используется загрузчик типов обозревателя ресурсов, который импортирует файлы с расширениями DIALOG и LG.

### Объект ресурса

Метод *получения ресурса* в обозревателе ресурсов считывает декларативный файл в объект ресурса. Объект ресурса содержит сведения о декларативном файле и может использоваться любым процессом, который должен ссылаться на него, например загрузчиком типов.

```
var resource = this.resourceExplorer.GetResource("main.dialog");
```

### Загрузчик типов

После того как метод обозревателя ресурсов `_get_resource_` считывает декларативный файл в объект ресурса, объект `_load_type_method_` приводит ресурс к `AdaptiveDialog` объекту. Объект `AdaptiveDialog` может использоваться так же, как любой другой недекларативный адаптивный диалог, при создании с помощью диспетчера диалогов.

```
dialogManager = new DialogManager(resourceExplorer.LoadType<AdaptiveDialog>(resource));
```

### Автоматическая перезагрузка диалогов при изменении файла

При каждом изменении декларативного файла во время работы бота возникает событие *Changed*. Вы можете перехватить это событие и перезагрузить декларативные файлы. Таким образом, когда потребуется обновить какой-либо адаптивный диалог, вам не нужно будет обновлять код и повторно компилировать исходный код или перезапускать бот. Это может быть особенно удобно в рабочей среде.

```
// auto reload the root dialog when it changes
this.resourceExplorer.Changed += (e, resources) =>
{
    if (resources.Any(resource => resource.Id == "main.dialog"))
    {
        Task.Run(() => this.LoadRootDialogAsync());
    }
};

private void LoadRootDialogAsync()
{
    var resource = this.resourceExplorer.GetResource("main.dialog");
    dialogManager = new DialogManager(resourceExplorer.LoadType<AdaptiveDialog>(resource));
}
```

## VersionChanged Событие

Если декларативные ресурсы были перезагружены для внесения изменений в `.dialog` файл, может потребоваться переоценить состояние всех текущих диалогов, чтобы учесть любые изменения в логике диалогового окна. Изменения в логике могут быть простыми, например, для очистки стека бесед и перезапуска. Более сложная логика позволит вам выполнять такие действия, как перезапуск диалогового окна с сохранением данных, что позволяет выбрать новые свойства и пути, которые ранее не существовали.

`DialogEvents.VersionChanged` Событие захватывается с помощью `OnDialogEvent` триггера.

```
Triggers = new List<OnCondition>()
{
    new OnDialogEvent(DialogEvents.VersionChanged)
    {
        Actions = new List<Dialog>()
        {
            new SendActivity("The VersionChanged event fired.")
        }
    }
}
```

## Декларативные ресурсы

В пакете SDK для Bot Framework доступны различные декларативные ресурсы, перечисленные ниже. Эти ресурсы можно указывать в DIALOG-файлах в качестве значения `$kind`.

### Триггеры

Этот раздел содержит все [триггеры](#), сгруппированные по типу.

#### Базовый триггер

ЗНАЧЕНИЕ <code>\$KIND</code>	ИМЯ ТРИГГЕРА	ДЕЙСТВИЕ, ВЫПОЛНЯЕМОЕ ТРИГГЕРОМ
<code>Microsoft.OnCondition</code>	<code>OnCondition</code>	Триггер <code>OnCondition</code> является базовым триггером, от которого наследуют все остальные триггеры. Триггеры в аддитивном диалоге определяются как список триггеров <code>OnCondition</code> .

#### Триггеры событий распознавателя

ЗНАЧЕНИЕ <code>\$KIND</code>	ИМЯ ТРИГГЕРА	ОПИСАНИЕ
<code>Microsoft.OnChooseIntent</code>	<code>OnChooseIntent</code>	Этот триггер выполняется при обнаружении неоднозначности результатов из нескольких распознавателей в <a href="#">CrossTrainedRecognizerSet</a> .
<code>Microsoft.OnIntent</code>	<code>OnIntent</code>	Действия, выполняемые при распознавании указанного намерения.

ЗНАЧЕНИЕ \$KIND	ИМЯ ТРИГГЕРА	ОПИСАНИЕ
Microsoft.OnQnAMatch	OnQnAMatch	Этот триггер выполняется, когда <a href="#">QnAMakerRecognizer</a> возвращает намерение <code>QnAMatch</code> . Сущность <code>@answer</code> будет содержать ответ <code>QnAMaker</code> .
Microsoft.OnUnknownIntent	OnUnknownIntent	Действия, выполняемые, когда входные данные пользователя не распознаны или не соответствуют ни одному из триггеров <code>OnIntent</code> .

#### Триггеры событий диалога

ЗНАЧЕНИЕ \$KIND	ИМЯ ТРИГГЕРА	ОПИСАНИЕ
Microsoft.OnBeginDialog	OnBeginDialog	Действия, выполняемые при запуске этого диалога. Для использования только в дочерних диалогах.
Microsoft.OnCancelDialog	OnCancelDialog	Это событие позволяет предотвратить отмену текущего диалога из-за дочернего диалога, выполняющего действие <code>CancelAllDialogs</code> .
Microsoft.OnEndOfActions	OnEndOfActions	Это событие порождается после обработки всех действий и событий неоднозначности.
Microsoft.OnError	OnError	Действие, выполняемое при возникновении события <code>Error</code> диалога. Это событие аналогично <code>OnCancelDialog</code> в том, что не позволяет завершить текущий диалог, в данном случае из-за ошибки в дочернем диалоге.
Microsoft.OnRepromptDialog	OnRepromptDialog	Действия, выполняемые при возникновении события <code>RepromptDialog</code> .

#### IMPORTANT

Не используйте триггер `OnBeginDialog` в корневом диалоге, так как это может вызвать проблемы. Вместо этого можно использовать триггер `OnUnknownIntent`, который будет срабатывать при запуске корневого диалога.

#### TIP

Большинство дочерних диалогов содержит триггер `OnBeginDialog`, реагирующий на событие `beginDialog`. Этот триггер автоматически срабатывает при запуске диалога, что позволяет боту немедленно реагировать приветствием или [запросом ввода данных пользователем](#).

```
{
    "$schema": "../app.schema",
    "$kind": "Microsoft.AdaptiveDialog",
    "triggers": [
        {
            "$kind": "Microsoft.OnBeginDialog",
            "actions": [
                {
                    "$kind": "Microsoft.SendActivity",
                    "activity": "Hello world!"
                }
            ]
        }
    ]
}
```

### Триггеры событий действия

Триггеры действия позволяют связать действия с любым входящим действием клиента. Например, когда новый пользователь присоединяется и бот начинает новую беседу. Дополнительные сведения о действиях можно найти в разделе [Схема действия в Bot Framework](#).

ЗНАЧЕНИЕ \$KIND	ИМЯ ТРИГГЕРА	ОПИСАНИЕ
Microsoft.OnConversationUpdateActivity	OnConversationUpdateActivity	Обработка событий, возникающих, когда пользователь начинает новую беседу с ботом.
Microsoft.OnEndOfConversationActivity	OnEndOfConversationActivity	Действия, выполняемые при получении действия типа EndOfConversation .
Microsoft.OnEventActivity	OnEventActivity	Действия, выполняемые при получении действия типа Event .
Microsoft.OnHandoffActivity	OnHandoffActivity	Действия, выполняемые при получении действия типа HandOff .
Microsoft.OnInvokeActivity	OnInvokeActivity	Действия, выполняемые при получении действия типа Invoke .
Microsoft.OnTypingActivity	OnTypingActivity	Действия, выполняемые при получении действия типа Typing .

### Триггеры событий сообщения

Триггеры **событий сообщения** позволяют реагировать на любое событие сообщения, например, когда сообщение изменено (`MessageUpdate`) либо удалено (`MessageDeletion`), или когда кто-либо реагирует (`MessageReaction`) на сообщение (например, к распространенным реакциям на сообщение относятся такие действия, как отметка "Нравится", "Сердечко", "Смех", "Удивление" и "Злость").

События сообщения относятся к типу события действия, поэтому все события сообщения имеют базовое событие `ActivityReceived`, которое дополнительно уточняется с помощью `ActivityType`. Базовый класс, от которого наследуются все триггеры сообщения, — `OnActivity`.

ЗНАЧЕНИЕ \$KIND	ИМЯ ТРИГГЕРА	ОПИСАНИЕ
Microsoft.OnMessageActivity	OnMessageActivity	Действия, выполняемые при получении действия типа <code>MessageReceived</code> .
Microsoft.OnMessageDeleteActivity	OnMessageDeleteActivity	Действия, выполняемые при получении действия типа <code>MessageDelete</code> .
Microsoft.OnMessageReactionActivity	OnMessageReactionActivity	Действия, выполняемые при получении действия типа <code>MessageReaction</code> .
Microsoft.OnMessageUpdateActivity	OnMessageUpdateActivity	Действия, выполняемые при получении действия типа <code>MessageUpdate</code> .

#### Триггеры пользовательских событий

Можно порождать собственные события, добавив действие `EmitEvent` в любой триггер. Затем можно будет обработать это пользовательское событие в любом триггере любого диалога бота, определив триггер *настраиваемого события*. Триггер пользовательского события — это триггер типа `OnDialogEvent`, в котором свойство `event` имеет то же значение, что и свойство `event name` выданного события.

#### TIP

Можно разрешить другим диалогам бота работать с пользовательским событием, задав для свойства `bubble event` выдаваемого события значение `true`.

ЗНАЧЕНИЕ \$KIND	ИМЯ ТРИГГЕРА	ОПИСАНИЕ
Microsoft.OnDialogEvent	OnDialogEvent	Действия, выполняемые при обнаружении настраиваемого события. Для порождения настраиваемого события используйте действие <a href="#">порождения настраиваемого события</a> .

## Действия

Этот раздел содержит все [действия](#), сгруппированные по типу.

#### Отправка ответа пользователю

ЗНАЧЕНИЕ \$KIND	НАЗВАНИЕ ДЕЙСТВИЯ	НАЗНАЧЕНИЕ ЭТОГО ДЕЙСТВИЯ
Microsoft.SendActivity	<code>SendActivity</code>	Позволяет отправить действие любого типа, например ответить пользователю.

#### Запрос данных пользователя

ЗНАЧЕНИЕ \$KIND	КЛАСС ВХОДНЫХ ДАННЫХ	ОПИСАНИЕ	РЕЗУЛЬТАТЫ
-----------------	----------------------	----------	------------

ЗНАЧЕНИЕ <small>\$KIND</small>	КЛАСС ВХОДНЫХ ДАННЫХ	ОПИСАНИЕ	РЕЗУЛЬТАТЫ
<code>Microsoft.AttachmentInput</code>	<code>AttachmentInput</code>	Используется, чтобы предложить или позволить пользователю <b>передать файл</b> .	Коллекция объектов вложения.
<code>Microsoft.ChoiceInput</code>	<code>ChoiceInput</code>	Используется, чтобы предложить выбор из <b>набора параметров</b> .	Значение или индекс выбранных данных.
<code>Microsoft.ConfirmInput</code>	<code>ConfirmInput</code>	Используется для запроса <b>подтверждения</b> у пользователя.	Значение типа Boolean.
<code>Microsoft.DateTimeInput</code>	<code>DateTimeInput</code>	Используется, чтобы предложить пользователю ввести <b>дату или время</b> .	Коллекция объектов даты и времени.
<code>MicrosoftInputDialog</code>	<code>InputDialog</code>	Это базовый класс, от которого наследуются все классы входных данных. Он определяет все общие свойства.	
<code>Microsoft.NumberInput</code>	<code>NumberInput</code>	Используется, чтобы предложить пользователю ввести <b>число</b> .	Числовое значение.
<code>Microsoft.OAuthInput</code>	<code>OAuthInput</code>	Используется для того, чтобы пользователи могли <b>входить на защищенный сайт</b> .	Ответ маркера.
<code>Microsoft.TextInput</code>	<code>TextInput</code>	Используется, чтобы предложить пользователю ввести <b>слово или предложение</b> .	Строка.

#### Создание условия

ДЕЙСТВИЕ ДЛЯ ВЫПОЛНЕНИЯ	ЗНАЧЕНИЕ <small>\$KIND</small>	НАЗВАНИЕ ДЕЙСТВИЯ	НАЗНАЧЕНИЕ ЭТОГО ДЕЙСТВИЯ
Ветвление: if/else	<code>Microsoft.IfCondition</code>	<code>IfCondition</code>	Используется для создания инструкций if и if-else, которые позволяют выполнять код только при соблюдении некоторого условия.
Ветвление: Switch (несколько вариантов)	<code>Microsoft.SwitchCondition</code>	<code>SwitchCondition</code>	Позволяет создать меню с множественным выбором.

ДЕЙСТВИЕ ДЛЯ ВЫПОЛНЕНИЯ	ЗНАЧЕНИЕ <small>\$KIND</small>	НАЗВАНИЕ ДЕЙСТВИЯ	НАЗНАЧЕНИЕ ЭТОГО ДЕЙСТВИЯ
Цикл: для каждого элемента	Microsoft.Foreach	ForEach	Циклический перебор набора значений, хранящихся в массиве.
Цикл: для каждой страницы (несколько элементов)	Microsoft.ForeachPage	ForEachPage	Постстраничный циклический перебор набора значений, хранящихся в массиве.
Выход из цикла	Microsoft.BreakLoop	BreakLoop	Прерывание цикла.
Продолжение цикла	Microsoft.ContinueLoop	ContinueLoop	Продолжение цикла.
Переход к другому действию	Microsoft.GotoAction	GotoAction	Немедленно переходит к указанному действию и возобновляет выполнение. Определяется <code>actionId</code> .

#### Управление диалогами

ЗНАЧЕНИЕ <small>\$KIND</small>	НАЗВАНИЕ ДЕЙСТВИЯ	НАЗНАЧЕНИЕ ЭТОГО ДЕЙСТВИЯ
Microsoft.BeginDialog	BeginDialog	Начинает выполнять другой диалог. Когда диалог завершится, будет возобновлено выполнение текущего триггера.
Microsoft.CancelDialog	CancelDialog	Отменяет активный диалог. Используется, когда нужно немедленно закрыть диалог, даже не завершая текущий процесс.
Microsoft.CancelAllDialogs	CancelAllDialogs	Отменяет все активные диалоги, включая родительские. Используйте это действие, если нужно извлечь все диалоги из стека, а очистить стек диалогов можно с помощью метода отмены всех диалогов в контексте диалога. Создает событие <code>CancelAllDialogs</code> .
Microsoft.EndDialog	EndDialog	Завершает активный диалог. Используйте это действие, если перед выходом нужно завершить работу диалога и возвратить результаты. Создает событие <code>EndDialog</code> .
Microsoft.EndTurn	EndTurn	Завершает текущий шаг диалога, не закрывая сам диалог.
Microsoft.RepeatDialog	RepeatDialog	Используется для перезапуска родительского диалога.

ЗНАЧЕНИЕ \$KIND	НАЗВАНИЕ ДЕЙСТВИЯ	НАЗНАЧЕНИЕ ЭТОГО ДЕЙСТВИЯ
Microsoft.ReplaceDialog	ReplaceDialog	Замена текущего диалога новым диалогом.
Microsoft.UpdateActivity	UpdateActivity	Позволяет обновить отправленное действие.
Microsoft.DeleteActivity	DeleteActivity	Позволяет удалить отправленное действие.
Microsoft.GetActivityMembers	GetActivityMembers	Позволяет получить список элементов действия и сохранить их в свойство в <a href="#">памяти</a> .
Microsoft.GetConversationMembers	GetConversationMembers	Позволяет получить список участников беседы и сохранить их в свойство в <a href="#">памяти</a> .
Microsoft.EditActions	EditActions	Позволяет изменять в режиме реального времени текущую последовательность действий на основе введенных пользователем данных. Особенно полезно при обработке <a href="#">прерываний</a> .

#### Управление свойствами

ЗНАЧЕНИЕ \$KIND	НАЗВАНИЕ ДЕЙСТВИЯ	НАЗНАЧЕНИЕ ЭТОГО ДЕЙСТВИЯ
Microsoft.EditArray	EditArray	Это действие позволяет выполнять с массивом операции редактирования.
Microsoft.DeleteProperty	DeleteProperty	Позволяет удалить свойство из <a href="#">памяти</a> .
Microsoft.DeleteProperties	DeleteProperties	Позволяет удалять одним действием сразу несколько свойств.
Microsoft SetProperty	SetProperty	Позволяет задать значение для свойства в <a href="#">памяти</a> .
Microsoft.SetProperties	SetProperties	Позволяет инициализировать одним действием одно или несколько свойств.

#### Доступ к внешним ресурсам

ЗНАЧЕНИЕ \$KIND	НАЗВАНИЕ ДЕЙСТВИЯ	НАЗНАЧЕНИЕ ЭТОГО ДЕЙСТВИЯ
Microsoft.BeginSkill	BeginSkill	Используйте адаптивный диалог навык для выполнения навыков.
Microsoft.HttpRequest	HttpRequest	Позволяет отправить HTTP-запросы к любой конечной точке.

ЗНАЧЕНИЕ \$KIND	НАЗВАНИЕ ДЕЙСТВИЯ	НАЗНАЧЕНИЕ ЭТОГО ДЕЙСТВИЯ
Microsoft.EmitEvent	EmitEvent	Позволяет выполнить пользовательское событие, на которое бот будет реагировать через механизм <a href="#">пользовательского триггера</a> .
Microsoft.SignOutUser	SignOutUser	Позволяет текущему пользователю выйти из системы.

#### Варианты для отладки

ЗНАЧЕНИЕ \$KIND	НАЗВАНИЕ ДЕЙСТВИЯ	НАЗНАЧЕНИЕ ЭТОГО ДЕЙСТВИЯ
Microsoft.LogAction	LogAction	Записывает сообщение в консоль и (необязательно) отправляет его как действие трассировки.
Microsoft.TraceActivity	TraceActivity	Отправляет действие трассировки с произвольными полезными данными.

#### Распознаватели

Пакет SDK для Bot Framework предоставляет несколько десятков различных распознавателей. вы указываете, какой распознаватель следует использовать в файле Dialog, например:

```
{
  "$schema": "../app.schema",
  "$kind": "Microsoft.AdaptiveDialog",
  "recognizer": {
    "$kind": "Microsoft.RegexRecognizer",
    "intents": [
      {
        "intent": "JokeIntent",
        "pattern": "(?i)joke"
      },
      {
        "intent": "FortuneTellerIntent",
        "pattern": "(?i)fortune|future"
      },
      {
        "intent": "CancelIntent",
        "pattern": "(?i)cancel"
      }
    ]
  },
  ...
}
```

Адаптивные диалоги поддерживают следующие распознаватели:

- [распознаватель RegexRecognizer](#);
- [Распознаватель для LUIS](#)
- [распознаватель QnA Maker](#);
- [многоязычный распознаватель](#);
- [набор взаимно-обучаемых распознавателей](#);
- [RecognizerSet](#).

## Генераторы

Значение [generator](#) содержит ссылку на LG-файл, связанный с адаптивным диалогом, который определяет данный DIALOG-файл. Пример:

```
{  
  "$schema": "../app.schema",  
  "$kind": "Microsoft.AdaptiveDialog",  
  "generator": "MyBotGeneratorFile.lg",  
  ...  
}
```

## Интерфейс Command-Lineной платформы Bot

Несколько новых команд [Command-Line интерфейса командной строки \(CLI\)](#) были добавлены в выпуске адаптивных диалоговых окон в пакете SDK для Bot Framework. Сюда входят две команды, связанные с диалогом, для работы с `.dialog` `.schema` файлами и которые очень полезны при использовании декларативного подхода к адаптивной разработке диалоговых окон.

Новая группа CLI `dialog` имеет две следующие команды: `dialog:merge` и `dialog:verify`.

### Команда merge

Файл корневой схемы содержит схемы всех компонентов, используемых программой Bot. Каждому потребителю декларативных файлов, включая [Composer](#), требуется файл схемы.

`dialog:merge` Команда используется для создания файла схемы проекта. Эту команду необходимо выполнять в любое время, когда вы добавляете новый пакет или создаете или изменяете собственные компоненты.

При этом в текущем каталоге создается файл с именем `app.Schema`, если не указано иное с помощью `-o` параметра. На этот файл ссылается `"$schema"` ключевое слово в каждом файле `.dialog` проекта.

#### NOTE

Допустимый файл App. Schema необходим для использования [интеллектуальных средств завершения кода](#), таких как [IntelliSense](#), для работы с любыми декларативными ресурсами.

Чтобы использовать команду MERGE, введите следующую команду в командной строке, а в корневом каталоге проекта:

```
bf dialog:merge <filename.csproj>
```

Дополнительные сведения об использовании этой команды см. в разделе [диалоговое окно BF. слияние](#) в файле сведений интерфейса командной строки BF Luis.

Пример см. в статье Создание [файла схемы](#) в разделе [Создание программы-робота с помощью декларативных адаптивных диалоговых окон](#).

### Команда verify

`dialog:verify` Команда проверит `.dialog` файлы, чтобы убедиться, что они совместимы со схемой.

Чтобы использовать `verify` команду, введите следующую команду в командной строке, а в корневом каталоге проекта:

```
bf dialog:verify <filename.csproj>
```

Дополнительные сведения об использовании этой команды см. в разделе [диалоговое окно BF. Проверьте](#) в файле сведений о BF CLI Luis.

#### NOTE

[Composer](#) создает Объединенный `.schema` файл и допустимые `.dialog` файлы, но команда Verify может быть очень полезной, если вы создаете эти файлы вручную.

## Установка интерфейса командной строки для Bot Framework

Для работы с Bot Framework CLI требуется [Node.js](#).

1. Убедитесь, что установлена последняя версия NPM:

```
npm i -g npm
```

2. С помощью Node.js установите последнюю версию Bot Framework CLI из командной строки.

```
npm i -g @microsoft/botframework-cli
```

Дополнительные сведения см. в статье [Bot Framework CLI Tool](#).

#### Релевантная информация

- [Команды диалогового окна](#)

## Дополнительные сведения

- Узнайте, как [создать бот с использованием декларативных адаптивных диалогов](#).

# Перекрестное обучение моделей LUIS и QnA Maker

27.03.2021 • 21 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Можно выполнить перекрестное обучение файлов. luis и QnA, чтобы каждое Адаптивное диалоговое окно знало о возможностях других адаптивных диалоговых окон программы-робота. Это позволяет активному адаптивному диалоговому окну откладываться на другой распознаватель, либо в том же диалоговом окне, либо в другом, когда пользователь вводит любой запрос или ввод, который не может самостоятельно справиться с ним.

## Общие сведения о перекрестном обучении

Адаптивные диалоговые окна предлагают диалоговое окно для моделирования диалоговых окон, при этом каждое Адаптивное диалоговое окно имеет собственную модель понимания языка (LU). Хотя это дает программам-роботам чрезвычайно гибкие возможности, она также может представлять некоторые проблемы. Перекрестное обучение может оказаться полезным, когда для адаптивного диалогового окна может потребоваться выяснить, когда пользователь выражает нечто, которое может быть обработано другим диалоговым окном в роботе. Ниже приведены некоторые примеры, в которых может оказаться полезным перекрестное обучение.

- Робот, позволяющий пользователям отменять текущий поток разговора или запрашивать помощь, не создавая дублирующие методы отмены или справки в каждом адаптивном диалоговом окне.
- Программа-робот, которая поддерживает нелинейные беседы, в которых пользователь может подумать о том, что они ранее забыли, или просто изменили их в посередине по задаче.
- Робот, позволяющий пользователю исправить предоставленные ранее сведения.

В статье [обработка прерываний в адаптивных диалоговых окнах](#) появилось понятие прерываний в адаптивных диалоговых окнах. В нем объясняется, как можно получить доступ к родительскому адаптивному диалоговому окну, когда средство распознавания активного адаптивного диалогового окна не находит подходящее совпадение. В этом случае активное диалоговое окно не знает, можно ли ответить на родительское или родственное диалоговое окно, чтобы выяснить, что оно должно отправить фразы продолжительностью или вопрос родительскому элементу с помощью механизма *консультации* платформы Bot.

Перекрестное обучение может создавать и улучшать возможности, обеспечиваемые прерываниями, несколькими способами:

1. **Обучение в нескольких диалоговых окнах.** Перекрестное обучение моделей LU всех адаптивных диалоговых окон программы-робота дает каждому диалоговому окну возможность определять, могут ли другие диалоги отвечать на запросы пользователя. Таким образом, роботу не нужно обращаться ко всему стеку диалоговых окон, чтобы определить, может ли другое диалоговое окно лучше обрабатывать данные, вводимые пользователем. Это подробно описано в разделе [Luis-to-Luis Cross Training](#).
2. **Обучение в диалоговом окне.** Это перекрестное обучение разных языковых механизмов в рамках одного диалогового окна. LUIS и QnA Maker — это разные языковые механизмы. После того как модели для каждого из них перекрестно обучены, можно обратиться к распознавателю для обоих пользователей, чтобы определить, какой из них лучше всего подходит для

реагирования на запрос пользователя. Более подробно этот процесс описан в разделе "перекрестное обучение Luis и QnA Maker Models".

#### TIP

Если языки, связанные с различными адаптивными диалогами в Bot, не обучены, то фразы продолжительностью или вопросы из других диалоговых окон будут рассматриваться только в том случае, если средство распознавания активного диалогового окна не вернет неизвестное намерение. Когда модели были перекрестно обучены, фразы продолжительностью и вопросы из родительских и одноуровневого диалоговых окон всегда будут рассматриваться, так как они связаны с новой намерением, созданной в модели «основные сведения о языке активного диалогового окна». К родителю больше не нужно обращаться для выяснения, может ли он отвечать на введенные пользователем данные. При возвращении этой новой цели программа-Bot знает, что ее обрабатывает родительскую или родственную платформу.

## Перекрестное обучение LUIS-to-LUIS

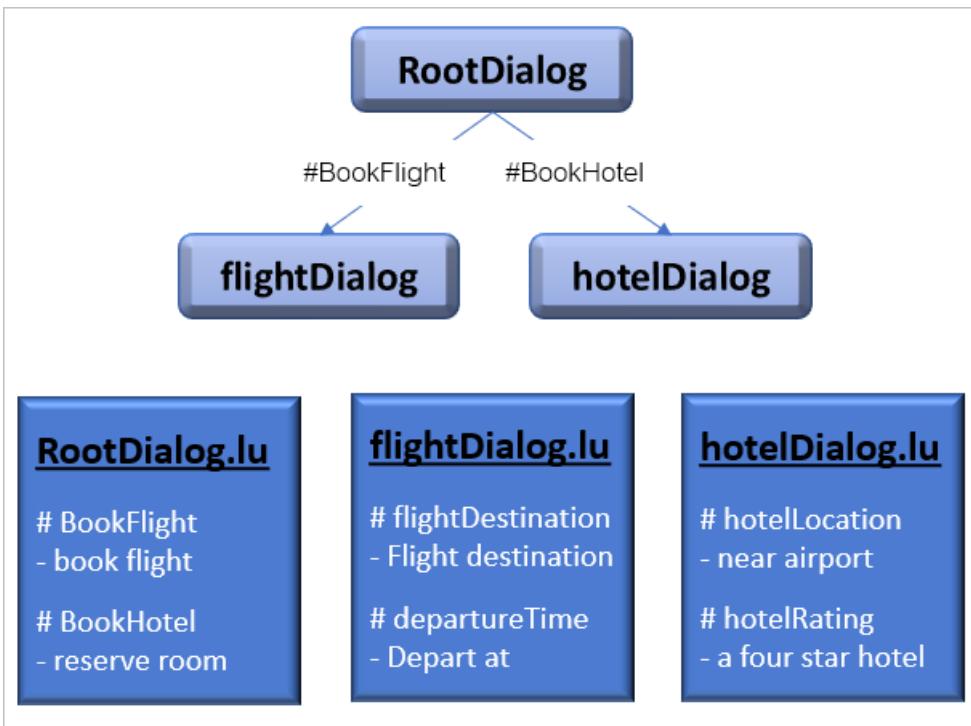
Перекрестное обучение моделей LUIS в коде робота позволяет повысить способность программы-робота управлять *глобальными прерываниями*. Это означает, что если в активном адаптивном диалоговом окне нет триггеров, которые могут обрабатывать намерение, возвращенное распознавателем, то Bot восправит его в родительское диалоговое окно, используя *механизм консультации* по адаптивным диалогам. Если в родительском диалоговом окне нет триггера, обрабатывающего намерение, он переходит в состояние, пока не достигнет корневого диалогового окна. Как только цель будет обработана, поток диалога продолжится там, где она была отключена. Более подробное обсуждение глобальных прерываний см. в [статье обработка перерывов](#) в разделе *обработка прерываний в адаптивных диалоговых окнах*. Для моделей LUIS, перекрестных обучении с помощью других моделей LUIS, будет использоваться [распознаватель Luis](#).

Типичные варианты использования глобальных прерываний включают создание базовых функций управления диалогами, таких как Справка или Отмена в корневом диалоговом окне, которые затем доступны для любого из его дочерних диалоговых окон, как упоминалось во введении. Глобальные прерывания также позволяют пользователям легко изменять направление потоковой передачи. Рассмотрим следующий пример в этом вымышленном роботе путешествия.

### Робот поездки

В следующем примере программы-робота в качестве примера демонстрируется перекрестное обучение ленты с несколькими адаптивными диалогами, каждый из которых имеет собственную модель LUIS.

Этот робот состоит из трех адаптивных диалоговых окон, каждый из которых имеет собственную модель LUIS:



1. RootDialog
  - Целей: `BookFlight`, `BookHotel`.
2. флигхтдиалог
  - Целей: `flightDestination`, `departureTime`.
3. хотелдиалог
  - Целей: `hotelLocation`, `hotelRating`.

#### Сценарий для клиентов Bot в командировках

В следующем примере Exchange демонстрируется потенциальный сценарий использования клиента, который не может быть обработан программой Bot, но может быть после перекрестного обучения моделей LUIS Bot.

Bot: Привет, как я могу помочь вам на сегодняшний день?

**Пользователь:** я хочу зарезервировать перелет

Bot: отлично, я могу помочь вам. Что такое Дата отправления?

Bot будет ожидать ответа на вопрос о том, что такое *Дата отправления*?, что примерно равно 29 февраля, но как будет обрабатывать прерывание потоковой передачи, когда пользователь получит сообщение о том, что диалоговое окно рейса не может быть обработано? Что-то вроде следующего:

**Пользователь:** сначала необходимо зарезервировать комнату

В приведенном выше примере, когда пользователь запросил попытаться зарегистрировать перелет, распознаватель корневого диалогового окна вернул `BookFlight` намерение. При этом выполняется `flightDialog`. Адаптивное диалоговое окно, которое обрабатывает авиарейсы, но не знает о гостиницах. Когда пользователь запрашивает резервирование отеля, диалоговое окно рейса не распознается, поскольку utterance «мне нужно зарезервировать комнату First» не соответствует каким-либо смыслам в диалоговых окнах «Flight» LUIS Model.

После перекрестного обучения файлов с. lu программа-робот теперь сможет определить, что пользователь запрашивает что-то, на которое может ответить другое диалоговое окно, чтобы передать запрос на его родительский диалог, в данном случае это корневое диалоговое окно, которое

обнаруживает `BookHotel` намерение. При этом выполняется `hotelDialog` Адаптивное диалоговое окно, которое обрабатывает резервирования гостиницы. После того, как диалоговое окно отеля закончит запрос на резервирование отеля, управление передается обратно в диалоговое окно рейса, чтобы завершить бронирование авиабилетов.

#### Перекрестное обучение моделей LUIS робота «путешествие»

Чтобы этот вымышленный робот поездки обрабатывал прерывание в предыдущем примере, необходимо обновить модель LUIS диалогового окна рейсов, содержащуюся в файле `flightDialog.lu`, чтобы включить новое намерение с именем `_interruption`, а затем добавить фразы продолжительностью для `BookHotel` цели. Файл `flightDialog.lu` используется для создания приложения Luis, связанного с диалоговым окном рейса.

#### TIP

Перекрестное обучение. все модели LUIS в стандартном роботе могут быть очень сложным и утомительным процессом. Существует команда, входящая в интерфейс командной строки Bot Framework (BF CLI), которая автоматизирует эту работу. Это подробно описано в разделе [команда перекрестной подготовки CLI для интерфейса командной строки](#).

Перед этим обновлением пример файла "Flight бронирования. Lu" выглядит следующим образом:

```
# flightDestination
- book a flight

# departureTime
- I need to depart next thursday
```

После перекрестного обучения с помощью файла бронирования. lu он будет выглядеть следующим образом:

```
# flightDestination
- book a flight

# departureTime
- I need to depart next thursday

# _Interruption
- reserve a hotel room
```

Utterance *резервирует комнату в отеле*, связанную с `_interruption` намерением. При `_interruption` обнаружении намерения он передает все utterance, связанные с ним, в родительское диалоговое окно, распознаватель которого возвращает `BookHotel` намерение. При перекрестном обучении LUIS в LUIS необходимо включить все пользовательские фразы продолжительностью из всех целей из диалогового окна, с которым выполняется перекрестное обучение.

# After cross training

## RootDialog.lu

# BookFlight  
- book flight  
  
# BookHotel  
- reserve room

## flightDialog.lu

# flightDestination  
- Flight destination  
  
# departureTime  
- Depart at  
  
# \_interruption  
- reserve room

## hotelDialog.lu

# hotelLocation  
- near airport  
  
# hotelRating  
- a four star hotel  
  
# \_interruption  
- Book flight

### IMPORTANT

На прогнозы LUIS влияет количество фразы продолжительностью в каждом намерении. Если у вас есть цель с 100 примером фразы продолжительностью и цель с 20 примерами фразы продолжительностью, цель 100-utterance будет иметь более высокий уровень прогноза, и, скорее всего, будет выбрано больше. Это может повлиять на перекрестные модели, поскольку все фразы продолжительностью из моделей всех родительских и родственных диалоговых окон становится фразы продолжительностью новой `_Interruption` цели. В некоторых случаях это может привести к тому, что в родительском или родственном диалоговом окне пользователь будет отвечать на запросы пользователя, когда допустимые соответствия будут возвращены распознавателем текущих диалоговых окон до перекрестного обучения. При необходимости сократите их влияние, ограничив число примеров фразы продолжительностью в новой `_Interruption` цели.

## Перекрестное обучение LUIS и моделей QnA Maker

Хорошо спроектированная программа-робот может отвечать на соответствующие вопросы по продуктам или службам, независимо от того, какое диалоговое окно активно в данный момент. LUIS идеально подходит для обработки диалоговых потоков, а QnA Maker идеально подходит для обработки пользователей и часто задаваемых вопросов. Наличие доступа к обеим функциям в адаптивных диалоговых окнах может улучшить возможности программы-роботы в соответствии с потребностями пользователей.

Чтобы включить эту возможность, выполните *перекрестное обучение* файлов. lu и. QnA, чтобы включить сведения, необходимые распознавателю, чтобы определить, какой ответ, LUIS или QnA Maker, лучше всего подходит для пользователя. Для модели LUIS, обученной с помощью QnA Maker модели, будет использоваться [набор расученных распознавателей](#).

Прежде чем создавать приложения LUIS и QnA Maker базу знаний, необходимо *переучить* файлы. lu и. QnA, чтобы включить сведения, необходимые распознавателю робота, чтобы определить, подходит ли для пользователя ответ LUIS или QnA Maker.

Для каждого адаптивного диалогового окна, имеющего связанный файл. lu и. QnA, при перекрестном обучении этих файлов выполняются следующие обновления:

1. В файлах. lu добавляется новое намерение с именем `DeferToRecognizer_qna_<dialog-file-name>` . Каждый вариант вопроса и вопроса из соответствующего qna-файла превращается в utterance, связанный с этим новым намерением.
2. В QnA-файлах добавляется новый ответ с именем И `intent=DeferToRecognizer_luis_<dialog-file-name>` каждый utterance из каждой цели в соответствующем Lu-файле. Эти фразы продолжительностью становятся вопросами, связанными с этим ответом. Кроме того, все фразы продолжительностью из файлов с ссылкой на файлы. lu также становятся вопросами, связанными с этим ответом.

Когда пользователь осуществляет обратную передачу с помощью программы-робота, `CreateCrossTrainedRecognizer` распознаватель отправляет введенные данные пользователя в Luis и базу знаний QnA Maker для обработки.

### Ответы распознавателя

В следующей таблице показана матрица возможных ответов и результирующее действие, предпринимаемое программой Bot.

ФУНКЦИЯ РАСПОЗНАВАНИЯ LUIS ВОЗВРАЩАЕТ	QNA MAKER РАСПОЗНАВАТЕЛЬ ВОЗВРАЩАЕТ	ОКОНЧАТЕЛЬНЫЕ РЕЗУЛЬТАТЫ
Допустимая цель LUIS	Отложить до Luis намерения	Выберите ответ от распознавателя Luis . Обработайте с помощью <code>OnIntent</code> триггера.
Отложить на QnA Maker намерение	Допустимое QnA Maker намерение	Выберите ответ от распознавателя QnA Maker . Обработайте с помощью <code>OnQnAMatch</code> триггера.
Отложить на QnA Maker намерение	Отложить до Luis намерения	<code>UnknownIntent</code> Событие создается распознавателем. Обработайте с помощью <code>OnUnknownIntent</code> триггера.
Допустимая цель LUIS	Допустимое QnA Maker намерение	<code>ChooseIntent</code> Событие создается распознавателем. Обработайте с помощью <code>OnChooseIntent</code> триггера.

#### IMPORTANT

Перекрестное обучение. все модели LUIS и QnA Maker в стандартном роботе могут быть очень сложным и утомительным процессом. Существует команда, входящая в интерфейс командной строки Bot Framework (BF CLI), которая автоматизирует эту работу. Это подробно описано в разделе [команда перекрестной подготовки CLI для интерфейса командной строки](#).

Выполнение этой команды в проекте-роботе, который содержит модели LUIS и QnA Maker, автоматически перекрестно обучить LUIS LUIS по всем адаптивным диалогам во всем проекте, а также LUIS для QnA Maker перекрестного обучения в каждом из адаптивных диалоговых окон, имеющих обе модели, то есть файлы. luis и QnA.

### Перекрестное обучение нескольких LUIS и моделей QnA Maker

Перекрестное обучение с помощью моделей LUIS и QnA Maker повышает глобальное прерывание, как описано выше в [Luis to Luis Cross Training](#). Это также относится к QnA Maker. Пример:

- Когда модель LUIS корневого диалогового окна перекрестно обучена с помощью QnA Makerной модели корневого диалогового окна, команда создает `DeferToRecognizer_qna` намерение в RootDialog.luis, при этом все вопросы указаны как фразы продолжительностью.
- Далее, когда дочерний диалог является подчиненным, он выбирает эти смыслы и, в свою очередь, передает их дочернему диалоговому окну, и это продолжается до тех пор, пока не появится никаких дочерних диалоговых окон.
- Когда пользователь запрашивает любой вопрос, связанный с Рутдиалог. QnA, когда активное диалоговое окно является дочерним или потомком, активное диалоговое окно не сможет ответить, но, поскольку оно было переработано, оно будет знать, что еще одно диалоговое окно может ответить, а затем передать ему на родительский вопрос. Вопрос, в свою очередь, перетекает до

корневого диалогового окна, который отвечает на вопрос, прежде чем вернуть управление назад к предыдущему потоку.

Преимущество глобальных прерываний в этом сценарии заключается в том, что она предоставляет возможность использовать QnA Maker базу знаний, связанную с корневым диалоговым окном, для обработки всех вопросов, которые может иметь пользователь, независимо от того, где они находятся в беседе с Bot.

#### NOTE

Глобальные прерывания приводят к созданию нескольких транзакций для служб LUIS и QnA Maker. Чем глубже иерархия диалоговых окон, тем больше транзакций может возникать для заданного запроса пользователя. Это увеличение количества транзакций может быть тем, что следует учитывать при проектировании программы-робота.

## Команда перекрестной подготовки CLI-инфраструктуры

Перекрестное обучение. программа-робот может быстро стать сложной задачей, подверженной ошибкам, даже при минимально сложном коде робота, особенно при регулярном обновлении моделей LUIS или QnA Maker. Пакет SDK для Bot Framework предоставляет средство для автоматизации этого процесса. Сведения о команде CLI-обучения для Bot Framework см. в разделе [перекрестно обученный набор распознавателей](#) в пошаговом руководстве по адаптивным диалоговым окнам.

## Обновления исходного кода

После перекрестного обучения моделей LUIS и QnA Maker необходимо убедиться в том, что перекрестно обученный распознаватель используется в качестве адаптивного распознавателя диалоговых окон в исходном коде, как описано в разделе [распознаватели в адаптивных диалоговых окнах](#) — справочное руководство.

## Дополнительные сведения

- [Создание перекрестной обученной программы Bot для использования распознавателей Luis и QnA Maker](#)
- Раздел [перекрестно обученный набор распознавателей](#) в разделе Справочное руководство по адаптивным диалоговым окнам.

# Управление ресурсами бота

27.03.2021 • 5 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Боты часто используют разные службы, например [LUIS.ai](#) или [QnAMaker.ai](#). При разработке ботов вам необходимо иметь возможность отслеживать их все. Вы можете использовать разные средства, например файлы appsettings.json, web.config или .env.

## IMPORTANT

До выпуска для Bot Framework пакета SDK версии 4.3 мы предлагали файл .bot в качестве средства управления ресурсами. Но в будущем мы рекомендуем вам использовать для этого файлы appsettings.json или .env. Боты, которые используют файл .bot, пока будут работать и дальше, хотя файл .bot был объявлен **нерекомендуемым**. Если вы используете файл .bot для управления ресурсами, выполните соответствующие шаги по переносу параметров.

## Перенос параметров из файла .bot

В разделах ниже описано, как перенести параметры из файла .bot. Следуйте подходящему для вас сценарию.

### Сценарий 1. Локальный бот с файлом .bot

В этом сценарии вам доступен локальный бот, который использует файл .bot, но *бот не был перенесен* на портал Azure. Выполните следующие шаги, чтобы перенести параметры из файла .bot в файл appsettings.json или .env.

- Если файл .bot зашифрован, расшифруйте его с помощью следующей команды:

```
msbot secret --bot <name-of-bot-file> --secret "<bot-file-secret>" --clear
```

- Откройте расшифрованный файл .bot, скопируйте значения и добавьте их в файл appsettings.json или .env.
- Обновите код, чтобы считать параметры из файла appsettings.json или .env.
  - [C#](#)
  - [JavaScript](#)

В методе `ConfigureServices` воспользуйтесь предоставляемым ASP.NET Core объектом конфигурации, например:

`Startup.cs`.

```
var appId = Configuration.GetSection("MicrosoftAppId").Value;
var appPassword = Configuration.GetSection("MicrosoftAppPassword").Value;
options.CredentialProvider = new SimpleCredentialProvider(appId, appPassword);
```

*При необходимости* подготовьте ресурсы и подключите их к своему боту с помощью файла appsettings.json или .env.

## Сценарий 2. Бот, развернутый в Azure с помощью файла .bot

В этом сценарии вы уже развернули бот на портале Azure с помощью файла .bot и теперь хотите перенести параметры из файла .bot в файл appsettings.json или .env.

- Скачайте код бота с портала Azure. При скачивании кода вам будет предложено включить файл appsettings.json или .env, который будет содержать MicrosoftAppId и MicrosoftAppPassword, а также другие дополнительные параметры.
  - Откройте скачанный файл appsettings.json или .env и скопируйте из него параметры в локальный файл appsettings.json или .env. Не забудьте удалить записи botSecret и botFilePath из локального файла appsettings.json или .env.
  - Обновите код, чтобы считать параметры из файла appsettings.json или .env.
- [C#](#)
  - [JavaScript](#)

В методе `ConfigureServices` воспользуйтесь предоставляемым ASP.NET Core объектом конфигурации, например:

`Startup.cs`.

```
var appId = Configuration.GetSection("MicrosoftAppId").Value;
var appPassword = Configuration.GetSection("MicrosoftAppPassword").Value;
options.CredentialProvider = new SimpleCredentialProvider(appId, appPassword);
```

Также необходимо удалить `botFilePath` и `botFileSecret` из раздела **параметры приложения** в **портал Azure**.

При необходимости подготовьте ресурсы и подключите их к своему боту с помощью файла appsettings.json или .env.

## Сценарий 3. Боты, которые используют файл appsettings.json или .env

В этом сценарии рассматривается случай, когда вы с нуля разрабатываете боты с помощью пакета SDK 4.3 и у вас нет файлов .bot для переноса. Все параметры, которые вы хотите использовать в своем боте, доступны в файле appsettings.json или .env, как показано ниже:

```
{
  "MicrosoftAppId": "<your-AppId>",
  "MicrosoftAppPassword": "<your-AppPwd>"
}
```

- [C#](#)
- [JavaScript](#)

Чтобы считать указанные выше параметры в коде C#, вам нужно воспользоваться предоставляемым ASP.NET Core объектом конфигурации, например: `Startup.cs`.

```
var appId = Configuration.GetSection("MicrosoftAppId").Value;
var appPassword = Configuration.GetSection("MicrosoftAppPassword").Value;
options.CredentialProvider = new SimpleCredentialProvider(appId, appPassword);
```

При необходимости подготовьте ресурсы и подключите их к своему боту с помощью файла appsettings.json или .env.

## Дополнительные ресурсы

- Подробнее см. в [инструкциях по развертыванию бота](#).
- Узнайте, как использовать [Azure Key Vault](#) для защиты и администрирования криптографических ключей и секретов, используемых облачными приложениями и службами.

# Принцип работы ботов Microsoft Teams

27.03.2021 • 19 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Это введение, которое познакомится с тем, что вы узнали в статье [как программы-роботы работают и обсуждения, управляемые событиями](#). Прежде чем читать эту статью, необходимо ознакомиться с этими статьями.

Основные различия в работе ботов, которые разрабатываются для Microsoft Teams, заключаются в способе обработки действий. *Обработчик действий групп* является производным от *обработчика действий* и обрабатывает типы действий, специфичные для групп, перед обработкой более общих типов действий.

## Обработчик действий команд

Чтобы создать программу-робот для групп, создайте класс-робот из класса *обработчика действий команды*. Когда такой робот получает действие, он направляет действие через различные *обрабатчики действий*. Начальный, базовый обработчик — это *обрабатчик включения*, который направляет действие в обработчик на основе типа действия. *Обработчик поворачивания* вызывает обработчик, предназначенный для обработки конкретного типа полученного действия. Класс *обработчика действия команды* является производным от класса *обработчика действия*. В дополнение к типам действий, которые может обработать *обработчик действия*, класс обработчика действий групп включает дополнительные обработчики для действий, связанных с командами.

Программа-робот, производная от обработчика действия команды, аналогична роботу, производному непосредственно от класса обработчика действия. Однако группы содержат дополнительные сведения в `conversationUpdate` действиях и отправляют действия, связанные с командами `invoke`.

- [C#](#)
- [JavaScript](#)
- [Python](#)

Когда программа-робот обработчика действий — получит действие сообщения, обработчик поочередности направляет действие входящего сообщения своему `OnMessageActivityAsync` обработчику, аналогично тому, как — это будет основано на обработчике действия. Однако если команды Bot получают действие обновления диалога, то версия `OnConversationUpdateActivityAsync` обработчика обрабатывает это действие.

Для большинства обработчиков действий, специфичных для команд, базовая реализация отсутствует. Вам потребуется переопределить эти обработчики и предоставить соответствующую логику для робота.

Все обработчики действий, описанные в разделе [Обработка действий](#) в диалоговых окнах, использующих события с помощью обработчика действий, будут продолжать работать, как и в случае с роботом, не являющимся участниками команды, за исключением того, что члены команды добавили или удалили действия, они будут отличаться в контексте группы, где новый член будет добавлен в группу, а не в поток сообщений. Дополнительные сведения см. в разделе [действия по обновлению команд](#).

Чтобы реализовать логику для таких обработчиков действий, специфичных для этих команд, вы переопределяете методы в Bot.

# Логика бота Teams

Логика бота обрабатывает входящие действия из одного или нескольких каналов бота и создает исходящие действия в ответ. Этот механизм сохраняется и для бота, производного от класса обработчика действий Teams, который первым делом проверяет действия для Teams, а затем передает все остальные действия обработчику действий Bot Framework.

## Действие обновления диалога Teams

В следующей таблице перечислены события команд, которые создают действие *обновления диалога* в роботе. В статье [события обновления обсуждений Microsoft Teams](#) содержатся сведения о том, как использовать каждое из этих событий.

- [C#](#)
- [JavaScript](#)
- [Python](#)

Ниже приведен список всех обработчиков действий команд, вызванных из `OnConversationUpdateActivityAsync` метода обработчика действий *команд*.

EVENTTYPE	ОБРАБОТЧИК	УСЛОВИЕ	ДОКУМЕНТАЦИЯ ПО КОМАНДАМ
channelCreated	<code>OnTeamsChannelCreatedAsync</code>	Посыпается при создании нового канала в группе, в которой установлен робот.	<a href="#">Канал создан.</a>
channelDeleted	<code>OnTeamsChannelDeletedAsync</code>	Отправляется при каждом удалении канала в группе, в которой установлена программа Bot.	<a href="#">Канал удален.</a>
channelRenamed	<code>OnTeamsChannelRenamedAsync</code>	Отправляется при каждом переименовании канала в группе, в которой установлена программа Bot.	<a href="#">Канал переименован.</a>
ченнелресторед	<code>OnTeamsChannelRestoredAsync</code>	Отправляется каждый раз, когда канал, который был удален ранее, восстанавливается в команде, в которой уже установлена программа Bot.	<a href="#">Канал восстановлен.</a>
мемберсаддед	<code>OnTeamsMembersAddedAsync</code>	По умолчанию вызывает <code>ActivityHandler.OnMembersAddedAsync</code> метод. Посыпается в первый раз, когда Bot добавляется в беседу, и каждый раз, когда новый пользователь добавляется в команду или в группу чатов, в которой установлена программа Bot.	<a href="#">Добавлены члены команды.</a>

EVENTTYPE	ОБРАБОТЧИК	УСЛОВИЕ	ДОКУМЕНТАЦИЯ ПО КОМАНДАМ
мемберсремовед	OnTeamsMembersRemovedAsync	По умолчанию вызывает <code>ActivityHandler.OnMembersRemovedAsync</code> метод. Отправляется, если программа-робот удаляется из команды и каждый раз, когда любой пользователь удаляется из команды, членом которой является Bot.	Члены команды удалены.
тимарчивед	OnTeamsTeamArchivedAsync	Посыпается, когда группа, в которой установлена программа Bot, архивируется.	Команда архивируется.
тимделетед	OnTeamsTeamDeletedAsync	Посыпается, когда группа, в которой находится Bot, была удалена.	Команда удалена.
teamRenamed	OnTeamsTeamRenamedAsync	Посыпается, когда группа, в которой находится Bot, переименована.	Команда переименована.
тамресторед	OnTeamsTeamRestoredAsync	Отправляется при восстановлении ранее удаленной команды, в которой находится Bot.	Команда восстановлена.
тамунарчивед	OnTeamsTeamUnarchivedAsync	Посыпается, если группа, в которой установлена программа Bot, не архивируется.	Команда неархивирована.

### Команды вызывают действия

В следующей таблице перечислены команды действий вызова, относящиеся к командам, которые отправляются в Bot. Перечисленные действия вызова предназначены для общения программы-роботы в командах. Пакет SDK для Bot Framework также поддерживает вызовы, относящиеся к расширениям для обмена сообщениями. Дополнительные сведения см. в статье [команды что такое расширения обмена сообщениями](#).

- [C#](#)
- [JavaScript](#)
- [Python](#)

Здесь приведен список всех обработчиков действий Teams, которые вызываются из обработчика действий `Teams OnInvokeActivityAsync`.

ВЫЗЫВАЕМЫЕ ТИПЫ	ОБРАБОТЧИК	ОПИСАНИЕ
actionableMessage/executeAction	OnTeamsO365ConnectorCardActionAsync	Teams O365 Connector Card Action.
CardAction.Invoke	OnTeamsCardActionInvokeAsync	Teams Card Action Invoke.

ВЫЗЫВАЕМЫЕ ТИПЫ	ОБРАБОТЧИК	ОПИСАНИЕ
fileConsent/invoke	<code>OnTeamsFileConsentAcceptAsync</code>	Teams File Consent Accept.
fileConsent/invoke	<code>OnTeamsFileConsentAsync</code>	Teams File Consent.
fileConsent/invoke	<code>OnTeamsFileConsentDeclineAsync</code>	Teams File Consent.
signin/verifyState	<code>OnTeamsSigninVerifyStateAsync</code>	Teams Sign in Verify State.
task/fetch	<code>OnTeamsTaskModuleFetchAsync</code>	Teams Task Module Fetch.
task/submit	<code>OnTeamsTaskModuleSubmitAsync</code>	Teams Task Module Submit.

## Дальнейшие действия

Сведения о создании ботов для Teams см. в [документации](#) для разработчика Microsoft Teams.

# Общие сведения о навыках

27.03.2021 • 13 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Начиная с версии 4.7, пакет SDK для Bot Framework позволяет расширить возможности бота за счет другого бота (навыка). Такой навык может использоваться несколькими другими ботами, что упрощает сценарии повторного использования логики. Также это позволяет создать бот, ориентированный на взаимодействие с пользователем, и дополнить его вашими собственными или сторонними навыками.

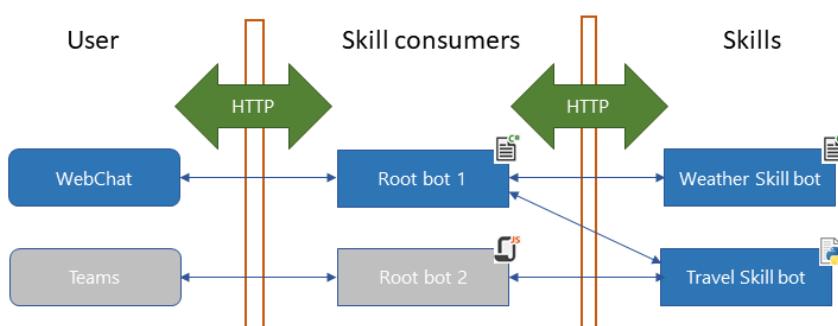
- *Навыком* здесь называется бот, который может выполнять ряд задач для другого бота. Бот может одновременно являться навыком и взаимодействовать с пользователем.
- *Потребителем навыка* называется бот, который умеет вызывать один или несколько навыков. Потребитель опыта, взаимодействующий с пользователем, называется также *корневым ботом*.
- *Манифест навыка* — это файл в формате JSON с описанием действий, которые может выполнять навык, его входных и выходных параметров, а также конечных точек навыка.
  - Разработчики, у которых нет доступа к исходному коду навыка, могут использовать информацию из этого манифеста для разработки потребителя навыка.
  - Схема манифеста навыка представляет собой файл в формате JSON с описанием схемы манифеста навыков. Текущая версия — [2.1.0](#).
  - Примеры манифестов навыка можно найти в статьях о том, как [реализовать навык](#), [написать манифест навыка версии 2.1](#) и [написать манифест навыка версии 2.0](#).

Иными словами, пользователь напрямую взаимодействует только с корневым роботом, который делегирует навыку часть логики своей беседы.

Поддержка навыков реализована так, чтобы обеспечивать следующие возможности:

- навыки и потребители взаимодействуют с использованием протоколов HTTP и Bot Framework;
- потребитель навыка может использовать несколько навыков;
- потребитель навыка может использовать навык, созданный на любом языке. Например, бот на C# может использовать навык, реализованный с помощью Python;
- навык может сам выполнять роль потребителя навыков, вызывая другие навыки;
- навыки поддерживают проверку подлинности пользователей, но она выполняется локально в навыке и не может быть передана в другой бот;
- навыки могут работать с адаптером Bot Framework и пользовательскими адаптерами;

На этой схеме показаны некоторые из возможных сочетаний.

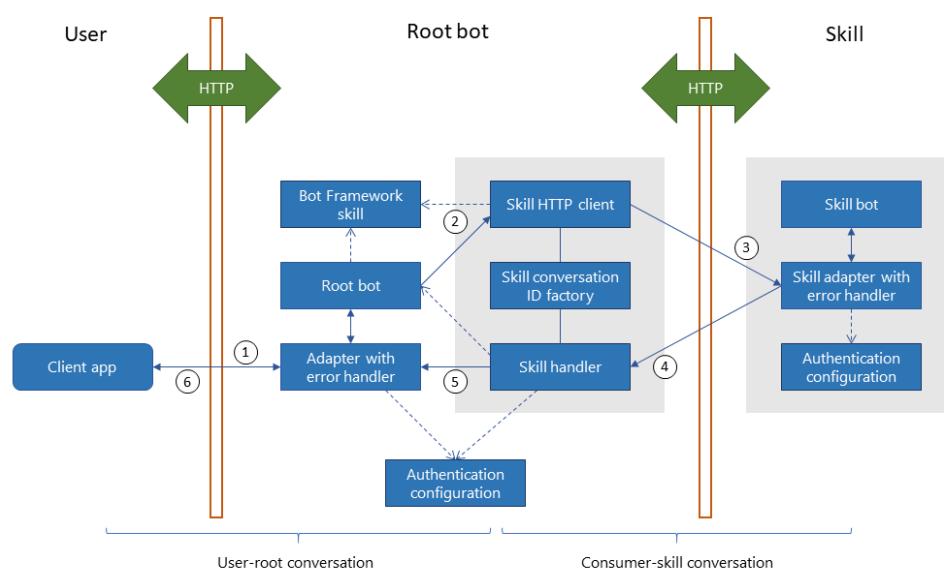


# Концепция архитектуры

Навык и его потребитель размещаются в отдельных ботах и публикуются независимо друг от друга.

- Потребитель опыта требует добавления логики для управления навыками, например, когда следует вызывать или отменять навык и т. д. Кроме обычных объектов бота и адаптера, потребитель включает несколько связанных с навыками объектов, которые используются для обмена действиями с навыком. Потребитель навыков реализует по крайней мере две конечные точки HTTP:
  - Конечная точка обмена сообщениями получает действия от пользователя или канала. Это обычная конечная точка обмена сообщениями, реализуемая всеми программы-роботами.
  - Конечная точка узла навыков для получения действий от навыка. Это действует как URL-адрес обратного вызова, URL-адрес службы, на который отвечают навыки. (Потребитель квалификации должен связать код, который получает запрос HTTP-метода от навыка с помощью обработчика навыков.)
- Навыку требуется дополнительная логика для отправки `endOfConversation` действия после ее завершения, чтобы потребитель квалификации знал, когда следует приостановка пересылки действий в навык.

На этой схеме показан поток действий от пользователя через корневой бот до навыка и обратно.



1. Адаптер корневого бота получает действия от пользователя и перенаправляет их обработчику действий корневого бота. (Действия пользователя получаются в конечной точке обмена сообщениями корневой ленты.)
2. Корневой бот использует HTTP-клиент навыка, чтобы отправить действие в навык. Клиент получает информацию о навыках потребителя на основе определения навыка и фабрики ИДЕНТИФИКАТОРов диалога навыков. Сюда относится и URL-адрес службы, который используется навыком для ответа на действие.
3. Адаптер навыка получает действия от потребителя навыка и перенаправляет их обработчику действий навыка. (Действия от потребителя получаются на конечной точке обмена сообщениями Bot.)
4. Когда навык отправляет ответ, обработчик навыка корневого бота получает соответствующее действие. Он получает сведения о беседе между корневым ботом и пользователем из фабрики идентификаторов бесед с навыками. Затем он передает действие в адаптер корневого бота. (Действия из навыка получаются на конечной точке узла "навык" корневой ленты.)
5. Этот адаптер создает упреждающее сообщение самому себе, чтобы возобновить беседу с

пользователем.

6. Адаптер корневого бота отправляет пользователю все сообщения, полученные от навыка.

Следующие объекты помогают управлять навыками и перенаправлять трафик от них:

- *Навык Bot Framework* описывает сведения о маршрутизации для навыка и считывается из файла конфигурации потребителя навыка.
- *HTTP-клиент навыка* отправляет действия в навык.
- *Обработчик навыка* получает действия от навыка.
- *Фабрика идентификаторов диалога для навыков*, преобразует ссылки между беседами пользователя с корневым ботом и корневого бота с навыком.
- Служба Bot Connector реализует проверку подлинности для канала и для взаимодействия между ботами. С помощью объекта *конфигурации проверки подлинности* можно добавить проверку утверждений к потребителю или квалификации, чтобы ограничить доступ к приложениям или пользователям.

В объектах клиента навыка и обработчика навыка используется *фабрика идентификаторов бесед* для преобразования между беседой, которую корневой бот использует для взаимодействия с пользователем, и беседой для взаимодействия корневого бота с навыком.

## Взаимодействие между ботами

Независимо от того, какой бот вы разрабатываете, важно понимать некоторые аспекты этой архитектуры.

### Ссылки на беседы

Для взаимодействия корневого бота с пользователем и навыком используются разные беседы.

*Фабрика идентификаторов бесед* помогает управлять трафиком между потребителем навыка и навыком. Эта фабрика преобразует идентификаторы бесед, которые корневой бот использует с пользователем и навыком. Другими словами, она создает идентификатор беседы между корневым ботом и навыком, и восстанавливает исходный идентификатор беседы корневого бота с пользователем из идентификатора беседы с навыком.

### Координация между серверами

Корневой бот и боты навыков обмениваются данными по протоколу HTTP. Это означает, что действие от навыка может получать не тот экземпляр корневого бота, который отправил начальное действие, т. е. эти два запроса могут обрабатываться на разных серверах.

- Всегда сохраняйте состояние в потребителе навыка, прежде чем перенаправлять действие в навык. Это позволит экземпляру, который получит трафик от навыка, продолжить работу с того места, где ее остановил предыдущий экземпляр перед вызовом навыка.
- Когда обработчик навыка получает действие от навыка, он преобразует его в форму, подходящую для потребителя навыка, и перенаправляет его в адаптер этого потребителя.

### Потребитель и состояние навыка

Навык и его потребитель управляют своими состояниями независимо друг от друга. Впрочем, потребитель создает идентификатор беседы, которую он использует для обмена данными с навыком. Это может повлиять на состояние беседы в навыке.

#### **IMPORTANT**

Как отмечалось ранее, когда потребитель навыка передает в навык действие пользователя, ответ от этого навыка может получить другой экземпляр потребителя. Потребителю следует всегда сохранять состояние беседы непосредственно перед тем, как перенаправить действие в навык.

### **Аутентификация взаимодействия между ботами**

Начиная с версии 4,11 вам не нужно использовать идентификатор и пароль приложения для проверки уровня квалификации и квалификации потребителя локально в эмуляторе. Подписка Azure по-прежнему необходима для развертывания вашего навыка в Azure.

Проверка подлинности на уровне службы выполняется службой Bot Connector. Эта платформа использует токены носителя и идентификаторы приложения бота для идентификации каждого бота. (Для проверки заголовка проверки подлинности во входящих запросах платформа Bot использует объект *конфигурации проверки подлинности*.)

#### **IMPORTANT**

Для этого требуется, чтобы все развернутые программы-роботы (потребитель опыта и все навыки, используемые им) имели действительные учетные данные приложения.

### **Проверка утверждений**

Необходимо добавить *проверяющий элемент управления утверждения* в конфигурацию проверки подлинности. Утверждения оцениваются после заголовка проверки подлинности. Код проверки должен создавать исключение или ошибку, чтобы отклонить запрос.

#### **NOTE**

Bot выполняет проверку утверждений, если у нее есть идентификатор приложения и пароль. В противном случае проверка утверждений не выполняется.

Отклонение запроса, уже прошедшего проверку подлинности, может потребоваться по разным причинам:

- потребитель навыка должен принимать трафик только от навыков, с которыми он сам начал беседу;
- навык входит в состав платной службы и к нему не должны иметь доступ пользователи, отсутствующие в базе данных;
- вы хотите ограничить доступ к навыку для конкретных пользователей.

#### **IMPORTANT**

Если вы не предложите проверяющий элемент управления утверждениями, программа Bot создаст ошибку или исключение при получении действия из другой программы-робота, будь то навык или потребитель опыта.

## **Дополнительные сведения**

С точки зрения пользователя взаимодействие выполняется только с корневым ботом. С точки зрения навыка потребителем является канал, по которому он обменивается данными с пользователем.

- Дополнительные сведения о ботах и манифестах навыков см. в [этой статье](#).
- Дополнительные сведения о потребителях навыков см. в [этой статье](#).

# Сведения о ботах навыка

27.03.2021 • 2 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

*Навыком* здесь называется бот, который может выполнять ряд задач для другого бота. Навык может использовать другой бот, что упрощает повторное использование. Таким образом вы можете создать бот, ориентированный на пользователя, и расширить его, используя собственные или сторонние навыки. Любой бот с незначительными изменениями может выполнить роль навыка.

Боты навыка выполняют следующее:

- реализуют ограничения доступа через средство проверки утверждений;
- если это требуется, проверяют параметры инициализации в свойстве `value` исходного действия;
- отправляют сообщения пользователю обычным образом;
- сообщают о завершении или отмене выполнения через действие `endOfConversation` ;
  - предоставляют возвращаемое значение, если оно есть, в свойстве `value` отправляемого действия;
  - предоставляют код ошибки, если это применимо, в свойстве `code` отправляемого действия.

Дополнительные сведения см. в разделах [Общие сведения о навыках](#) и [Сведения о потребителях навыков](#).

## Действия навыка

Некоторые навыки могут выполнять разные задачи или *действия*. Например, навык "список задач" может поддерживать действия создания, обновления, просмотра и удаления, которые можно рассматривать как отдельные беседы.

- Узнайте, как [реализовать простой навык](#) с одним действием.
- Узнайте, как [использовать диалоги в навыке](#), который использует диалоги для реализации нескольких действий.

## Манифести навыков

*Манифест навыка* — это файл в формате JSON с описанием действий, которые может выполнять навык, его входных и выходных параметров, а также конечных точек навыка. При использовании версии 2.1 схемы манифеста навыка манифест может также описывать упреждающие действия, которые может отправлять навык, и используемые этим навыком модели отправки.

Сведения о схеме манифеста навыка см. в разделах [Создание манифеста навыков на основе схемы версии 2.1](#) и [Создание манифеста навыков на основе схемы версии 2.0](#).

# Сведения о потребителях навыков

27.03.2021 • 9 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

*Потребителем навыка* называется бот, который умеет вызывать один или несколько навыков. В контексте использования *корневой бот* — это потребитель навыка, который также взаимодействует с пользователем.

С точки зрения пользователя взаимодействие выполняется только с корневым ботом. С точки зрения навыка потребителем является канал, по которому он обменивается данными с пользователем.

(Дополнительные сведения см. в статье с [обзором навыков](#) и [сведениями о ботах навыков](#).)

Корневой бот выполняет роль потребителя навыка, и для этого ему нужна дополнительная логика управления трафиком между собой и навыком:

- Сведения о конфигурации для каждого навыка, который используется корневым ботом.
- *Фабрика идентификаторов бесед*, которая позволяет корневому боту переключаться между беседами, которые он поддерживает с пользователем и с навыком.
- *Клиент навыка*, который может упаковывать действия и переадресовывать их боту навыка.
- *Обработчик навыка*, который может принимать действия от бота навыка и распаковывать их.

## Управление навыками

Для запуска навыка и одного выполнения до завершения работы нужны лишь минимальные дополнения к потребителю навыков. Поддерживаются и более сложные сценарии, например с несколькими навыками или потоками бесед.

Потребитель навыков реализует по крайней мере две конечные точки HTTP:

- *Конечная точка обмена сообщениями* получает действия от пользователя или канала. Это обычная конечная точка обмена сообщениями, реализуемая всеми программы-роботы.
- *Конечная точка узла навыков* для получения действий от навыка. Это действует как URL-адрес обратного вызова, URL-адрес службы, на который отвечают навыки. (Потребитель квалификации должен связать код, который получает запрос HTTP-метода от навыка с помощью обработчика навыков.)

### Описания навыков

Для каждого навыка добавьте объект *навыка Bot Framework* в файл конфигурации потребителя навыка. Каждый из них содержит идентификатор, идентификатор приложения и конечную точку для навыка.

СВОЙСТВО	ОПИСАНИЕ
<i>Идентификатор</i>	Идентификатор или ключ навыка, относящийся к конкретному потребителю навыка.
<i>Идентификатор приложения</i>	Идентификатор <code>appId</code> , назначенный ресурсу бота при регистрации навыка в Azure.

СВОЙСТВО	ОПИСАНИЕ
<i>Конечная точка навыка</i>	Конечная точка обмена сообщениями для навыка. Это URL-адрес, с которым будет взаимодействовать потребитель для работы с навыком.

### Клиент навыка и обработчик навыка

Потребитель навыка использует клиент навыка для отправки действий в навык. Клиент выполняет следующее:

- принимает полученное от пользователя или созданное потребителем действие, которое нужно отправить в навык;
- Задает URL-адрес службы для действия, которое отсылается навыку конечной точке узла навыка потребителя.
- заменяет исходную ссылку на беседу другой, которая соответствует беседе между потребителем и навыком;
- добавляет токен проверки подлинности между ботами;
- отправляет обновленное действие в навык.

Потребитель навыка использует обработчик навыка для получения действий от навыка. Этот обработчик выполняет следующее:

- обрабатывает методы REST API службы канала;
- применяет проверку подлинности и проверку утверждений;
- получает ссылку на исходную беседу;
- создает действие для адаптера потребителя. Это действие подает сигнал о завершении навыка или передается пользователю.

## Непосредственное управление навыком

Вам нужно добавить в потребитель навыков логику, которая отслеживает активные навыки. Именно от потребителя зависит, как выполняется управление навыками в целом, поддерживается ли параллельная работа нескольких активных навыков и т. д. Следует оценить несколько конкретных сценариев:

- Создание новой беседы между потребителем и навыком. (Она будет связана с конкретной беседой между потребителем и пользователем.)
  - Чтобы передать параметры навыку, установите значение свойства *value* в исходном действии, которое передается этому навыку.
- Продолжение существующей беседы между потребителем и навыком.
- Распознавание действия `endOfConversation`, полученного от навыка, как сигнала для окончания беседы между потребителем и навыком.
  - Чтобы получить значения, возвращаемые из навыка, проверьте свойство *value* действия.
  - Чтобы узнать причину завершения навыка, проверьте параметр *code*, который может сообщить об ошибке, возникшей в навыке.
- Отмена навыка из потребителя путем отправки действия `endOfConversation` в навык.

В статье [о реализации потребителя навыка](#) представлен пример потребителя, который напрямую управляет навыком.

## Управление навыком через диалог навыка

Если вы используете [библиотеку диалогов](#), для управления навыком можно использовать [диалог навыка](#). Активным диалогом будет считаться диалог навыка, но все действия он будет пересыпать связанному

навыку.

- При создании диалога навыка используйте параметр *dialog options*, чтобы предоставить всю информацию, которая потребуется этому диалогу для управления навыком, в том числе идентификатор приложения потребителя, URL-адрес обратного вызова, фабрика идентификаторов беседы, свойства навыка и т. п.
  - Если вам нужно управлять в диалоге несколькими навыками, следует создать отдельный диалог для каждого навыка.
  - Часто вы будете добавлять диалог навыка в компонентный диалог.
- Чтобы открыть диалог навыка, вызовите метод *begin* из контекста диалога окна и укажите идентификатор диалога навыка. С помощью параметра *options* укажите действие, которое потребитель будет передавать в качестве первого действия для навыка.
- Диалог навыка можно отменить или прервать, как любой другой диалог. Пример см. в статье [Обработка прерываний со стороны пользователя](#).

Пример потребителя, который управляет навыком через диалог навыка, вы найдете [в этой статье](#).

## Использование режима доставки «ожидание ответов»

Программы-роботы и навыки используют стандартные отраслевые возможности и JSON по протоколу HTTPS для обмена данными. Поток обработки обычных действий запускается, когда корневой элемент Bot получает запись из канала в *конечной точке обмена сообщениями*. Затем корневой элемент Bot отправляет действие на уровень для обработки. Ответы от навыка отправляются обратно в *конечную точку основного узла*, а не в конечную точку обмена сообщениями. Наконец, ответы обрабатываются дополнительно или отправляются обратно в канал с помощью корневого робота. Этот нормальный поток можно изменить, изменив *режим доставки* действия, отправляемого в навык. Если для *режима доставки* задано значение "спектреплиес", то навык не будет относиться к конечной точке узла навыка. Вместо этого все действия ответа сериализуются в текст ответа. Затем корневой элемент Bot выполняет перебор этих действий, обрабатывая их аналогично тому, как они были бы обработаны конечной точкой узла навыка.

Дополнительные сведения см. в описании [режима доставки](#) в спецификации действия.

# Адаптивные выражения

27.03.2021 • 8 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Программы-роботы используйте адаптивные выражения для вычисления результата условия на основе сведений о времени выполнения, доступных в памяти для диалогового окна или системы [формирования языка](#). Эти вычисления позволяют выбрать реакцию бота на поступившие от пользователя данные с учетом других факторов, которые могут влиять на его функционирование.

Адаптивные выражения предназначены для этой основной потребности, предоставляя адаптивный язык выражений, который можно использовать с пакетом SDK для Bot Framework и другими компонентами искусственного интеллекта, такими как [Bot Framework Composer](#), [Создание языка](#), [адаптивные диалоговые окна](#) и [шаблоны адаптивных карточек](#).

Адаптивное выражение может содержать одно или несколько значений, [встроенных функций](#) или настраиваемых функций. Получателям адаптивных выражений также доступна возможность включать в них дополнительные поддерживаемые функции. Например, все шаблоны создания языка доступны как функции, а также дополнительные функции, доступные только в пределах использования адаптивных выражений в этом компоненте.

## Операторы

Адаптивные выражения поддерживают следующие типы операторов и синтаксис выражений.

- арифметический
  - сравнение
  - логические
  - другие операторы и синтаксис выражений
- 
- [Арифметические](#)
  - [Сравнение](#)
  - [Логическое](#)
  - [Другое](#)

ОПЕРАТОР	ФУНКЦИОНАЛЬНОСТЬ	ЭКВИВАЛЕНТ ВСТРОЕННОЙ ФУНКЦИИ
+	Сложение. Пример A + B	<a href="#">добавление</a>
-	Вычитание. Пример: A-B	<a href="#">sub</a>
Унарный +	Положительное значение. Пример: +1, +A	Недоступно
Унарный -	Отрицательное значение. Пример:- 2,-B	Недоступно
*	Умножение. Пример: A * B	<a href="#">mul</a>
/	Деление. Пример A / B	<a href="#">div</a>

ОПЕРАТОР	ФУНКЦИОНАЛЬНОСТЬ	ЭКВИВАЛЕНТ ВСТРОЕННОЙ ФУНКЦИИ
<code>^</code>	Возведение в степень. Пример <code>A ^ B</code>	<code>exp</code>
<code>%</code>	Вычисление модуля. Пример <code>A % B</code>	<code>mod (модуль)</code>

## Переменные

Ссылки на переменные всегда содержат их имя в формате  `${myVariable}` . На них можно ссылаться с помощью оператора выбора свойства вида  `myParent.myVariable` , оператора выбора индекса элемента (например,  `myParent myList[0]` ) или с помощью функции  `getProperty()` .

Существуют две специальные переменные. `[]` представляет пустой список, а `{}` представляет пустой объект.

## Явные значения

Явные значения могут быть заключены в одинарные кавычки 'myExplicitValue' или двойные кавычки "myExplicitValue".

## Функции

Адаптивное выражение имеет одну или несколько функций. Дополнительные сведения о функциях, поддерживаемых адаптивными выражениями, см. в справочной статье о [встроенных функциях](#).

## Конструктор Bot Framework

Составитель Framework Composer — это визуальный холст для разработки с открытым кодом, позволяющий разработчикам и эксплуатирующей группам создавать программы-роботы. Composer использует адаптивные выражения для создания, вычисления и изменения значений. Адаптивные выражения можно использовать в определениях шаблонов создания языка и в качестве свойств на холсте разработки. Как показано в примере ниже, свойства в памяти можно также использовать в адаптивном выражении.

Выражение `(dialog.orderTotal + dialog.orderTax) > 50` добавляет значения свойств и и вычисляет значение  `dialog.orderTotal | dialog.orderTax , True` . Если сумма превышает 50 или  `False` . Если сумма меньше 50.

Дополнительные сведения о том, как выражения используются в памяти, см. в статье [поток диалога и память](#).

## Создание речи

Адаптивные выражения используются системами формирования языка (LG) для вычисления условий, описанных в статье шаблоны LG. В приведенном ниже примере предварительно подготовленная функция `Join` используется для перечисления всех значений в  `recentTasks`  коллекции.

```
# RecentTasks
- IF: ${count(recentTasks) == 1}
  - Your most recent task is ${recentTasks[0]}. You can let me know if you want to add or complete a task.
- ELSEIF: ${count(recentTasks) == 2}
  - Your most recent tasks are ${join(recentTasks, ', ', ' and ')}. You can let me know if you want to add or complete a task.
- ELSEIF: ${count(recentTasks) > 2}
  - Your most recent ${count(recentTasks)} tasks are ${join(recentTasks, ', ', ' and ')}. You can let me know if you want to add or complete a task.
- ELSE:
  - You don't have any tasks.
```

Дополнительные сведения см. в разделе "использование предварительно [подготовленной функции](#)" статьи [LG File Format](#).

## Шаблоны адаптивных карточек

[Шаблоны адаптивных карт](#) могут использоваться разработчиками программы-роботы и других технологий для разделения данных из макета на адаптивной карте. Разработчики могут предоставлять [данные в виде встроенных данных AdaptiveCard](#) или более распространенный подход к [разделению данных из шаблона](#).

Например, предположим, что у вас есть следующие данные:

```
{
  "id": "1291525457129548",
  "status": 4,
  "author": "Matt Hidinger",
  "message": "
    \"type\": \"Deployment\",
    \"buildId\": \"9542982\",
    \"releaseId\": \"129\",
    \"buildNumber\": \"20180504.3\",
    \"releaseName\": \"Release-104\",
    \"repoProvider\": \"GitHub\"
  ",
  "start_time": "2018-05-04T18:05:33.3087147Z",
  "end_time": "2018-05-04T18:05:33.3087147Z"
}
```

[message](#) Свойство является строкой, сериализованной в формате JSON. Чтобы получить доступ к значениям в строке, можно вызвать предварительно созданную функцию [JSON](#):

```
{
  "type": "TextBlock",
  "text": "${json(message).releaseName}"
}
```

И приведет к следующему объекту:

```
{
  "type": "TextBlock",
  "text": "Release-104"
}
```

Дополнительные сведения и примеры см. в [документации по шаблонам адаптивных карт](#).

## Дополнительные ресурсы

- [Пакет NuGet AdaptiveExpressions](#) для C#.
- [пакет NPM с адаптивными выражениями](#) для JavaScript

- [Встроенные функции](#), поддерживаемые библиотекой адаптивных выражений
- [Справочник по API для C#](#)
- [Справочник по API JavaScript](#)

# Создание текста

27.03.2021 • 4 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Создание текста позволяет разработчикам извлекать внедренные строки из файлов ресурсов и кода, а также управлять ими, используя соответствующие среду выполнения и формат файла. С помощью Создания текста разработчики могут создавать более естественные диалоги, определяя несколько вариантов фраз, выполняя простые выражения на основе контекста и обращаясь к сохраненным данным беседы.

С помощью Создания текста разработчики могут:

- создавать характерные особенности и интонации для бота;
- отделять бизнес-логику от представления;
- добавлять вариации и расширенное разрешение на основе композиции во все ответы бота;
- создавать адаптированное сопоставление речи и отображения;
- создавать карточки, предлагаемые действия и вложения.

В основе Создания текста лежат возможности расширения шаблона и подстановки сущностей. Вы можете предоставить один вариант для расширения, а также расширить шаблон на основе условий. Выходными данными Создания текста может быть простая текстовая строка, многострочный ответ или полезная нагрузка сложного объекта, которые слой вышестоящий слой будет использовать для создания **действия**.

Ниже приведен простой шаблон Создания текста для приветствия. Обратите внимание, что в приветствии используются ссылки на имя пользователя, размещенное в памяти в переменной  `${user.name}`  .

```
# greetingTemplate
- Hello ${user.name}, how are you?
- Good morning ${user.name}. It's nice to see you again.
- Good day ${user.name}. What can I do for you today?
```

## Применение Создания текста

Вы можете применять Создание текста при разработке ботов разными способами. Для начала создайте один или несколько файлов [.lg](#), чтобы охватить все возможные сценарии, в которых вы можете использовать подсистему создания языка с ответами бота пользователю.

- [C#](#)
- [JavaScript](#)

Включите библиотеку Создания текста `Microsoft.Bot.Builder.LanguageGeneration`. Затем проанализируйте и загрузите шаблоны в файл [.lg](#), добавив следующий код:

```
_templates = Templates.ParseFile(fullPath);
```

Если требуется расширение шаблона, примените `Evaluate` и передайте имя соответствующего шаблона.

- [C#](#)

- [JavaScript](#)

```
var lgOutput = _templates.Evaluate(<TemplateName>);
```

Если этому шаблону нужно передать определенные свойства для разрешения или расширения, их можно включить в вызов `Evaluate`.

- [C#](#)
- [JavaScript](#)

```
var lgOutput = _templates.Evaluate("WordGameReply", new { GameName = "MarcoPolo" } );
```

## Многоязыковая политика создания и переключения языка

Вы можете создать бот, который работает с текстом на нескольких языках — произносимом или отображаемом. В таких случаях следует использовать отдельные экземпляры `TemplateEngine` для каждого целевого языка. Чтобы узнать, как добавить в бот несколько языков (т. н. смену языка), ознакомьтесь с примером многоэтапного диалога со сменой языка на [C#](#) или [JavaScript](#).

## API расширения

Чтобы получить все возможные расширения шаблона, можно использовать `ExpandTemplate`.

- [C#](#)
- [JavaScript](#)

```
var results = lgTemplates.ExpandTemplate("WordGameReply", { GameName = "MarcoPolo" } )
```

Пример с содержимым Создания текста:

```
# Greeting
- Hi
- Hello

#TimeOfDay
- Morning
- Evening

# FinalGreeting
- ${Greeting()} ${TimeOfDay()}

# TimeOfDayWithCondition
- IF: ${time == 'morning'}
  - ${Greeting()} Morning
- ELSEIF: ${time == 'evening'}
  - ${Greeting()} Evening
- ELSE:
  - ${Greeting()} Afternoon
```

Вызов `ExpandTemplate("FinalGreeting")` возвращает четыре варианта:

- Hi Morning
- Hi Evening
- Hello Morning
- Hello Evening

Вызов `ExpandTemplate("TimeOfDayWithCondition", new { time = "evening" })` с указанием области действия возвращает два расширения:

- Hi Evening
- Hello Evening

## Дополнительные ресурсы

- См. сведения о [файлах .lg](#).
- См. сведения о [структурированных шаблонах ответов](#).
- [Справочник по API для C#](#)
- [Справочник по API JavaScript](#)

# Принципы разработки ботов

27.03.2021 • 6 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Bot Framework позволяет разработчикам создавать востребованные боты, которые решают различные бизнес-проблемы. После изучения основных понятий, описанных в этом разделе, вы будете иметь широкие возможности для разработки бота, который соответствует рекомендациям, и приобретенный опыт пригодится вам для продвижения в этой относительно новой области.

## Разработка бота

Если вы создаете бот, можно с уверенностью предположить, что вы ожидаете его востребованности у пользователей. Можно также предположить, что вы надеетесь на предпочтение вашего бота таким альтернативам, как приложения, веб-сайты, телефонные звонки и другие средства удовлетворения особых потребностей пользователей. Другими словами, бот конкурирует с приложениями и веб-сайтами за время пользователей. Итак, как вы можете повысить шанс того, что бот выполнит свою основную задачу — привлечение и удержание пользователей? Нужно всего лишь расставить приоритеты на правильные факторы при разработке бота.

## Факторы, которые не гарантируют успешность бота

При разработке бота следует учитывать, что ни один из следующих факторов не гарантирует его успешность:

- **Как работает Интеллектуальный робот:** в большинстве случаев маловероятно, чтобы ваш Bot был уверен, что вы сможете с радостью проделать пользователей или внедрять свою платформу. На самом деле многие боты не сильно продвинуты в машинном обучении или в возможностях естественного языка. Бот может обладать этими возможностями, если они необходимы для решения проблем, для которых его разрабатывали, однако не следует предполагать, что существует некая корреляция между интеллектуальными возможностями бота и переходом пользователей на взаимодействие с ним.
- **Число поддерживаемых естественных языков бота.** Ваш бот может отлично справляться с диалогами. Он может иметь огромный словарный запас и даже использовать отличные шутки. Но если он не решает проблемы пользователей, то эти возможности мало повлияют на успешность бота. На самом деле у некоторых ботов вообще отсутствует разговорная возможность. И во многих случаях это вполне нормально.
- **Voice:** это не всегда так, что включение программы-роботы для речи может привести к отличному интерфейсу пользователя. Если пользователи вынуждены пользоваться голосовой связью, это может привести их к раздражению в процессе взаимодействия. При проектировании бота всегда думайте о том, подходит ли голосовая связь в качестве канала для решения конкретной задачи. Будет ли окружающая среда шумной? Будет ли голосовая связь передавать информацию для совместного использования с пользователем?

## Факторы, действительно влияющие на успех бота

Наиболее успешные приложения или веб-сайты имеют по крайней мере одну общую вещь — отличное взаимодействие с пользователем. В этом плане боты ничем не отличаются. Таким образом обеспечение отличного взаимодействия с пользователем должно быть приоритетом номер один при разработке

бота. Основные рекомендации:

- Легко ли программа-робота решить проблему пользователя с минимальным количеством шагов?
- Разрешает ли программа-робота проблему пользователя лучше, проще или быстрее, чем любые другие возможности?
- Запускается ли бот на устройствах и платформах, которые интересуют пользователя?
- Доступен ли бот? Имеет ли бот интуитивно понятное взаимодействие с пользователями?

Ни один из этих вопросов прямо не связан с такими факторами, как интеллектуальность бота, числом поддерживаемых естественных языков, использованием машинного обучения или языком, использованным для написания бота. Пользователи вряд ли заинтересовались бы этими факторами, если бот решает необходимую проблему и обеспечивает отличное взаимодействие. Отличное взаимодействие бота с пользователем не требует от пользователя много печатать, слишком много говорить, повторять сказанное несколько раз или объяснять вещи, которые бот должен знать по умолчанию.

#### TIP

Независимо от типа приложения, которое вы создаете (бот, веб-сайт или приложение), сделайте взаимодействие с пользователем главным приоритетом.

Процесс разработки бота подобен процессу разработки приложения или веб-сайта, поэтому опыт, накопленный десятилетиями создания пользовательского интерфейса и UX для приложений и веб-сайтов, по-прежнему актуален при проектировании ботов.

Каждый раз, когда вы не уверены в правильности подхода к разработке бота, вернитесь на шаг назад и задайте себе следующий вопрос: "Как следовало бы решить эту проблему в приложении или на веб-сайте?" Скорее всего, тот же ответ применим и для разработки ботов.

## Дальнейшие действия

Теперь, когда вы ознакомились с основными принципами разработки бота, узнайте о том, как [разработать процесс взаимодействия](#) пользователя с ботом.

# Разработка первого взаимодействия бота с пользователем

27.03.2021 • 3 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

## Важность первого впечатления

Первое взаимодействие между пользователем и ботом имеет решающее значение. При разработке бота помните, что первое сообщение — это не просто приветствие. При разработке приложения вы размещаете на первом экране важные подсказки по [навигации](#). Пользователи должны интуитивно понимать, где находится меню и как оно работает, куда обратиться за помощью, что говорится в политике конфиденциальности и т. д. При разработке бота вы должны помнить, что при первом взаимодействии пользователь должен получать такую же информацию.

## Язык и меню

Рассмотрите два следующих варианта:

### Вариант 1



Hello user, how can I help you?

### Вариант 2



Hello! How can I help you?

Orders

Products

Help

Начинать разговор с вопросов, вроде "Чем я могу вам помочь?", обычно не рекомендуется. Если у программы-робота есть сотни различных вещей, которые он может сделать, скорее всего, пользователи не смогут угадать большинство из них. Ваш робот не дал им того, что это возможно, и как они могут быть уверены?

Меню станет простым решением этой проблемы. Во-первых, перечислив доступные варианты, ваш бот сообщит пользователю о своих возможностях. Во-вторых, при наличии меню пользователю не придется вводить много текста — достаточно будет нажать на нужный вариант. Наконец, меню может значительно упростить модели естественного языка, сузив варианты вводной информации от пользователя.

**TIP**

Меню — это ценное средство при проектировании программы-роботы для удобного взаимодействия с пользователем. Не отменяйте их как неразумные. Вы можете настроить меню и при этом сохранить ввод произвольного текста. Если пользователь видит исходное меню и вводит текст, а не выбирает вариант, бот попытается проанализировать этот текст.

Кроме того, вы можете задавать конкретные вопросы, чтобы направить пользователя, если бот имеет определенную функцию. Например, если бот принимает заказы на сандвичи, сначала он скажет: "Здравствуйте! Я готов принять ваш заказ. Какой хлеб вы предпочитаете? У нас есть белый, цельнозерновой и ржаной". Так пользователь понимает, что ответить, и следует подсказкам.

## Другие замечания

Помимо интуитивно понятного и простого первого взаимодействия, хороший бот предоставляет пользователю доступ к сведениям о политике конфиденциальности и условиям использования.

**TIP**

Если ваш бот собирает персональные данные пользователя, важно сообщить ему об этом и рассказать, как будут обрабатываться эти данные.

## Дальнейшие действия

Теперь, когда вы ознакомились с основными принципами создания первого взаимодействия пользователя с ботом, узнайте больше о том, как [разработать дальний диалог](#).

# Проектирование потока беседы и управление им

27.03.2021 • 8 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В обычном приложении пользовательский интерфейс состоит из серии экранов, и одно приложение или веб-сайт могут использовать один или несколько экранов по мере необходимости для обмена информацией с пользователем. В большинстве приложений сначала открывается основной экран, который предоставляет средства навигации, помогающие перейти к другим экранам для выполнения различных функций, таких как запуск нового заказа, просмотр продуктов или получение помощи.

Как у приложений и веб-сайтов, у ботов есть пользовательский интерфейс, но он состоит из *сообщений*, а не экранов. Сообщения могут содержать кнопки, текст и другие элементы или быть полностью речевыми.

Хотя обычное приложение или веб-сайт могут запрашивать сразу несколько фрагментов информации на экране, бот будет собирать одинаковый объем информации с помощью нескольких сообщений. Таким образом, процесс сбора информации от пользователя является активным взаимодействием, при котором пользователь ведет активную беседу с ботом.

Хорошо спроектированная программа-робот будет иметь естественный поток. Программа-робот должна иметь возможность легко обрабатывать основной диалог и иметь возможность корректно обрабатывать прерывания или переходить к разделам.

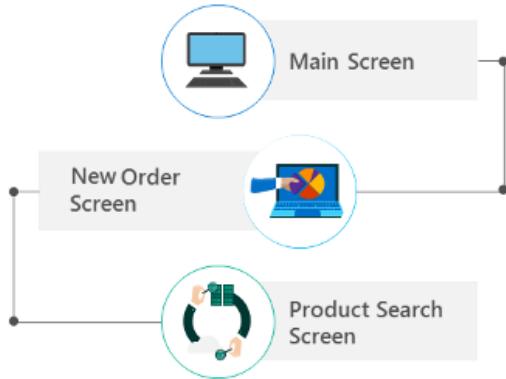
## Процедурный поток беседы

Диалоги с Bot обычно сосредоточены на задаче, которую пытается выполнить Bot, что называется процедурным потоком. Именно здесь бот задает пользователю ряд вопросов, чтобы собрать всю необходимую информацию перед обработкой задачи.

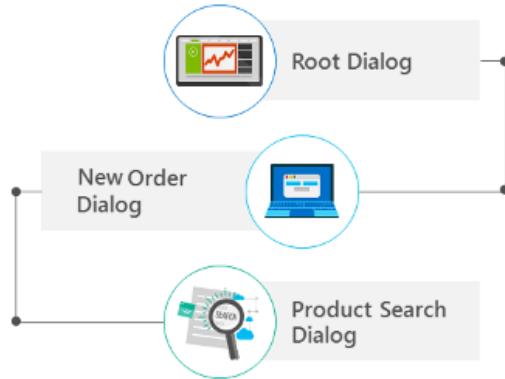
В процедурном потоке беседы вы определяете порядок вопросов, и бот будет задавать вопросы в указанном вами порядке. Вы можете упорядочить вопросы по логическим группам, чтобы не усложнять код, а управлять им. Например, вы можете создать один модуль, содержащий логику, которая помогает пользователю просматривать продукты, и отдельный модуль, содержащий логику, которая помогает пользователю создать новый заказ.

Вы можете структурировать эти модули любым способом, начиная от произвольной формы и заканчивая последовательной. Пакет SDK для Bot Framework предоставляет библиотеку диалогов, которая позволяет создавать любые диалоговые потоки, необходимые для работы программы Bot. Библиотека содержит [каскадные диалоговые окна](#) для создания последовательности шагов и запросы на вопросы пользователей. Дополнительные сведения см. в разделе [Библиотека диалогов](#).

## TRADITIONAL APPLICATION



## BOT



В традиционном приложении все начинается с *главного экрана*. На главном экране вызывается *Новый экран заказа*. Новый экран заказа остается в элементе управления до тех пор, пока он не закрывается или вызывает другие экраны, например экран *поиска продукта*. Если экран нового заказа закрывается, пользователь возвращается на основной экран.

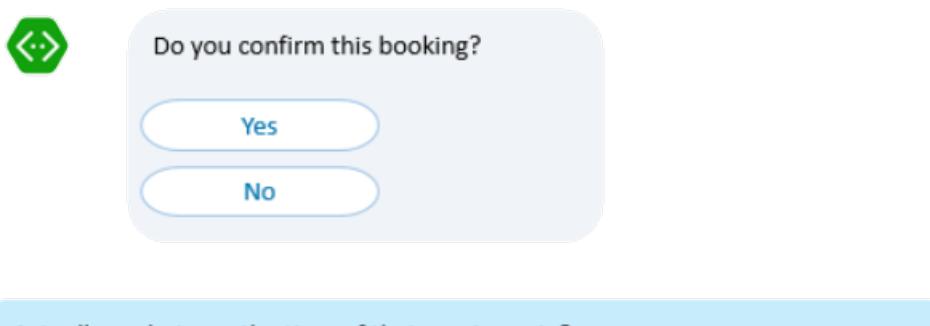
В роботе, использующем диалоговые окна, все начинается с *корневого диалогового окна*. Корневое диалоговое окно вызывает *диалоговое окно нового заказа*. В этот момент диалоговое окно Новый порядок берет на себя управление диалогом и остается в контроле до тех пор, пока не закроет или не вызовет другое диалоговое окно, например *диалоговое окно поиска продукта*. Если диалоговое окно Новый порядок закрывается, управление диалогом возвращается к корневому диалоговому окну.

Пример реализации потока диалога с помощью библиотек диалоговых окон см. в разделе [реализация последовательного потока диалога](#).

## Обработка прерываний

Было бы заманчиво предположить, что пользователи будут выполнять процедурные задачи по очереди упорядоченным образом. Например, в потоковой беседе с помощью диалоговых окон пользователь начнет работу в корневом диалоговом окне и вызовет диалоговое окно Новый порядок. В диалоговом окне Новый порядок они вызывают диалоговое окно Поиск продукта. Затем при выборе одного из результатов, перечисленных в диалоговом окне Поиск продукта, они вызывают диалоговое окно Новый порядок. После завершения заказа они снова поступают в корневое диалоговое окно.

Было бы хорошо, если бы пользователи всегда выполняли переход по линейному логическому пути, но это происходит редко. Пользователи не всегда взаимодействуют в последовательном порядке. Они часто меняют свое мнение. Рассмотрим следующий пример:



Пока бот будет выполнять процедуру, пользователь может решить сделать что-то совершенно другое

или задать вопрос, который не связан с текущей темой. В примере выше пользователь спрашивает, а не предоставляет ответ "Да" или "Нет", который ожидает бот. Как должен отвечать бот?

- Пользователю сначала нужно ответить на вопрос.
- Не обращать внимание на все, что сделал пользователь ранее, сбросить весь стек диалоговых окон и начать с самого начала, пытаясь ответить на вопрос пользователя.
- Попытаться ответить на вопрос пользователя, а затем вернуться к вопросу с ответом "Да" или "Нет" и продолжить работу с того момента.

Нет единственного *правильного* ответа на этот вопрос, лучшее решение будет зависеть от особенностей сценария и ожидания пользователя на ответ от бота. Узнайте, как [управлять прерываниями пользователей](#) для роботов, предназначенных для обработки некоторых типов прерываний.

## Завершение срока действия диалога

Иногда бывает полезно перезапустить диалог с самого начала. Например, если пользователь не отвечает по истечении определенного периода времени. Ниже приведены различные методы завершения диалога.

- Отследить время последнего получения сообщения от пользователя и очистить состояние, если время больше, чем предварительно настроенная длина при получении следующего сообщения от пользователя.
- Используйте функцию уровня хранилища, например функцию "[срок жизни](#)" Cosmos DB, чтобы очистить состояние после предварительно настроенного промежутка времени.

Дополнительные сведения см. в разделе [Окончание срока действия диалога](#).

## Следующие шаги

Управление навигацией пользователя в диалоговых окнах и проектирование потоковой передачи таким образом, чтобы пользователи могли достичь своих целей (даже в нелинейном режиме), — это фундаментальная задача Bot-дизайна. В [этой статье рассматриваются](#) некоторые распространенные проблемы плохо спроектированной навигации и обсуждаются стратегии предотвращения этих ловушек.

# Разработка навигации для бота

27.03.2021 • 8 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Пользователи могут перемещаться по веб-сайтам с помощью строки навигации, в приложениях с помощью меню и в веб-браузерах с помощью кнопок, таких как **Вперед** и **Назад**. Однако ни одна из этих широко известных навигационных методик полностью не учитывает требования к навигации в рамках бота. Как уже говорилось [ранее](#), пользователи часто взаимодействуют с ботами нелинейным способом, что затрудняет разработку навигации для бота, которая стабильно обеспечивает удобство работы для пользователя.

Рассмотрим следующие дилеммы.

- Как гарантировать, что пользователь не потерпается в диалоге с ботом?
- Может ли пользователь переходить "назад" в диалоге с ботом?
- Как пользователю переходить к "главному меню" на протяжении беседы с ботом?
- Как пользователю "отменить" операцию на протяжении беседы с ботом?

Особенности проектирования навигации для вашего бота будут во многом зависеть от возможностей и функций, которые поддерживает ваш бот. Независимо от типа разрабатываемого бота вы захотите избежать распространенных ошибок плохо разработанных диалоговых интерфейсов. В этой статье описываются эти ошибки с точки зрения пяти персонажей: "упрямого бота", "невежественного бота", "тайного бота", "бота Капитан Очевидность" и "бота, который не может забыть".

## TIP

Минимизировать возможность превращения бота в одного из этих персонажей часто можно с помощью правильной [обработки пользовательских прерываний](#).

## "Упрямый бот"

"Упрямый бот" настаивает на поддержании текущего курса беседы, даже когда пользователь пытается направить разговор в другое русло.

Рассмотрим следующий сценарий.

What city are you travelling to?

Bot

cancel

User

I'm sorry, I didn't understand that. What city are you travelling to?

Bot

help

User

I'm sorry, I didn't understand that. What city are you travelling to?

Bot

STOP

User

I'm sorry, I didn't understand that. What city are you travelling to?

Bot

Please stop, cancel, help, abort, do anything!

Пользователи часто меняют свое мнение, решают отменить или иногда хотят начать все заново.

#### TIP

**Рекомендуется.** Проектировать бота с учетом того, что пользователь может изменить ход беседы в любое время.

**Не рекомендуется.** Проектировать бота, который игнорирует ввод данных пользователем и повторяет тот же вопрос в бесконечном цикле.

Есть много способов избежать этой ошибки, но, возможно, самый простой способ не дать боту задавать один и тот же вопрос бесконечно, просто указав максимальное количество повторных попыток для каждого вопроса. Если он разработан таким образом, бот не делает ничего "умного", чтобы понять введенные пользователем данные и отвечать соответствующим образом, но, по крайней мере, он не задает один и тот же вопрос бесконечно.

## "Невежественный бот"

"Невежественный бот" отвечает бессмысленно, когда не понимает попытки пользователя получить доступ к определенной функции. Пользователь может попытаться использовать ключевые команды, такие как "справка" или "отмена", ожидая, что бот будет реагировать соответствующим образом.

Рассмотрим следующий сценарий.



Please enter the code for the request:

Help?



Thank you, using code 'Help?'. Have a good day.

Huh?

Хотя у вас может возникнуть соблазн спроектировать каждый диалог внутри вашего бота для прослушивания и соответствующего ответа на определенные ключевые слова, этот подход не рекомендуется.

#### TIP

**Рекомендуется.** Реализовать [ПО промежуточного слоя](#), которое анализирует введенные пользователем ключевые слова (например, "справка", "отмена", "начать заново" и т. д.), и реагирует соответствующим образом.

**Не рекомендуется.** Проектировать каждый диалог так, чтобы во введенных пользователем данных выполнялся поиск ключевых слов.

Определив логику в **ПО промежуточного слоя**, вы делаете его доступным для каждого обмена с пользователем. При таком подходе отдельные диалоги и запросы можно спроектировать таким образом, чтобы пропускать ключевые слова, если нужно.

## "Таинственный бот"

"Таинственный бот" не может немедленно подтвердить входные пользовательские данные любым способом.

Рассмотрим следующий сценарий.



How can I help you today?

Which movies are showing this week?

Hello?

Which movies are showing this week?

Hey bot, are you there?

В некоторых случаях эта ситуация может указывать на сбои в работе бота. Однако возможно, что бот просто занят обработкой входящих пользовательских данных и еще не завершил составление своего ответа.

**TIP**

**Рекомендуется.** Проектировать бота, который будет немедленно подтверждать введенные данные, даже когда для составления ответа нужно время.

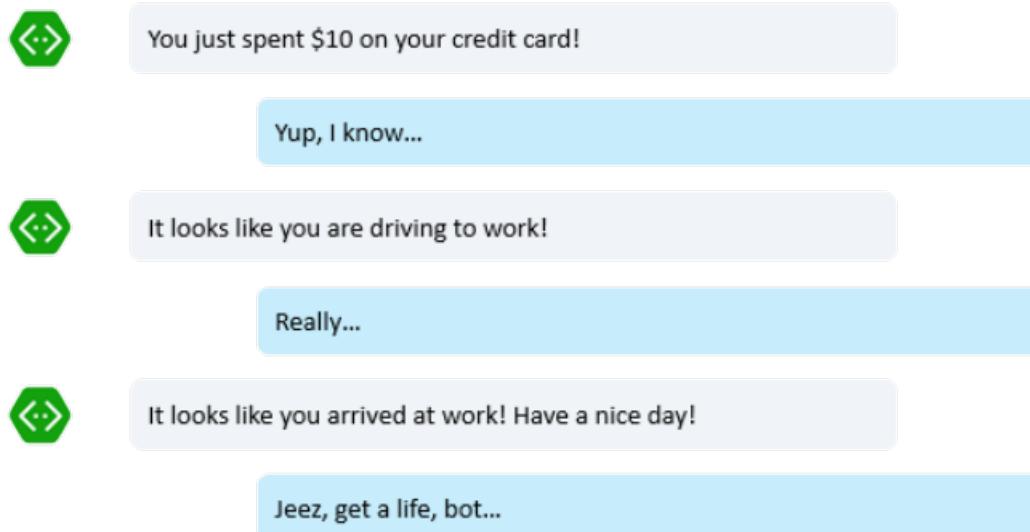
**Не рекомендуется.** Проектировать бота, который будет подтверждать введенные данные только после составления ответа.

Немедленно подтверждая введенные данные, вы устраняете любую возможную путаницу в отношении состояния бота. Если время компиляции отклика занимает много времени, рассмотрите возможность отправки сообщения "Ввод", чтобы указать, что программа-робот работает, а затем задействует [упреждающее сообщение](#).

## "Бот Капитан Очевидность"

"Бот Капитан Очевидность" предоставляет незапрашиваемые сведения, которые совершенно очевидны и поэтому бесполезны для пользователя.

Рассмотрим следующий сценарий.

**TIP**

**Рекомендуется.** Проектировать бота, который будет предоставлять полезную для пользователя информацию.

**Не рекомендуется.** Проектировать бота, который будет предоставлять ненужную (бесполезную для пользователя) информацию.

Создав бот для получения полезных сведений, вы увеличиваете вероятность взаимодействия пользователя с ботом.

## "Бот, который не может забыть"

"Бот, который не может забыть" ошибочно интегрирует информацию из прошлых диалогов с текущей беседой.

Рассмотрим следующий сценарий.

I want to travel to Italy



Please confirm: Are you ok with me charging \$200 for your trip to Las Vegas?

Which Trip to Las Vegas???



Your trip to Las Vegas you wanted me to book for June the 5th

OMG that was 3 months ago, bot...

#### TIP

**Рекомендуется.** Проектировать бота, который будет поддерживать текущую тему беседы, пока пользователь не выразит желание вернуться к предыдущей теме.

**Не рекомендуется.** Проектировать бота, который будет использовать информацию из прошлых бесед тогда, когда это не относится к текущему разговору.

Поддерживая текущую тему разговора, вы уменьшаете вероятность путаницы и недовольства, а также увеличиваете вероятность того, что пользователь будет продолжать взаимодействовать с вашим ботом.

## Дальнейшие действия

При проектировании бота, который избегает этих распространенных ошибок плохо разработанных диалоговых интерфейсов, вы делаете важный шаг на пути к обеспечению отличного пользовательского интерфейса.

Далее, узнайте больше об [элементах пользовательского интерфейса](#), которые боты используют чаще всего для обмена данными с пользователями.

# Проектирование взаимодействия с пользователем

27.03.2021 • 15 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Вы можете создавать боты с разнообразными компонентами, такими как текст, кнопки, изображения и функциональные карточки, отображаемые в карусели или в формате списка, и многое другое. При этом каждый канал, такой как Facebook, Slack и др., определяет, как его клиенты для обмена сообщениями будут обрабатывать эти компоненты. Даже если несколько каналов поддерживают компонент, каждый канал может преобразовать его немного по-разному. В тех случаях, когда сообщение содержит компоненты, которые канал не поддерживает изначально, канал может попытаться упрощенно преобразовать содержимое сообщения в текст или в статическое изображение, что может существенно повлиять на вид сообщения в клиенте. В некоторых случаях канал может вообще не поддерживать определенный компонент. Например, клиенты GroupMe не отображают индикатор ввода.

## Функциональные пользовательские элементы управления

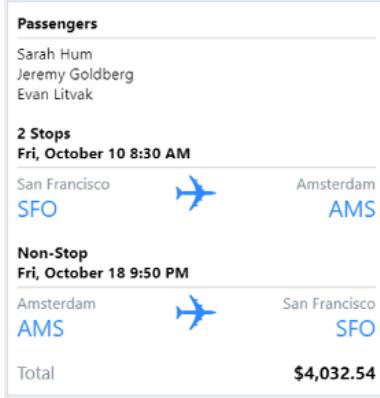
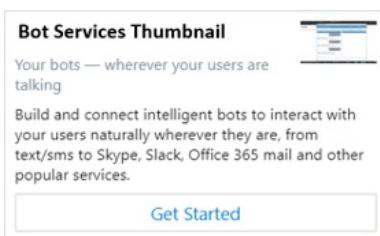
**Функциональные пользовательские элементы управления** — это обычные элементы пользовательского интерфейса, такие как кнопки, изображения, карусели и меню, которые бот представляет пользователю и с которыми пользователь, в свою очередь, взаимодействует для выражения выбора и намерения. Бот может использовать коллекцию элементов управления пользовательского интерфейса для имитации приложения или даже запускаться, будучи встроенным в приложение. Когда бот встроен в приложение или на веб-сайт, он может представлять практически любой элемент управления пользовательского интерфейса, используя возможности приложения, в котором он размещен.

На протяжении десятилетий разработчики приложений и веб-сайтов полагались на элементы управления пользовательского интерфейса, чтобы пользователи могли взаимодействовать со своими приложениями. Эти же элементы управления пользовательского интерфейса также могут быть очень эффективными в ботах. Например, кнопки — это отличный способ предоставить пользователю простой выбор. Пользователю проще и быстрее выбрать "Отели", нажав кнопку с надписью **Отели**, чем вводить это слово на клавиатуре. Это особенно справедливо на мобильных устройствах, где щелчок значительно предпочтительнее ввода.

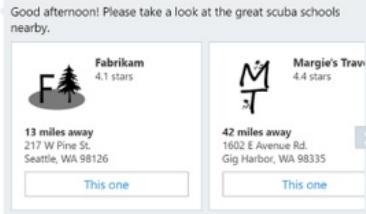
## Карточки

Карточки позволяют предоставлять пользователям различные визуальные, звуковые и (или) выбираемые сообщения и помогают в ведении диалога. Если пользователь должен выбрать из фиксированного набора элементов, вы можете отобразить карусель карточек, каждая из которых содержит изображение, текстовое описание и одну кнопку для выбора. Если у пользователя есть набор вариантов для одного элемента, вы можете представить меньшее одиночное изображение и набор кнопок с различными параметрами на выбор. Запрашивал ли пользователь дополнительную информацию по теме? Карточки могут предоставлять подробную информацию, используя выходное аудио или видео, или квитанцию, в которой подробно описывается история покупок. Существует невероятно широкий спектр применений карт, помогающих в обсуждении взаимодействия между пользователем и программой-роботом. Тип используемой карточки будет определяться потребностями приложения. Давайте подробнее рассмотрим карточки, их действия и некоторые рекомендуемые варианты использования.

Карточки Microsoft Bot Service — это программируемые объекты, содержащие стандартизованные коллекции функциональных пользовательских элементов управления, которые распознаются в широком спектре каналов. В следующей таблице приведен список доступных карточек и рекомендации по использованию для каждого типа.

ТИП КАРТОЧКИ	ПРИМЕР	ОПИСАНИЕ
AdaptiveCard (адаптивная карточка)		Открытый формат обмена карточками, передаваемый как объект JSON. Обычно используется для межканального развертывания карточек. Карточки адаптируются к внешнему виду каждого канала узла.
AnimationCard (анимационная карточка)	 Azure Bot Service Animation Card	Карточка, которая может воспроизводить GIF-файлы с анимацией или короткие видеоролики.
AudioCard (аудиокарточка)		Карточка, которая может воспроизводить звуковой файл.
HeroCard (имиджевая карточка)		Карточка, которая содержит одно большое изображение, одну или несколько кнопок и текст. Обычно используется для визуального выделения потенциального варианта выбора пользователя.
ThumbnailCard (эскизная карточка)		Карточка, которая содержит один эскиз, одну или несколько кнопок и текст. Обычно используется для визуального выделения кнопок для потенциального выбора пользователя.

ТИП КАРТОЧКИ	ПРИМЕР	ОПИСАНИЕ												
ReceiptCard (карточка квитанции)	<p>John Doe</p> <table> <tbody> <tr> <td>Order Number</td> <td>1234</td> </tr> <tr> <td>Payment Method</td> <td>VISA 555...</td> </tr> <tr> <td> Data Transfer</td> <td>\$38.25</td> </tr> <tr> <td> App Service</td> <td>\$45.20</td> </tr> <tr> <td>Tax</td> <td>\$7.50</td> </tr> <tr> <td><b>Total</b></td> <td><b>\$90.95</b></td> </tr> </tbody> </table> <p><a href="#">More Information</a></p>	Order Number	1234	Payment Method	VISA 555...	 Data Transfer	\$38.25	 App Service	\$45.20	Tax	\$7.50	<b>Total</b>	<b>\$90.95</b>	Карточка, с помощью которой бот выдает квитанцию пользователю. Обычно она содержит список элементов, включаемых в квитанцию, налог, а также общую информацию и другой текст.
Order Number	1234													
Payment Method	VISA 555...													
 Data Transfer	\$38.25													
 App Service	\$45.20													
Tax	\$7.50													
<b>Total</b>	<b>\$90.95</b>													
SignInCard (карточка входа в систему)	<p>Bot Services Sign-in Card</p> <p><a href="#">Sign-in</a></p>	Карточка, в которой бот запрашивает вход пользователя. Обычно она содержит текст и одну или несколько кнопок, которые можно нажать, чтобы начать процесс входа.												
Карточка рекомендуемых действий	<p>Chat</p> <p><a href="#">Open Url</a></p> <p><a href="#">Submit</a></p> <p><a href="#">POST</a></p> <p><a href="#">Show Card</a></p>	Представляет вашему пользователю набор CardActions, представляющий собой выбор пользователя. Эта карточка исчезает после выбора любого из рекомендуемых действий.												
VideoCard (видеокарточка)	 <p><b>Big Buck Bunny</b> by the Blender Institute Big Buck Bunny (code-named Peach) is a short computer-animated comedy film by the Blender Institute, part of the Blender Foundation.</p> <p><a href="#">Learn More</a></p>	Карточка, которая может воспроизводить видео. Обычно используется для открытия URL-адреса и потокового воспроизведения доступного видео.												

ТИП КАРТОЧКИ	ПРИМЕР	ОПИСАНИЕ
CardCarousel (карусель карточек)		Горизонтально прокручиваемая коллекция карточек, которая позволяет пользователю легко просматривать ряд возможных пользовательских вариантов выбора.

Карточки позволяют вам создать свой бот один раз и обеспечить его работу на разных каналах. Однако не все типы карточек полностью поддерживаются на всех доступных каналах.

Подробные инструкции по добавлению карточек в ваш бот можно найти в статьях [Добавление мультимедиа в сообщения](#) и [Add suggested actions to messages](#) (Добавление предлагаемых действий в сообщения). Примеры исходного кода для карточек: [C#/JS](#) для адаптивных карточек: [C#/JS](#), для вложений: [C#/JS](#), для предлагаемых действий: [C#/JS](#).

При разработке своего бота не следует автоматически отклонять общие элементы пользовательского интерфейса, из-за того, что их возможностей недостаточно. Как уже было сказано [ранее](#), бот должен быть разработан таким образом, чтобы решить проблему пользователя самым оптимальным, быстрым и простым способом. Избегайте соблазна начать с внедрения возможности распознавания естественного языка, так как зачастую это не нужно и неоправданно усложняет процесс.

#### TIP

Начните с использования минимального количества элементов управления пользовательского интерфейса, которые позволяют боту решать проблему пользователя, и добавляйте другие элементы позже, если текущих будет недостаточно.

## Распознавание текста и естественного языка

Бот может принимать **текст** от пользователей и пытаться проанализировать его с использованием соответствующего регулярного выражения или **API-интерфейсов для распознавания естественного языка**, например [LUIS](#). В зависимости от типа входных данных, предоставляемых пользователем, распознавание естественного языка может оказаться оптимальным или не очень решением.

В некоторых случаях пользователь может **отвечать на конкретный вопрос**. Например, если бот спрашивает: "Как вас зовут?", пользователь может ответить с помощью текста, в котором указано только имя ("Джон"), или с помощью предложения ("Меня зовут Джон").

Формулировка конкретного вопроса уменьшает объем возможных ответов, которые бот может получить, что упрощает логику, необходимую для синтаксического анализа и понимания ответа. Например, рассмотрим следующий открытый вопрос: "Как вы себя чувствуете?". Охватить множество возможных ответов на такой вопрос — очень сложно.

В противоположность этому, такие конкретные вопросы, как "Вам больно? Да или нет" и "Что у вас болит? Грудь, голова, рука или нога", скорее всего, будут способствовать более конкретным ответам, которые бот сможет проанализировать и понять без необходимости реализации распознавания естественного языка.

**TIP**

По возможности задавайте конкретные вопросы, которые не потребуют возможностей распознавания естественного языка для анализа ответа. Это упростит ваш бот и повысит вероятность понимания пользователя ботом.

В других случаях пользователь может **вводить определенную команду**. Например, бот DevOps, который позволяет разработчикам управлять виртуальными машинами, может быть разработан для принятия определенных команд, таких как /STOP VM XYZ или /START VM XYZ. Проектирование бота для принятия определенных команд, подобных этим, обеспечивает хорошее взаимодействие с пользователем, так как синтаксис прост в освоении и ожидаемый результат каждой команды ясен. Кроме того, боту не потребуется распознавание естественного языка, так как входные данные пользователя можно легко проанализировать с использованием регулярных выражений.

**TIP**

При проектировании бота, требующего от пользователя ввода конкретных команд, вы зачастую обеспечиваете оптимальное взаимодействие с пользователем и устраняете необходимость в распознавании естественного языка.

В случае бота *базы знаний* или бота *вопросов и ответов* пользователь может **задавать общие вопросы**. Например, представьте себе бот, который может отвечать на вопросы, основываясь на содержании тысяч документов. [QnA Maker](#) и [Поиск Azure](#) — это технологии, специально разработанные для этого типа сценариев. Дополнительные сведения см. в статье [Проектирование ботов базы знаний](#).

**TIP**

Если вы разрабатываете бот, который будет отвечать на вопросы на основе структурированных или неструктурированных данных из баз данных, веб-страниц или документов, попробуйте использовать технологии, специально разработанные для этого сценария, а не решайте проблему за счет распознавания естественного языка.

В других сценариях пользователь может **вводить простые запросы на основе естественного языка**. Например, пользователь может напечатать "Я хочу пиццу пепперони" или "Есть ли какие-нибудь вегетарианские рестораны в 3 милях от моего дома, открытые сейчас?". API-интерфейсы распознавания естественного языка, такие как [LUIS.ai](#), отлично подходят для таких сценариев.

Используя API-интерфейсы, ваш бот может извлечь ключевые компоненты текста пользователя, чтобы определить его намерение. При реализации возможностей распознавания естественного языка в своем боте подойдите реалистично к возможному уровню детализации входных данных пользователей.

*"I want to find a house for sale that has 3 or 4 bedrooms, priced between \$300 and \$350 with a large garden, about 2000 square feet, preferably green, within 10 miles from my work which is in the city center, with a large garage and a backyard with a pool"*

*-Said nobody, ever*

*"I want to find a house"*

*-Said everybody else*

#### TIP

При создании моделей естественного языка не предполагайте, что пользователи предоставят всю необходимую информацию в первоначальном запросе. Проектируйте бот так, чтобы он запрашивал требуемую информацию, направляя пользователя для ее предоставления, задав ряд вопросов, если это необходимо.

## Речь

Бот может использовать **речевое** средство ввода и/или вывода для обмена данными с пользователями. Если бот предназначен для поддержки устройств без клавиатуры или монитора, речь является единственным средством общения с пользователем.

## Выбор между функциональными пользовательскими элементами управления, текстом, естественным языком и речью

Подобно тому как люди общаются друг с другом, используя комбинацию жестов, голоса и символов, боты могут общаться с пользователями, используя сочетание функциональных пользовательских элементов управления, текст (иногда включающий естественный язык) и речь. Эти методы коммуникации могут использоваться вместе, и вам не нужно выбирать тот или иной.

Например, представьте себе "кулинарный бот", который помогает пользователям с рецептами, и может предоставлять инструкции, воспроизводя видео или отображая серию снимков, чтобы объяснить, что нужно сделать. Некоторые пользователи предпочитают перелистывать страницы рецепта или задавать вопросы боту, используя речь, когда они готовят блюдо по рецепту. Другие могут предпочесть касаться экрана устройства, а не взаимодействовать с ботом с помощью речи. При проектировании программы-робота внедрите элементы UX, которые поддерживают способы взаимодействия пользователей с программой-роботом, учитывая конкретные варианты использования, которые она поддерживает.

# Проектирование ботов базы знаний

27.03.2021 • 15 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

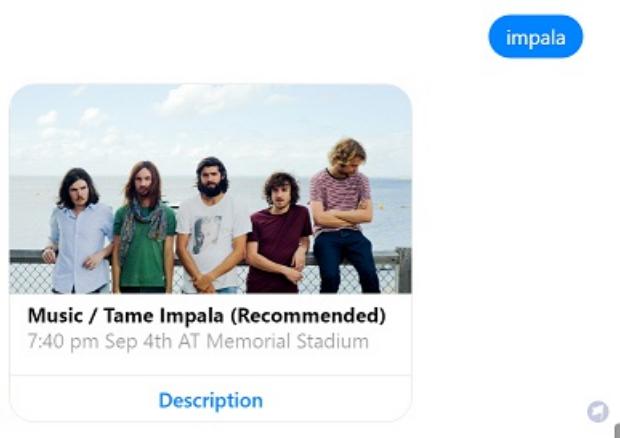
Бот базы знаний может быть разработан для предоставления информации практически на любую тему. Например, один бот базы знаний может отвечать на вопросы о таких событиях, как "What bot events are there at this conference?", "When is the next Reggae show?" или "Who is Tame Impala?" Другой может ответить на вопросы, связанные с ИТ, такие как "How do I update my operating system?" или "Where do I go to reset my password?" Еще один может ответить на вопросы о таких контактах, как "Who is John Doe?" или "What is Jane Doe's email address?"

Независимо от варианта использования, для которого создан бот базы знаний, его основная цель всегда одна и та же: найти и предоставить информацию, которую пользователь запросил, используя массив данных, таких как реляционные данные в базе данных SQL, данные JSON в нереляционном хранилище или PDF-файлы в хранилище документов.

## ПОИСК

Функция поиска может быть ценным инструментом бота.

Во-первых, "нечеткий поиск" позволяет боту предоставлять информацию, которая может иметь отношение к вопросу пользователя, не требуя, чтобы пользователь вводил точные входные данные. Например, если пользователь запрашивает у бота музыкальной базы знаний информацию об "impala" (вместо "Tame Impala"), бот может посыпать информацию, которая, скорее всего, будет иметь отношение к этим входным данным.



Оценки поиска показывают уровень надежности результатов конкретного поиска, позволяя боту соответственно упорядочить результаты или даже настроить их связь на основе уровня надежности. Например, если уровень надежности высокий, бот может ответить "Here is the event that best matches your search".

Atlas Geniuses

Here is the event that best matches your search:



Music / Atlas Genius (Recommended)  
3:40 pm Sep 2nd AT Fisher Green Stage

Description

Если уровень надежности низкий, бот может ответить "Hmm... were you looking for any of these events?"

super smart atlas

Hmm... were you looking for any of these events?



Music / Atlas Genius (Recommended)  
3:40 pm Sep 2nd AT Fisher Green Stage

Description

Music / Super Square  
4:30 pm Sep 3rd At KeyArena

Desc

### Использование поиска для ведения общения

Если необходимо включить базовую функциональность поисковой системы, бот может не понадобиться. Что же такого предлагает интерфейс общения, чего пользователи не могут получить из обычной поисковой системы в веб-браузере?

Боты базы знаний, как правило, наиболее эффективны, если они предназначены для ведения общения. Общение состоит из обмена информацией между ботом и пользователем, который предоставляет боту возможность задавать уточняющие вопросы, предлагать варианты и проверять результаты таким образом, каким не способен выполнить базовый поиск. Например, следующий бот направляет пользователя в общении, ограничивая и фильтруя набор данных, пока не найдет информацию, которую ищет пользователь.

Events

What are you interested in?



Music Comedy Film Laser Dome >

Music

What Music are you interested in?



Any Rock/Pop Hiphop/Rap Soul/R&B >

## Rock/Pop



What day would you like to see Rock/Pop?

Friday

Saturday

Sunday

Any

Here were the Rock/Pop shows Saturday:



### Music / Pony Time (Recommended)

3:00 pm Sep 3rd AT KEXP



### Music / Desi Valentine

3:30 pm Sep 3rd AT Starbucks Stage

Обработав входные данные пользователя на каждом шаге и представив соответствующие параметры, бот направляет пользователям информацию, которую они ищут. Когда бот предоставит эту информацию, он может даже дать рекомендации относительно более эффективных способов поиска подобной информации в будущем.



(By the way - you can also just type "Rock friday" or search an event by name)

Type a message...



## Поиск Azure

Используя [Поиск Azure](#), можно создать эффективный индекс поиска, с помощью которого бот может легко искать, ограничивать и фильтровать информацию. Рассмотрим индекс поиска, созданный с помощью портала Azure.

The screenshot shows the Azure portal's search index configuration screen. It displays the index name 'musicianindex' and its key 'id'. Below this is a table for defining field properties:

FIELD NAME	TYPE	RETRIEVABLE	FILTERABLE	SORTABLE	FACETABLE	SEARCHABLE
imageURL	Edm.String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Name	Edm.String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Era	Edm.String	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Description	Edm.String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
id	Edm.String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
rid	Edm.String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Если необходимо иметь доступ ко всем свойствам хранилища данных, установите каждое свойство как "retrievable". Если необходимо найти музыкантов по имени, установите свойство **Имя** как "searchable". Наконец, если необходимо иметь возможность ограничивающего фильтра над эпохами музыкантов, установите свойство **Эпохи** как "facetable" и "filterable".

Ограничение определяет значения, существующие в хранилище данных для данного свойства, а также величину каждого значения. Например, этот снимок экрана показывает, что в хранилище данных существует 5 различных эпох.

Which era of music are you interested in?

Romantic (11)    Classical (3)    Baroque (2)

Impressionist (2)    Modernist (1)

Фильтрация, в свою очередь, выбирает только указанные экземпляры определенного свойства. Например, можно отфильтровать результаты выше, чтобы содержались только те элементы, где **Эпоха** равна "Romantic".

#### NOTE

Полный пример бота базы знаний, который создается с использованием Azure Document DB, Поиска Azure и Microsoft Bot Framework, см. в [примере бота](#) в репозитории GitHub.

Для простоты в приведенном выше примере показан индекс поиска, который создается с помощью портала Azure. Индексы также можно создавать программными средствами.

## QnA Maker

Некоторые боты базы знаний могут назначаться, чтобы просто отвечать на часто задаваемые вопросы

(FAQs). QnA Maker является мощным средством, предназначенным специально для этого варианта использования. Служба QnA Maker обладает встроенной способностью проверять вопросы и ответы на существующем сайте часто задаваемых вопросов, а также позволяет вручную настраивать собственный список вопросов и ответов. Служба QnA Maker имеет возможность обработки естественного языка, что позволяет ей даже отвечать на вопросы, сформулированные несколько иначе, чем ожидалось. Однако она не имеет возможностей распознавания семантического языка. Например, она не может определить, что щенок — это тип собаки.

Используя веб-интерфейс службы QnA Maker, можно настроить базу знаний с тремя парами вопросов и ответов.

## Knowledge base

The screenshot shows the QnA Maker knowledge base interface. At the top, there is a search bar labeled "Search the knowledge base" and buttons for "Add QnA pair" and settings. Below the search bar, there are three entries:

Question	Answer
can I bring my dog? <span style="border: 1px solid #ccc; padding: 2px;">X</span> +	Dogs are not allowed <span style="border: 1px solid #ccc; padding: 2px;">X</span>
can I bring my vodka? <span style="border: 1px solid #ccc; padding: 2px;">X</span> +	Alcohol is not allowed <span style="border: 1px solid #ccc; padding: 2px;">X</span>
hi <span style="border: 1px solid #ccc; padding: 2px;">X</span> +	hello <span style="border: 1px solid #ccc; padding: 2px;">X</span>

Затем можно проверить ее, задав ряд вопросов.

The screenshot shows the QnA Maker test interface. It includes a "Test" button, a "Start over" button, and a message input field with placeholder "Type your message...". Below the input field, there are three test interactions:

- Interaction 1: Question "Can I bring my tea?" is highlighted in blue. Response "Alcohol is not allowed" is shown in a grey box. Below it, "ScreenShots at 9:35 AM" is displayed.
- Interaction 2: Question "Can I bring my vodka?" is highlighted in blue. Response "Alcohol is not allowed" is shown in a grey box. Below it, "ScreenShots at 9:34 AM" is displayed.
- Interaction 3: Question "Can I bring my dog?" is highlighted in blue. Response "Dogs are not allowed" is shown in a grey box. Below it, "ScreenShots at 9:34 AM" is displayed.

Бот правильно отвечает на вопросы, которые непосредственно сопоставляются с теми, которые были настроены в базе знаний. Однако он неправильно отвечает на вопрос "can I bring my tea?", потому что этот вопрос наиболее похож по структуре на вопрос "can I bring my vodka?". Причина, по которой служба QnA Maker дает неверный ответ, заключается в том, что она не распознает смысл слова. Она не знает, что "tea" — это тип безалкогольного напитка. Поэтому она отвечает: "Alcohol is not allowed".

#### TIP

Создайте пары QnA, а затем проверьте и переустановите бот, нажав кнопку "Проверить" в общении, чтобы выбрать альтернативный ответ для каждого неверного ответа.

## LUIS

Некоторые боты базы знаний требуют возможности обработки естественного языка (NLP), чтобы анализировать сообщения пользователя для определения его намерений. [API распознавания речи \(LUIS\)](#) обеспечивает быстрое и эффективное средство добавления возможностей NLP к ботам. Служба LUIS позволяет использовать существующие, предварительно построенные модели от Bing и Cortana всякий раз, когда они соответствуют потребностям, а также создавать собственные специализированные модели.

При работе с огромными наборами данных необязательно обучать модель NLP при каждом изменении сущности. Например, при работе с музыкальным ботом пользователь может сообщить "Play Reggae", "Play Bob Marley" или "Play One Love". Хотя бот мог сопоставлять каждое из этих сообщений с намерением "playMusic", не обучаясь каждому исполнителю, жанру и названию песни, модель NLP не сможет определить, является ли сущность жанром, исполнителем или песней. Используя модель NLP для идентификации универсальной сущности типа "music", бот может искать свое хранилище данных для этой сущности и исходить оттуда.

## Объединение Поиска, QnA Maker и/или LUIS

Поиск, QnA Maker и LUIS — это все мощные средства, но они также могут быть объединены для создания ботов базы знаний, которые обладают более чем одной из этих возможностей.

### LUIS и Поиск

В примере бота музыкального фестиваля, [описанного ранее](#), бот ведет общение, показывая кнопки, которые представляют список. Однако этот бот может также включать понимание естественного языка, используя LUIS для определения намерений и сущностей в таких вопросах, как "what kind of music does Romit Girdhar play?". Бот может выполнять поиск в индексе Поиска Azure, используя имя музыканта.

Было бы невозможно обучить модель любому возможному имени музыканта, так как существует много возможных значений, но можно предоставить достаточно типичных примеров для LUIS, чтобы правильно идентифицировать попавшуюся сущность. Например, учтите, что модель обучается на примерах музыкантов.

what kind of music does romit girdhar play ?

answerGenre

Submit

what genre of music is foxygen ?

answerGenre

Submit

При тестировании этой модели с новыми высказываниями типа "what kind of music do the beatles play?", LUIS успешно определяет намерение "answerGenre" и идентифицирует сущность "beatles". Однако, если задать более длинный вопрос, например "what kind of music does the devil makes three play?", LUIS идентифицирует "the devil" как сущность.

```
"entities": [
  {
    "entity": "the devil",
    "type": "musician",
    "startIndex": 25,
    "endIndex": 33,
    "score": 0.300864249
  }
]
```

Обучая модель примерам сущностей, которые представляют собой базовый набор данных, можно повысить точность распознавания языка бота.

#### TIP

В общем, лучше, чтобы модель ошибалась, идентифицируя лишние слова в распознавании сущности, например "John Smith please" из высказывания "Call John Smith please", вместо того, чтобы идентифицировать слишком мало слов, например "John" из высказывания "Call John Smith please". Индекс поиска будет игнорировать неподходящее слова, такие как "please" во фразе "John Smith please".

## LUIS и QnA Maker

Некоторые боты базы знаний могут использовать QnA Maker для ответа на основные вопросы в сочетании с LUIS, чтобы определять намерения, извлекать сущности и вызывать более сложные диалоги. К примеру, возьмем простой бот службы технической поддержки ИТ. Этот бот может использовать QnA Maker для ответа на основные вопросы о Windows или Outlook, но также может потребоваться облегчить такие сценарии, какброс пароля, которые требуют распознавания намерений и обратной связи между пользователем и ботом. Существует несколько способов, с помощью которых бот может реализовать гибрид LUIS и QnA Maker.

1. Вызовите одновременно службы QnA Maker и LUIS и ответьте пользователю, используя информацию из первой, которая возвращает оценку определенного порога.
2. Сначала вызовите LUIS, и если никакое намерение не соответствует определенному пороговому значению, например запускается намерение None, после чего вызывается QnA Maker. В качестве альтернативы создайте намерение LUIS для QnA Maker, предоставляя модель LUIS с примерами вопросов QnA, сопоставляемые с QnAIIntent.
3. Сначала вызовите QnA Maker и, если ответ не соответствует определенному пороговому значению, вызовите LUIS.

Пакет SDK Bot Framework предоставляет встроенную поддержку LUIS и QnA Maker. Это позволяет активировать диалоги или автоматически отвечать на вопросы с помощью LUIS и/или QnA Maker без необходимости реализации пользовательских вызовов для любого из этих средств. Дополнительные сведения см. в [руководстве по средству Dispatch](#).

**TIP**

При реализации комбинации LUIS, QnA Maker и/или Поиск Azure введите тестовые входные данные в каждое из средств, чтобы определить пороговое значение для каждой из моделей. Каждая из служб (LUIS, QnA Maker и Поиск Azure) генерирует значение, используя разные критерии оценки, поэтому оценки, полученные в этих средствах, не сопоставляются напрямую. Кроме того, LUIS и QnA Maker нормализуют оценки. Определенную оценку можно считать "хорошой" в одной модели LUIS, но "не очень" в другой модели.

## Образец кода

- Пример, в котором показано, как создать базовый бот базы знаний с помощью пакета SDK Bot Framework для .NET, см. в [примере бота базы знаний](#) в репозитории GitHub.

# Перевод разговора с бота на человека

27.03.2021 • 8 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Каким бы искусственным интеллектом ни обладал бот, иногда необходимо перевести разговор на человека. Это может быть необходимо, так как бот не понимает пользователя (из-за ограничения искусственного интеллекта) или запрос невозможно автоматизировать и требуется вмешательство человека. В таких случаях бот должен понимать, когда это делать, и обеспечивать пользователю удобный переход.

Microsoft Bot Framework — это открытая платформа, позволяющая разработчикам интегрировать различные платформы привлечения агентов.

## Модели интеграции передачи

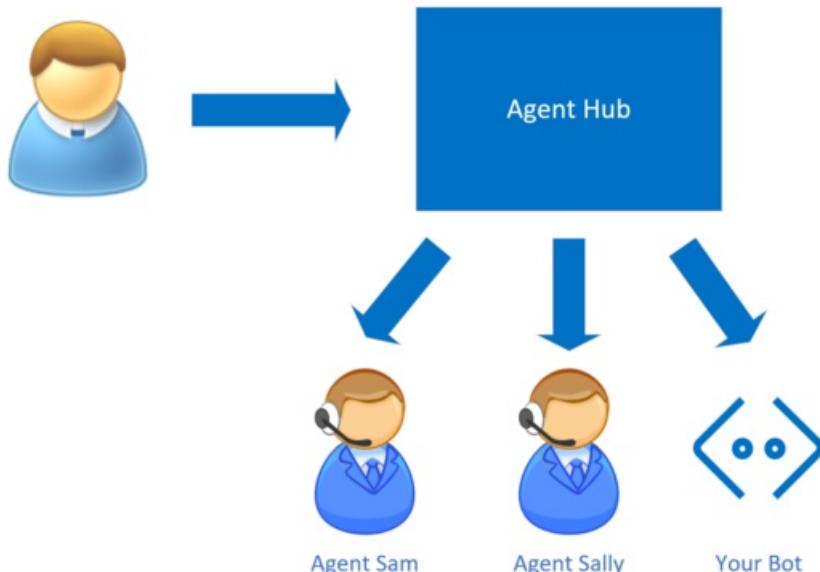
Платформа Microsoft Bot Framework поддерживает две модели интеграции с платформами привлечения агентов. Протокол передачи одинаков для обеих моделей, однако сведения об адаптации различаются для различных моделей и платформ привлечения агентов.

Целью этого не является предоставление универсального решения для интеграции с любой системой клиента, **а не** предоставление **общего языка** и рекомендаций для разработчиков-роботов и системных интеграторов, создающих беседы с человеком в цикле.

### Бот как агент

В первой модели, называемой "Бот как агент", бот объединяет ранги оперативных агентов, подключенных к центру агентов, и отвечает на запросы пользователей, как если бы эти запросы поступали из любого другого канала Bot Framework. Беседу между пользователем и ботом можно передать агенту-человеку, после чего бот покидает активную беседу.

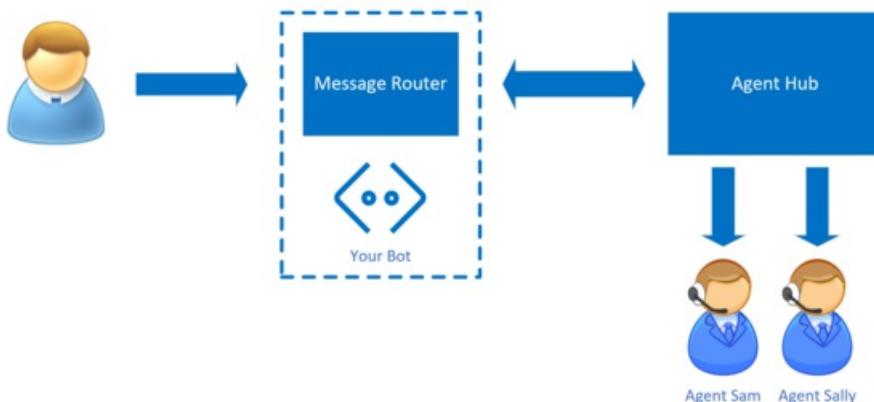
Основное преимущество этого режима заключается в простоте. Существующий бот можно подключить к центру агентов, приложив минимальные усилия, при этом все задачи перенаправления сообщений берет на себя центр агентов.



## Бот в качестве посредника

Вторая модель называется "Бот как посредник". Пользователь напрямую говорит с ботом, пока бот не решит, что ему нужна помощь агента. Компонент маршрутизатора сообщений в боте перенаправляет беседу в центр агентов, который передает его соответствующему агенту. Бот остается в цикле и может записать расшифровку диалога, отфильтровать сообщения или предоставить дополнительное содержимое как агенту, так и пользователю.

Гибкость и возможности управления являются основными преимуществами этой модели. Бот может поддерживать различные каналы и контролировать передачу и перенаправление бесед между пользователем, ботом и центром агентов.



## Протокол передачи

Протокол выдается по центру событий для инициации, отправки программой-робота в канал и обновления состояния, отправленного каналом в Bot.

### Инициация передачи

Событие *передачи, инициированное программой-роботом*, создается с целью передачи.

Это событие состоит из двух компонентов.

- **Контекст запроса** на пересылку, необходимый для маршрутизации диалога к правому агенту.
- Запись **разговора**. Агент может прочитать диалог, состоявшийся между клиентом и ботом перед иницированием передачи.

Ниже приведены поля событий, которые будут передаваться при инициации:

- **Name** — `name` **Обязательное** поле, для которого задано значение `"handoff.initiate"`.
- **Value** — это `value` объект, содержащий содержимое JSON, зависящее от концентратора агента, например требуемый навык агента и т. д. Это поле является **необязательным**.

```
{ "Skill" : "credit cards" }
```

- **Вложения** — `attachments` это **необязательное** поле, содержащее список `Attachment` объектов. The Bot Framework определяет тип вложения "Транскрипция", который используется для отправки записи разговора в центр агента, если это необходимо. Вложения могут отправляться либо в виде встроенных данных (с ограничением по размеру), либо в автономном режиме путем предоставления `ContentUrl`.

```
handoffEvent.Attachments = new List<Attachment> {
    new Attachment {
        Content = transcript,
        ContentType = "application/json",
        Name = "Transcript",
    }};
}
```

#### NOTE

Концентраторы агентов **должны игнорировать** типы вложений, которые они не понимают.

- CONVERSATION — `conversation` является **обязательным** полем типа, `ConversationAccount` описывающим переданный диалог. Критическое, оно должно включать диалог `Id`, который можно использовать для корреляции с другими событиями.

Когда программа-робот обнаруживает необходимость передачи диалога в агента, она сигнализирует о своем намерении, отправив событие передающей инициации. В C# метод API более высокого уровня `CreateHandoffInitiation` можно использовать, как показано в следующем фрагменте кода.

```
var activities = GetRecentActivities();
var handoffContext = new { Skill = "credit cards" };
var handoffEvent =
    EventFactory.CreateHandoffInitiation(
        turnContext, handoffContext, new Transcript(activities));
await turnContext.SendActivityAsync(handoffEvent);
```

#### Состояние передачи

Событие " состояния передачи" отправляется в Bot концентратором агентов. Событие информирует робот о состоянии инициированной операции передачи.

#### NOTE

Программы-роботы **не требуются** для работы с событием, однако они **не должны** отклонять его.

Ниже приведены поля событий «состояние передачи».

- Name — `name` **Обязательное** поле, для которого задано значение `"handoff.status"`.
- Value — `value` **Обязательное** поле, описывающее текущее состояние операции передачи. Это объект JSON, содержащий **требуемое** поле `state` и необязательное поле `message`, как определено ниже.

`state` Имеет одно из следующих значений:

- `accepted` — Агент принял запрос и отдал управление диалогом.
- `failed` -Не удалось выполнить переданный запрос. `message` Может содержать дополнительные сведения, относящиеся к сбою.
- `completed` -Запрос на перевыполнение завершен.

Формат и возможное значение `message` поля не указаны.

- Успешное завершение передачи:

```
{ "state" : "completed" }
```

- Сбой операции передачи из-за истечения времени ожидания:

```
{ "state" : "failed", "message" : "Cannot find agent with requested skill" }
```

- Диалоговое обсуждение** - `Conversation` является **обязательным** полем типа `ConversationAccount`, описывающим диалог, который был принят или отклонен. Объект `Id` диалога должен быть таким же, как и в хэндлере инитиации, который инициировал передачу.

## Библиотека передачи

[Библиотека](#) передачи была создана, чтобы дополнить пакет SDK для Bot Framework версии 4 в поддержке передачи; относятся

- Реализует дополнения к пакету SDK для Bot Framework для поддержки передачи агенту (также известному как **эскалация**).
- Содержит определения трех типов событий для сигнализации операций передачи.

### NOTE

Интеграции с конкретными концентриаторами агентов не являются частью библиотеки.

## Дополнительные ресурсы

- [Интеграция с Microsoft Dynamics Omnichannel для обслуживания клиентов](#)
- [Интеграция с платформой Ливерпуль Ливингаже](#)
- [Диалоги](#)
- [Службы для переприятия](#)

# Внедрение бота в приложение

27.10.2020 • 3 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Несмотря на то что боты чаще всего находятся за пределами приложения, их также можно интегрировать с приложениями. Например, можно внедрить [бот набора знаний](#) в приложение, чтобы пользователи могли быстро найти сведения, которые в противном случае было бы трудно найти в сложной структуре приложения. Бот можно внедрить в приложение службы поддержки, которое будет выступать в качестве первого отвечающего устройства на входящие запросы пользователей. Бот может независимо устранять простые проблемы и [передавать](#) более сложные агенту-человеку.

## Интеграция бота с приложением

Способ интеграции бота с приложением зависит от типа приложения.

### Собственное мобильное приложение

Приложение, создаваемое с использованием машинного кода, может взаимодействовать с Bot Framework через [Direct Line API](#) с помощью REST или подключения WebSocket.

### Мобильное веб-приложение

Мобильное приложение, созданное с помощью веб-языка и платформ, таких как [Cordova](#), может взаимодействовать с Bot Framework, используя те же компоненты, что и [бот, внедренный на веб-сайте](#), только инкапсулированные в собственной оболочке приложения.

### Приложение Интернета вещей

Приложение Интернета вещей может взаимодействовать с Bot Framework с помощью [Direct Line API](#). В некоторых сценариях они также могут использовать [Microsoft Cognitive Services](#) для предоставления таких возможностей, как распознавание изображений и речь.

### Другие виды приложений и игр

Другие виды приложений и игр могут взаимодействовать с Bot Framework с помощью [Direct Line API](#).

## Создание кроссплатформенного мобильного приложения для выполнения бота

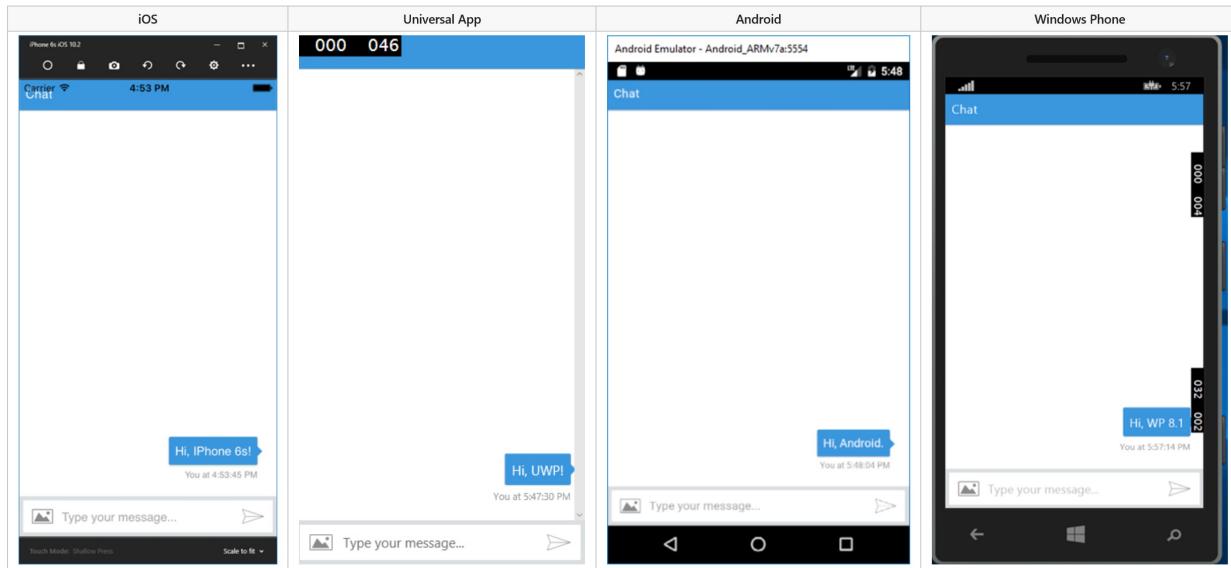
В этом примере создания мобильного приложения для выполнения бота используется [Xamarin](#), популярное средство для создания кроссплатформенных мобильных приложений.

Сначала создайте простой веб-компонент для просмотра и используйте его для размещения [элемента управления веб-чатом](#). Затем с помощью портала Azure добавьте канал веб-чата.

Затем укажите зарегистрированный URL-адрес веб-чата в качестве источника для элемента управления для просмотра в приложении Xamarin:

```
public class WebPage : ContentPage
{
    public WebPage()
    {
        var browser = new WebView();
        browser.Source = "https://webchat.botframework.com/embed/<YOUR SECRET KEY HERE>";
        this.Content = browser;
    }
}
```

С помощью этого процесса вы можете создать кроссплатформенное мобильное приложение, преобразовывающее внедренное веб-представление с элементом управления веб-чата для просмотра.



## Дополнительные ресурсы

- [Direct Line API](#)
- [Microsoft Cognitive Services](#).

# Внедрение бота на веб-сайт

27.03.2021 • 5 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Несмотря на то что боты часто находятся за пределами веб-сайтов, их также можно внедрить на веб-сайтах. Например, можно внедрить [бота набора знаний](#) на веб-сайте, чтобы пользователи могли быстро найти сведения, которые в противном случае было бы трудно найти в сложной структуре веб-сайта. Кроме того, бот можно внедрить на веб-сайте службы поддержки, который будет выступать в качестве первого отвечающего устройства на входящие запросы пользователей. Бот может независимо устранять простые проблемы и [передавать](#) более сложные агенту-человеку.

В этой статье рассматриваются интеграции ботов с веб-сайтами и использование механизма *обратного канала* для упрощения частной связи между веб-страницей и ботом.

Корпорация Майкрософт предоставляет два различных способа интеграции бота с веб-сайтом: веб-элемент управления Skype и веб-элемента управления с открытым кодом.

## Веб-элемент управления Skype

### NOTE

Начиная с 31 октября 2019 г. канал Skype не принимает новые запросы на публикацию ботов. Это означает, что вы можете разрабатывать боты с использованием канала Skype, но бот будет доступен только 100 пользователям. Вы не сможете опубликовать бота для большего числа пользователей. Текущие боты в Skype будут работать без прерываний. Узнайте больше о том, [почему некоторые функции недоступны в Skype](#).

[Веб-элемент управления Skype](#) — это по сути клиент Skype в веб-элементе управления. Встроенная проверка подлинности Skype позволяет боту проводить проверку подлинности и распознавание пользователей, не требуя от разработчика написания пользовательского кода. Служба Skype автоматически распознает учетные записи Майкрософт, используемых в ее веб-клиенте.

Так как веб-элемент управления Skype выступает в качестве внешнего интерфейса для Skype, клиент Skype пользователя автоматически имеет доступ ко всему контексту любого диалога, осуществляемого с использованием веб-элемента управления. Даже закрыв веб-браузер, пользователь может продолжать взаимодействие с ботом с помощью клиента Skype.

## Веб-элемент управления с открытым кодом

[Элемент управления веб-чатом с открытым кодом](#) создан на базе ReactJS и использует [Direct Line API](#) для взаимодействия с Bot Framework. Элемент управления веб-чатом предоставляет пустой холст для реализации веб-чата, его возможности, а также дает полный контроль над его поведением.

Механизм *обратного канала* обеспечивает прямое взаимодействие веб-страницы, на которой размещается элемент управления, с ботом способом, полностью невидимым для пользователя. Эта возможность позволяет использовать ряд полезных сценариев:

- Веб-страница может отправлять боту соответствующие данные (например, GPS-расположение).
- Веб-страница может сообщать боту о действиях пользователя (например, "пользователь только что выбрал вариант A из раскрывающегося списка").
- Веб-страница может отправить боту маркер проверки подлинности для пользователя, вошедшего в

систему.

- Бот может отправлять соответствующие данные на веб-страницу (например, текущее значение портфеля пользователя).
- Бот может отправлять "команды" веб-странице (например, изменение цвета фона).

## Использование механизма обратного канала

Элемент управления веб-чата с открытым кодом взаимодействует с ботами с помощью Direct Line API, что позволяет отправлять `activities` между клиентом и ботом. Наиболее распространенный тип действия — `message`, но существуют также другие типы. Например, тип действия `typing` указывает, что пользователь вводит текст или что бот составляет ответ.

Механизм обратного канала можно использовать для обмена данными между клиентом и ботом, не показывая их пользователю, задав тип действия `event`. Элемент управления веб-чата будет автоматически игнорировать все действия с `type="event"`.

## Образец кода

Элемент управления веб-чатом с открытым кодом доступен на сайте GitHub. Дополнительные сведения о том, как можно реализовать механизм обратного канала с помощью элемента управления веб-чатом с открытым кодом и пакета SDK Bot Framework для Node.js, см. в статье [Use the backchannel mechanism](#) (Использование механизма обратного канала).

## Дополнительные ресурсы

- [Direct Line API](#)
- [Элемент управления "Веб-чат" с открытым кодом](#)
- [Использование механизма обратного канала](#)

# Отправка и получение текстовых сообщений

27.03.2021 • 5 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Основным способом взаимодействия бота с пользователями будут действия с **сообщениями**. Некоторые сообщения будут состоять из обычного текста, а другие — включать расширенное содержимое, такое как карточки или вложения. Обработчик этапа бота получает сообщения от пользователя, и вы можете отправлять ответы пользователю из него. Объект контекста этапа предоставляет методы для отправки сообщений обратно пользователю. В этой статье объясняется, как отправлять простые текстовые сообщения.

Markdown поддерживается для большинства текстовых полей, но особенности поддержки зависят от канала.

Для выполняющегося бота, отправляющего и получающего сообщения, выполните краткие инструкции в верхней части содержания или ознакомьтесь со [статьей о работе ботов](#), в которой также содержатся ссылки на простые примеры, которые вы можете выполнить самостоятельно.

## Отправка текстового сообщения

Для отправки простого текстового сообщения укажите строку, которую вы хотите отправить как действие:

- [C#](#)
- [JavaScript](#)
- [Python](#)
- [LG](#)

В обработчиках действий бота используйте метод `SendActivityAsync` для объекта контекста шага, чтобы отправить ответ в виде одного сообщения. Кроме того, вы можете использовать метод `SendActivitiesAsync` объекта для отправки нескольких ответов за раз.

```
await turnContext.SendActivityAsync($"Welcome!");
```

## Получение текстового сообщения

Чтобы получить простое текстовое сообщение, используйте свойство `text` объекта `activity`.

- [C#](#)
- [JavaScript](#)
- [Python](#)
- [LG](#)

В обработчиках действий бота используйте следующий код для получения сообщения.

```
var responseMessage = turnContext.Activity.Text;
```

## Отправка индикатора ввода

Пользователи ожидают своевременный ответ на сообщения. Если бот выполняет какую-то длительную задачу, например, вызывает сервер или выполняет запрос, не давая пользователю никаких указаний на то, что его запрос принят во внимание, нетерпеливый пользователь может отправить дополнительные сообщения, предполагая, что бот сломан.

Боты в каналах Web Chat и Direct Line могут поддерживать отправку индикатора ввода, указывающего пользователю, что сообщение получено и обрабатывается. Имейте в виду, что бот должен завершить реплику в течение 15 секунд, иначе для службы Connector истечет времени ожидания. Чтобы реализовать длительную обработку, см. дополнительные сведения об [упреждающих сообщениях](#).

В следующем примере показана отправка индикатора ввода.

- [C#](#)
- [JavaScript](#)
- [Python](#)
- [LG](#)

```
protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    if (string.Equals(turnContext.Activity.Text, "wait",
System.StringComparison.InvariantCultureIgnoreCase))
    {
        await turnContext.SendActivitiesAsync(
            new Activity[] {
                new Activity { Type = ActivityTypes.Typing },
                new Activity { Type = "delay", Value= 3000 },
                MessageFactory.Text("Finished typing", "Finished typing"),
            },
            cancellationToken);
    }
    else
    {
        var replyText = $"Echo: {turnContext.Activity.Text}. Say 'wait' to watch me type.";
        await turnContext.SendActivityAsync(MessageFactory.Text(replyText, replyText), cancellationToken);
    }
}
```

## Дополнительные ресурсы

- [Обработка действий](#)
- [Раздел "Действие сообщений"](#)
- [Создание текста](#)

## Дальнейшие действия

[Добавление мультимедиа в сообщения](#)

# Добавление мультимедиа в сообщения

27.03.2021 • 25 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Обмен сообщениями между пользователем и ботом может включать вложения мультимедиа, такие как изображения, видео, аудио и файлы. Пакет SDK Bot Framework поддерживает задачу отправки пользователю форматированного сообщения. Чтобы определить, какой тип форматированных сообщений поддерживает канал (Slack, Facebook и др.), см. сведения об ограничениях в документации по этому каналу.

## Предварительные требования

- Базовые знания о [ботах](#).
- Код в этой статье основан на следующих примерах:
  - **Использование карточек:** [C#](#), [JavaScript](#), [Python](#)
  - **Обработка вложений:** [C#](#), [JavaScript](#), [Python](#)
  - **Предлагаемые действия:** [C#](#), [JavaScript](#), [Python](#)

## Отправка вложений

Чтобы отправить пользователю содержимое, например изображение или видео, нужно добавить вложение или список вложений в сообщение.

Примеры доступных карт см. [в статье Проектирование взаимодействия с пользователем](#).

См. также [ограничение размера файла, переданного с помощью каналов](#) в разделе часто задаваемых вопросов.

- [C#](#)
- [JavaScript](#)
- [Python](#)

Весь исходный код, приведенный в этом разделе, основан на примере [обработки вложений](#).

Свойство `Attachments` объекта `Activity` содержит массив объектов `Attachment`, представляющих вложения в виде форматированных карточек и файлов мультимедиа. Чтобы добавить мультимедийное вложение в сообщение, создайте объект `Attachment` для действия `reply` и задайте свойства `ContentType`, `ContentUrl` и `Name`.

Чтобы создать ответное сообщение, определите текст и настройте вложения. Присвоение вложений ответному сообщению выполняется одинаково для всех типов вложений, но настройка и определение разных вложений будут отличаться, как показано в следующих фрагментах. Ниже приведен код для настройки ответа со встроенным вложением:

`Bots/AttachmentsBot.cs`

```
reply = MessageFactory.Text("This is an inline attachment.");
reply.Attachments = new List<Attachment>() { GetInlineAttachment() };
```

Далее мы рассмотрим разные типы вложений. Во-первых, это встроенные вложения:

## Bots/AttachmentsBot.cs

```
private static Attachment GetInlineAttachment()
{
    var imagePath = Path.Combine(Environment.CurrentDirectory, @"Resources", "architecture-resize.png");
    var imageData = Convert.ToBase64String(File.ReadAllBytes(imagePath));

    return new Attachment
    {
        Name = @"Resources\architecture-resize.png",
        ContentType = "image/png",
        ContentUrl = $"data:image/png;base64,{imageData}",
    };
}
```

Во-вторых, отправленные вложения:

## Bots/AttachmentsBot.cs

```
private static async Task<Attachment> GetUploadedAttachmentAsync(ITurnContext turnContext, string
serviceUrl, string conversationId, CancellationToken cancellationToken)
{
    if (string.IsNullOrWhiteSpace(serviceUrl))
    {
        throw new ArgumentNullException(nameof(serviceUrl));
    }

    if (string.IsNullOrWhiteSpace(conversationId))
    {
        throw new ArgumentNullException(nameof(conversationId));
    }

    var imagePath = Path.Combine(Environment.CurrentDirectory, @"Resources", "architecture-resize.png");

    var connector = turnContext.TurnState.Get<IConnectorClient>() as ConnectorClient;
    var attachments = new Attachments(connector);
    var response = await attachments.Client.Conversations.UploadAttachmentAsync(
        conversationId,
        new AttachmentData
        {
            Name = @"Resources\architecture-resize.png",
            OriginalBase64 = File.ReadAllBytes(imagePath),
            Type = "image/png",
        },
        cancellationToken);

    var attachmentUri = attachments.GetAttachmentUri(response.Id);

    return new Attachment
    {
        Name = @"Resources\architecture-resize.png",
        ContentType = "image/png",
        ContentUrl = attachmentUri,
    };
}
```

И, в-третьих, вложения из Интернета:

## Bots/AttachmentsBot.cs

```

private static Attachment GetInternetAttachment()
{
    // ContentUrl must be HTTPS.
    return new Attachment
    {
        Name = @"Resources\architecture-resize.png",
        ContentType = "image/png",
        ContentUrl = "https://docs.microsoft.com/en-us/bot-framework/media/how-it-works/architecture-
resize.png",
    };
}

```

Если вложение представляет собой изображение, аудиофайл или видео, служба соединителя будет передавать данные вложения каналу так, чтобы позволить [каналу](#) обрабатывать это вложение в диалоге. Если вложение представляет собой файл, URL-адрес файла будет отображаться в диалоге как гиперссылка.

## Отправка карточки для имиджевого баннера

Помимо изображений или видео, вы можете прикрепить *карточку для имиджевого баннера*, которая позволяет совмещать изображения и кнопки в один объект и отправлять их в таком виде пользователю. Markdown поддерживается для большинства текстовых полей, но особенности поддержки зависят от канала.

- [C#](#)
- [JavaScript](#)
- [Python](#)

Чтобы создать сообщение с помощью карточки и кнопки Hero, можно присоединить [HeroCard](#) объект к сообщению.

Исходный код, показанный здесь, основан на примере [обработки вложений](#).

### Bots/AttachmentsBot.cs

```

private static async Task DisplayOptionsAsync(ITurnContext turnContext, CancellationToken cancellationToken)
{
    // Create a HeroCard with options for the user to interact with the bot.
    var card = new HeroCard
    {
        Text = "You can upload an image or select one of the following choices",
        Buttons = new List<CardAction>
        {
            // Note that some channels require different values to be used in order to get buttons to
            // display text.
            // In this code the emulator is accounted for with the 'title' parameter, but in other channels
            // you may
            // need to provide a value for other parameters like 'text' or 'displayText'.
            new CardAction(ActionTypes.ImBack, title: "1. Inline Attachment", value: "1"),
            new CardAction(ActionTypes.ImBack, title: "2. Internet Attachment", value: "2"),
            new CardAction(ActionTypes.ImBack, title: "3. Uploaded Attachment", value: "3"),
        },
    };

    var reply = MessageFactory.Attachment(card.ToAttachment());
    await turnContext.SendActivityAsync(reply, cancellationToken);
}

```

## Обработка событий в форматированных карточках

Для обработки событий в карточках с широкими возможностями используйте объекты *действий карты*, чтобы указать, что должно происходить при нажатии пользователем кнопки или касании раздела карточки. Каждое действие карточки имеет свойство *Type* и *value*.

Чтобы правильно работать, назначьте тип действия каждому элементу, который можно щелкнуть, на карточке Hero. В этой таблице перечислены и описаны доступные типы действий и требуемый формат для связанного свойства. `messageBack` Действие карточки имеет более обобщенное значение, чем другие действия с картой. Дополнительные сведения о типах действий карты см. в разделе [действие карты](#) [схемы действий](#) `messageBack`.

ТИП	ОПИСАНИЕ	ЗНАЧЕНИЕ
вызывает	Инициирует телефонный звонок.	Целевое назначение телефонного звонка в следующем формате: <code>tel:123123123123</code> .
downloadFile	Скачивает файл.	URL-адрес для скачивания файла.
imBack	Отправляет боту сообщение и отображает полученный ответ в чате.	Текст отправляемого сообщения.
мессажбакк	Представляет текстовый ответ, отправляемый через систему разговора.	Необязательное программное значение, включаемое в создаваемые сообщения.
openUrl	Открывает URL-адрес в окне встроенного браузера.	URL-адрес, который нужно открыть.
playAudio	Воспроизводит звук.	URL-адрес для воспроизведения звука.
playVideo	Воспроизводит видео.	URL-адрес для воспроизведения видео.
postBack	Отправляет боту сообщение, но не всегда отображает полученный ответ в чате.	Текст отправляемого сообщения.
showImage	Отображает изображение.	URL-адрес для отображения изображения.
signin	Инициирует процесс входа OAuth.	URL-адрес потока OAuth, который нужно запустить.

## Карточка для имиджевого баннера с различными типами событий

В следующем коде показаны примеры использования различных событий форматированных карточек.

- [C#](#)
- [JavaScript](#)
- [Python](#)

Примеры всех доступных карт см. в примере [использования карточек](#).

```

public static HeroCard GetHeroCard()
{
    var heroCard = new HeroCard
    {
        Title = "BotFramework Hero Card",
        Subtitle = "Microsoft Bot Framework",
        Text = "Build and connect intelligent bots to interact with your users naturally wherever they are,"
+
            " from text/sms to Skype, Slack, Office 365 mail and other popular services.",
        Images = new List<CardImage> { new CardImage("https://sec.ch9.ms/ch9/7ff5/e07cfef0-aa3b-40bb-9baa-7c9ef8ff7ff5/buildreactionbotframework_960.jpg") },
        Buttons = new List<CardAction> { new CardAction(ActionTypes.OpenUrl, "Get Started", value: "https://docs.microsoft.com/bot-framework") },
    };

    return heroCard;
}

```

## Cards.cs

```

public static SigninCard GetSigninCard()
{
    var signinCard = new SigninCard
    {
        Text = "BotFramework Sign-in Card",
        Buttons = new List<CardAction> { new CardAction(ActionTypes.Signin, "Sign-in", value: "https://login.microsoftonline.com/") },
    };

    return signinCard;
}

```

## Отправка адаптивной карточки

Хотя *фабрику сообщений* можно использовать для создания сообщения, содержащего вложение (любой сортировки), *Адаптивная карта* — это один конкретный тип вложения. Обратите внимание, что некоторые каналы не поддерживают адаптивные карты, и каналы, которые могут только частично поддерживать их. Например, при отправке адаптивной карточки в Facebook, кнопки не будут работать, а тексты и изображения будут отображаться. Фабрика сообщений — это класс поддержки пакета SDK для Bot Framework, используемый для автоматизации этапов создания.

Адаптивные карточки — это открытый формат обмена данными, который обеспечивает унифицированную и согласованную передачу содержимого пользовательского интерфейса между разработчиками. Однако не все каналы поддерживают адаптивные карты.

[Конструктор адаптивных карточек](#) предоставляет широкие интерактивные возможности для разработки адаптивных карт.

### NOTE

Вы должны протестировать эту функцию, выбрав каналы, которые будут использоваться ботом, чтобы определить, поддерживают ли они адаптивные карточки.

- [C#](#)
- [JavaScript](#)
- [Python](#)

Чтобы использовать адаптивные карточки, обязательно добавьте пакет NuGet [AdaptiveCards](#).

Представленный здесь исходный код основан на примере [использования карточек](#).

## Cards.cs

Этот пример считывает файл JSON с адаптивной картой из файла и добавляет его в качестве вложения.

```
public static Attachment CreateAdaptiveCardAttachment()
{
    // combine path for cross platform support
    var paths = new[] { ".", "Resources", "adaptiveCard.json" };
    var adaptiveCardJson = File.ReadAllText(Path.Combine(paths));

    var adaptiveCardAttachment = new Attachment()
    {
        ContentType = "application/vnd.microsoft.card.adaptive",
        Content = JsonConvert.DeserializeObject(adaptiveCardJson),
    };

    return adaptiveCardAttachment;
}
```

## Отправка карусели карточек

Сообщения также могут включать несколько вложений в макете карусели, где вложения помещаются одно за другим и пользователь может их прокручивать.

- [C#](#)
- [JavaScript](#)
- [Python](#)

Представленный здесь исходный код основан на примере [использования карточек](#).

## Dialogs/MainDialog.cs

Сначала создайте ответ и определите вложения в виде списка.

```
// Cards are sent as Attachments in the Bot Framework.
// So we need to create a list of attachments for the reply activity.
var attachments = new List<Attachment>();

// Reply to the activity we received with an activity.
var reply = MessageFactory.Attachment(attachments);
```

Затем добавьте вложения и задайте для типа макета значение « \_обойма\_ ». Здесь мы добавляем их по одному, но вы можете управлять этим списком и добавлять в него карточки любым удобным методом.

```
// Display a carousel of all the rich card types.
reply.AttachmentLayout = AttachmentLayoutTypes.Carousel;
reply.Attachments.Add(Cards.CreateAdaptiveCardAttachment());
reply.Attachments.Add(Cards.GetAnimationCard().ToAttachment());
reply.Attachments.Add(Cards.GetAudioCard().ToAttachment());
reply.Attachments.Add(Cards.GetHeroCard().ToAttachment());
reply.Attachments.Add(Cards.GetOAuthCard().ToAttachment());
reply.Attachments.Add(Cards.GetReceiptCard().ToAttachment());
reply.Attachments.Add(Cards.GetSigninCard().ToAttachment());
reply.Attachments.Add(Cards.GetThumbnailCard().ToAttachment());
reply.Attachments.Add(Cards.GetVideoCard().ToAttachment());
```

Завершив добавление вложений, вы можете отправить этот ответ так же, как и любой другой.

```
// Send the card(s) to the user as an attachment to the activity
await stepContext.Context.SendActivityAsync(reply, cancellationToken);
```

## Дополнительные ресурсы

Примеры доступных карт см. в статье [проектирование взаимодействия с пользователем](#).

См. дополнительные сведения о [схеме карточек Bot Framework](#) и [действиях в беседах](#).

### Пример кода для обработки входных данных адаптивной карточки

В следующем примере показан один из способов использования адаптивных входных карт в классе диалогового окна Bot. Он расширяет образец Hero Cards, проверяя входные данные, полученные в текстовом поле, от отвечающего клиента. Сначала необходимо добавить функциональные возможности ввода текста и кнопки в существующую адаптивную карту, добавив следующий код непосредственно перед завершающей скобкой adaptiveCard.json, расположенной в папке resources:

```
"actions": [
  {
    "type": "Action.ShowCard",
    "title": "Text",
    "card": {
      "type": "AdaptiveCard",
      "body": [
        {
          "type": "Input.Text",
          "id": "text",
          "isMultiline": true,
          "placeholder": "Enter your comment"
        }
      ],
      "actions": [
        {
          "type": "Action.Submit",
          "title": "OK"
        }
      ]
    }
  }
]
```

Обратите внимание, что идентификатор текстового поля ввода имеет значение `"text"`. При нажатии **кнопки OK** сообщение, формируемое адаптивной картой, будет иметь свойство `value` со свойством с именем, `"text"`, которое содержит сведения, введенные пользователем в поле ввода текста карточки.

- [C#](#)
- [JavaScript](#)
- [Python](#)

Наш проверяющий элемент управления использует [Newtonsoft.js](#), чтобы сначала преобразовать его в `Object`, а затем создать обрезанную текстовую строку для сравнения. Поэтому добавьте:

```
using System;
using System.Linq;
using Newtonsoft.Json.Linq;
```

чтобы `MainDialog.cs` и установить последний стабильный пакет NuGet [Newtonsoft.js](#). В коде

проверяющего элемента управления мы добавили поток логики в комментариях в коде. Этот `ChoiceValidator` метод помещается в образец `using cards` сразу после закрывающей скобки `public` для объявления майндиалог:

```
private async Task ChoiceValidator(
    PromptValidatorContext promptContext,
    CancellationToken cancellationToken)
{
    // Retrieves Adaptive Card comment text as JObject.
    // looks for JObject field "text" and converts that input into a trimmed text string.
    var jobject = promptContext.Context.Activity.Value as JObject;
    var jtoken = jobject?["text"];
    var text = jtoken?.Value().Trim();

    // Logic: 1. if succeeded = true, just return promptContext
    //        2. if false, see if JObject contained Adaptive Card input.
    //        No = (bad input) return promptContext
    //        Yes = update Value field with JObject text string, return "true".
    if (!promptContext.Recognized.Succeeded && text != null)
    {
        var choice = promptContext.Options.Choices.FirstOrDefault(
            c => c.Value.Equals(text, StringComparison.InvariantCultureIgnoreCase));
        if (choice != null)
        {
            promptContext.Recognized.Value = new FoundChoice
            {
                Value = choice.Value,
            };
            return true;
        }
    }
    return promptContext.Recognized.Succeeded;
}
```

Теперь над `MainDialog` изменением объявлений:

```
// Define the main dialog and its related components.
AddDialog(new ChoicePrompt(nameof(ChoicePrompt))');
```

на:

```
// Define the main dialog and its related components.
AddDialog(new ChoicePrompt(nameof(ChoicePrompt), ChoiceValidator));
```

Это вызовет проверяющий элемент управления для поиска адаптивных входных карт при каждом создании нового запроса выбора.

Чтобы протестировать код после отображения адаптивной карты, нажмите кнопку **текст**, введите допустимый выбор, например *Hero Card*, а затем нажмите кнопку **OK**.

The screenshot shows a Microsoft Bot Framework conversation window. At the top, there are tabs for 'Welcome' and 'Live Chat'. Below them are buttons for 'Restart conversation' and 'Save transcript'. The URL 'http://localhost:3978/api/messages' is displayed. The main area shows a flight itinerary from Amsterdam (AMS) to San Francisco (SFO) on Friday, October 18 at 9:50 PM. The total cost is \$4,032.54. A red box highlights a 'Hero Card' section containing the text 'Hero Card' and an 'OK' button. Below this, a message box says 'Type anything to see another card.' A timestamp 'A minute ago' is shown. Another message box asks 'What card would you like to see? You can click or type the card name'. A timestamp 'A minute ago' is shown again. At the bottom, a yellow bar labeled 'BotFramework Hero Card' contains a 'Get started' button and a 'New messages' button. A text input field with placeholder 'Type your message' and a send icon is also present.

1. Первые входные данные будут использоваться для запуска нового диалога.
2. Снова нажмите кнопку **OK**, и эти входные данные будут использоваться для выбора новой карточки.

## Дальнейшие действия

[Добавление кнопок для управления действиями пользователя](#)

# Использование кнопки для ввода данных

27.03.2021 • 3 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Кнопки расширяют возможности общения, позволяя пользователю ответить на вопрос или выбрать нужную кнопку вместо того, чтобы вводить ответ с помощью клавиатуры. В отличие от кнопок, отображаемых в карточках с широкими возможностями (которые остаются видимыми и доступными пользователю даже после выбора), кнопки, отображаемые на панели предлагаемые действия, исчезают после того, как пользователь сделает выбор. Это не позволяет пользователю выбирать устаревшие кнопки в диалоге и упрощает разработку с помощью Bot, поскольку вам не нужно учитывать этот сценарий.

## Предложение действий с использованием кнопки

Функция *предлагаемых действий* позволяет боту представлять кнопки. Вы можете создать список предлагаемых действий (также называемых *быстрыми ответами*), которые будут отображаться пользователю для единого включения диалога.

- [C#](#)
- [JavaScript](#)
- [Python](#)

Представленный здесь исходный код основан на примере [предложенных действий](#).

```
// Creates and sends an activity with suggested actions to the user
/// clicks one of the buttons the text value from the "CardAction" will be
/// displayed in the channel just as if the user entered the text. There are multiple
/// "ActionTypes" that may be used for different situations.
private static async Task SendSuggestedActionsAsync(ITurnContext turnContext, CancellationToken cancellationToken)
{
    var reply = MessageFactory.Text("What is your favorite color?");

    reply.SuggestedActions = new SuggestedActions()
    {
        Actions = new List<CardAction>()
        {
            new CardAction() { Title = "Red", Type = ActionTypes.ImBack, Value = "Red", Image =
"https://via.placeholder.com/20/FF0000?text=R", ImageAltText = "R" },
            new CardAction() { Title = "Yellow", Type = ActionTypes.ImBack, Value = "Yellow", Image =
"https://via.placeholder.com/20/FFFF00?text=Y", ImageAltText = "Y" },
            new CardAction() { Title = "Blue", Type = ActionTypes.ImBack, Value = "Blue", Image =
"https://via.placeholder.com/20/0000FF?text=B", ImageAltText = "B" },
        },
    };
    await turnContext.SendActivityAsync(reply, cancellationToken);
}
```

## Дополнительные ресурсы

Вы можете получить полный исходный код для примера **предлагаемых действий** в [C#](#), [JavaScript](#) и [Python](#).

## Дальнейшие действия

[Сохранение данных пользователя и диалога](#)

# Сохранение данных пользователя и диалога

27.03.2021 • 17 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Бот по своей природе не учитывает состояния. Развернутый бот не обязан выполнять следующий шаг в том же процессе или на том же компьютере, что и предыдущий. Но иногда боту нужно отслеживать контекст беседы, чтобы управлять ее ходом и запоминать ответы на предыдущие вопросы. Функции состояния и хранения, предоставляемые пакетом SDK Bot Framework, позволяют реализовать в боте поддержку состояния. Для администрирования и хранения данных боты используют объекты хранилища и управления состоянием. Диспетчер состояний предоставляет уровень абстракции, позволяющий пол учить доступ к свойствам состояния через методы доступа независимо от типа базового хранилища.

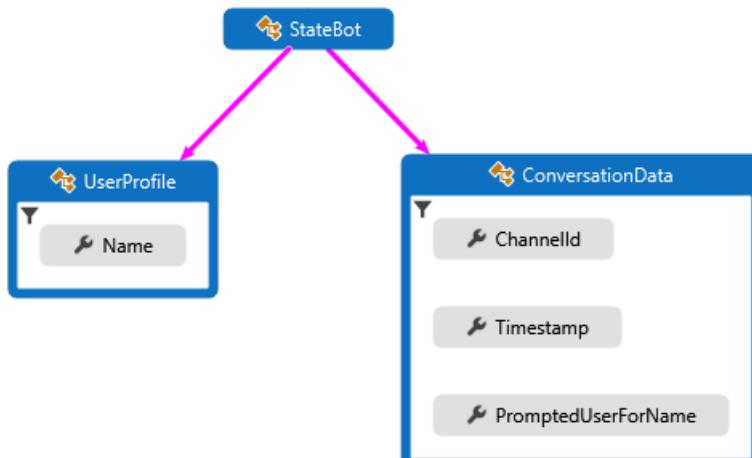
## Предварительные требования

- Требуется понимание [основных сведений о ботах](#) и [управления состоянием](#) для ботов.
- Код в этой статье основан на примере **бота управления состоянием**. Вам потребуется копия этого примера на языке [C#](#), [JavaScript](#) или [Python](#).

## Об этом примере

Получив от пользователя входные данные, этот пример проверяет сохраненное состояние беседы и определяет, выводился ли этому пользователю запрос для ввода имени. Если еще нет, он запрашивает имя пользователя и сохраняет ответ в состоянии пользователя. Если да, сохраненное в состоянии пользователя имя используется для взаимодействия с пользователем и обработки входных данных, а затем вместе со временем получения сообщения и идентификатором канала возвращается обратно пользователю. Значения времени и идентификатора канала извлекаются из данных беседы с пользователем, а затем сохраняются в состоянии беседы. На следующей схеме показана связь между ботом, профилем пользователя и классами данных беседы.

- [C#](#)
- [JavaScript](#)
- [Python](#)



## Определение классов

- C#
- JavaScript
- Python

При настройке управления состоянием первым делом нужно определить классы, которые будут содержать все нужные сведения для управления состоянием пользователя и беседы. В примере, используемом в этой статье, определяются следующие классы:

- В `UserProfile.cs` вы определяете `UserProfile` класс для сведений о пользователе, которые будут собираются Bot.
- В `ConversationData.cs` вы определяете `ConversationData` класс для управления нашим состоянием диалога при сборе сведений о пользователе.

В приведенных ниже примерах кода показаны определения классов `UserProfile` и `ConversationData`.

### UserProfile.cs

```
// Defines a state property used to track information about the user.
public class UserProfile
{
    public string Name { get; set; }
}
```

### ConversationData.cs

```
// Defines a state property used to track conversation data.
public class ConversationData
{
    // The time-stamp of the most recent incoming message.
    public string Timestamp { get; set; }

    // The ID of the user's channel.
    public string ChannelId { get; set; }

    // Track whether we have already asked the user's name
    public bool PromptedUserForName { get; set; } = false;
}
```

## Создание объектов состояния беседы и пользователя

- C#
- JavaScript
- Python

Далее вы регистрируетесь `MemoryStorage`, которая используется для создания `UserState` `ConversationState` объектов и. В `Startup` создаются объекты состояния пользователя и беседы, а в конструктор бота добавляются зависимости. Также для бота регистрируются дополнительные службы: поставщик учетных данных, адаптер и реализация бота.

### Startup.cs.

```
// Create the storage we'll be using for User and Conversation state.
// (Memory is great for testing purposes - examples of implementing storage with
// Azure Blob Storage or Cosmos DB are below).
var storage = new MemoryStorage();
```

```
// Create the User state passing in the storage layer.  
var userState = new UserState(storage);  
services.AddSingleton(userState);  
  
// Create the Conversation state passing in the storage layer.  
var conversationState = new ConversationState(storage);  
services.AddSingleton(conversationState);
```

## Bots/StateManagementBot.cs

```
private BotState _conversationState;  
private BotState _userState;  
  
public StateManagementBot(ConversationState conversationState, UserState userState)  
{  
    _conversationState = conversationState;  
    _userState = userState;  
}
```

## Добавление методов доступа к свойству состояния

- [C#](#)
- [JavaScript](#)
- [Python](#)

Теперь вы создаете методы доступа к свойствам с помощью `CreateProperty` метода, который предоставляет для `BotState` объекта маркер. Каждый метод доступа к свойству состояния позволяет получить или задать значение для соответствующего свойства состояния. Прежде чем использовать свойства состояния, используйте каждый метод доступа, чтобы загрузить свойство из хранилища и получить его из кэша состояний. Чтобы получить ключ с правильной областью действия, связанный со свойством `State`, вызовите `GetAsync` метод.

## Bots/StateManagementBot.cs

```
var conversationStateAccessors = _conversationState.CreateProperty<ConversationData>(nameof(ConversationData));  
var userStateAccessors = _userState.CreateProperty<UserProfile>(nameof(UserProfile));
```

## Доступ к состояниям из кода бота

В предыдущем разделе мы рассматривали шаги, которые на этапе инициализации добавляют в бота методы доступа к свойствам состояния. Теперь эти методы доступа можно использовать во время выполнения для чтения и записи сведений о состоянии. Следующий пример кода использует представленный здесь поток логики:

- [C#](#)
- [JavaScript](#)
- [Python](#)

- Если `userProfile.Name` является пустым и Конверсациондата. Промптедусерфорнаме имеет значение `true`, вы получаете указанное имя пользователя и сохраняете его в пользовательской среде.
- Если `userProfile.Name` является пустым и Конверсациондата. Промптедусерфорнаме имеет значение `false`, запрашивается имя пользователя.
- Если `userProfile.Name` ранее хранился, вы получаете время сообщения и идентификатор канала из

вводимых пользователем данных, выводите все данные обратно пользователю и сохраняете полученные данные в состоянии диалога.

## Bots/StateManagementBot.cs

```
protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    // Get the state properties from the turn context.

    var conversationStateAccessors = _conversationState.CreateProperty<ConversationData>
(nameof(ConversationData));
    var conversationData = await conversationStateAccessors.GetAsync(turnContext, () => new
ConversationData());

    var userStateAccessors = _userState.CreateProperty<UserProfile>(nameof(UserProfile));
    var userProfile = await userStateAccessors.GetAsync(turnContext, () => new UserProfile());

    if (string.IsNullOrEmpty(userProfile.Name))
    {
        // First time around this is set to false, so we will prompt user for name.
        if (conversationData.PromptedUserForName)
        {
            // Set the name to what the user provided.
            userProfile.Name = turnContext.Activity.Text?.Trim();

            // Acknowledge that we got their name.
            await turnContext.SendActivityAsync($"Thanks {userProfile.Name}. To see conversation data, type
anything.");
        }

        // Reset the flag to allow the bot to go through the cycle again.
        conversationData.PromptedUserForName = false;
    }
    else
    {
        // Prompt the user for their name.
        await turnContext.SendActivityAsync($"What is your name?");

        // Set the flag to true, so we don't prompt in the next turn.
        conversationData.PromptedUserForName = true;
    }
}
else
{
    // Add message details to the conversation data.
    // Convert saved Timestamp to local DateTimeOffset, then to string for display.
    var messageTimeOffset = (DateTimeOffset) turnContext.Activity.Timestamp;
    var localMessageTime = messageTimeOffset.ToLocalTime();
    conversationData.Timestamp = localMessageTime.ToString();
    conversationData.ChannelId = turnContext.Activity.ChannelId.ToString();

    // Display state data.
    await turnContext.SendActivityAsync($"{userProfile.Name} sent: {turnContext.Activity.Text}");
    await turnContext.SendActivityAsync($"Message received at: {conversationData.Timestamp}");
    await turnContext.SendActivityAsync($"Message received from: {conversationData.ChannelId}");
}
}
```

Перед выходом из обработчика поворачивания используйте метод *SaveChangesAsync()* объектов управления состоянием для записи всех изменений состояния обратно в хранилище.

## Bots/StateManagementBot.cs

```

public override async Task OnTurnAsync(ITurnContext turnContext, CancellationToken cancellationToken =
default(CancellationToken))
{
    await base.OnTurnAsync(turnContext, cancellationToken);

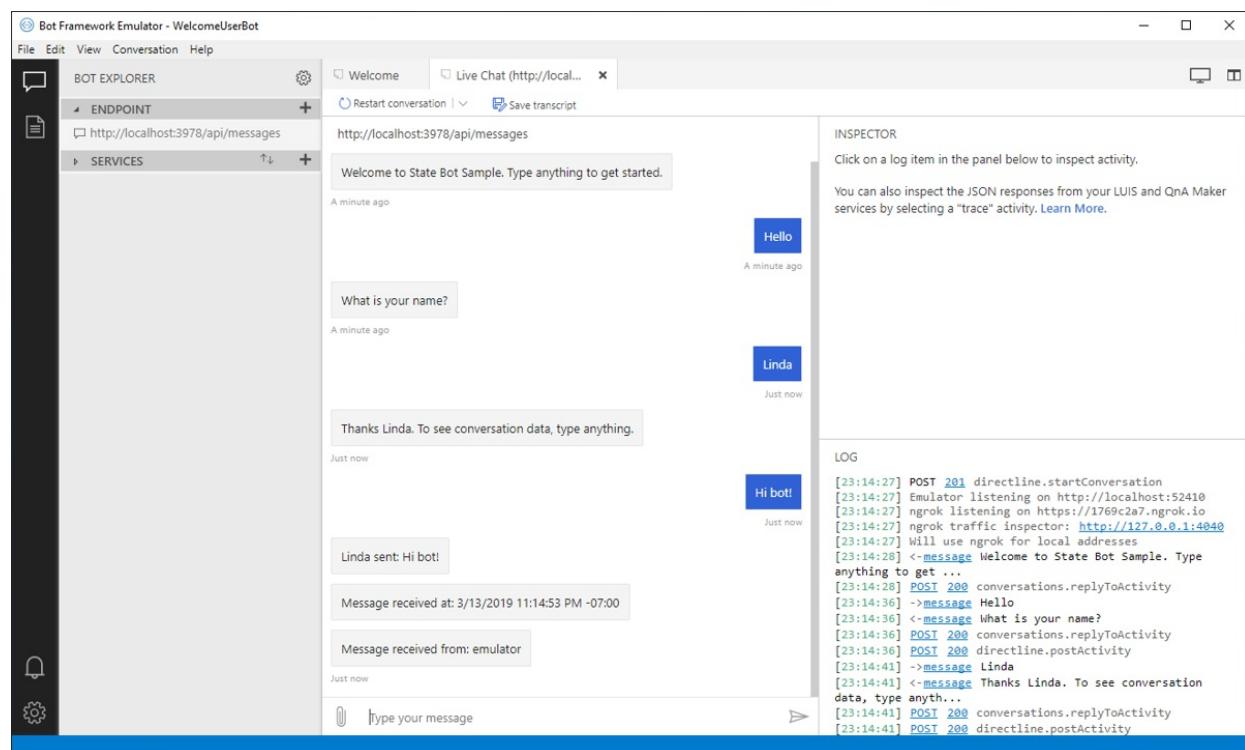
    // Save any state changes that might have occurred during the turn.
    await _conversationState.SaveChangesAsync(turnContext, false, cancellationToken);
    await _userState.SaveChangesAsync(turnContext, false, cancellationToken);
}

```

## Тестирование бота

Скачайте и установите последнюю версию [Bot Framework Emulator](#).

- Выполните этот пример на локальном компьютере. Если потребуются дополнительные инструкции, в файле README можно найти [пример кода на C#](#) или [пример кода на JS](#).
- Примените эмулятор для тестирования бота, как показано ниже.



## Дополнительные ресурсы

**Конфиденциальность.** Если вы собираетесь хранить персональные данные пользователя, обеспечьте соблюдение [Общего регламента по защите данных](#).

**Управление состоянием.** Все вызовы методов управления состоянием обрабатываются асинхронно, и по умолчанию применяется только последнее действие, выполняющее запись данных. На практике следует размещать методы get, set и save state как можно ближе друг к другу в коде бота.

**Критически важные бизнес-данные.** Используйте состояние бота для хранения настроек, имени пользователя или сведений о последнем заказе, но не используйте его для хранения критически важных бизнес-данных. Для критически важных данных [создайте собственные компоненты хранилища](#) или записывайте их непосредственно в [хранилище](#).

**Recognizer-Text.** В этом примере используются библиотеки Microsoft/Recognizer-Text для синтаксического анализа и проверки пользовательского ввода. Дополнительные сведения см. на странице [Использование Azure DNS для частных доменов](#).

## Дальнейшие действия

Теперь вы умеете настраивать состояние, которое упрощает чтение и запись данных бота в хранилище. Давайте перейдем к изучению методов, позволяющих задать пользователю ряд вопросов, проверить его ответы и сохранить полученные данные.

[Создание собственных запросов на сбор данных, вводимых пользователем.](#)

# Создание собственных запросов на сбор данных, вводимых пользователем

27.03.2021 • 17 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Общение между ботом и пользователем часто подразумевает запрашивание у пользователя информации, анализ его ответа и выполнение действий с учетом этой информации. Бот должен отслеживать контекст общения, чтобы управлять его ходом и запоминать ответы на предыдущие вопросы. *Состояние бота* — это информация, которую программа отслеживает для правильных ответов на входящие сообщения.

## TIP

Библиотека диалоговых окон содержит встроенные запросы, предоставляющие дополнительные функциональные возможности, которые пользователи могут использовать. Примеры таких запросов можно найти в статье [о реализации последовательного потока беседы](#).

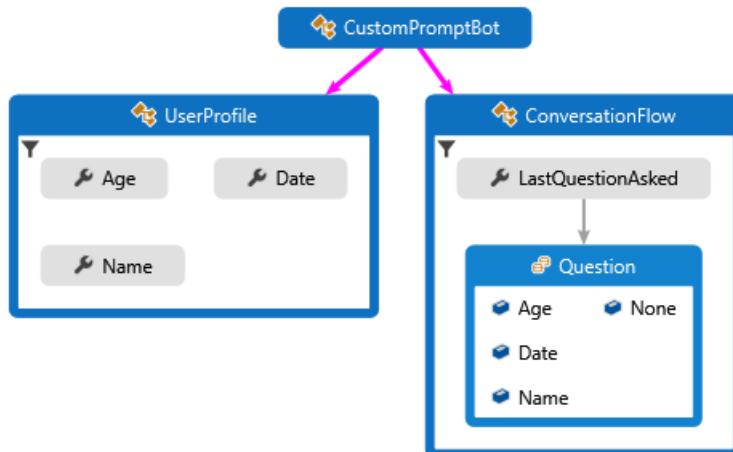
## Предварительные требования

- Код в этой статье основан на примере запроса на ввод данных пользователем. Вам потребуется копия примера для [C#, JavaScript](#) или [Python](#).
- Понимание принципов [управления состоянием](#) и [сохранения данных пользователя и диалога](#).

## Сведения о примере кода

Этот пример бота задает пользователю несколько вопросов, а затем проверяет и сохраняет введенные данные. На следующей схеме показана связь между ботом, профилем пользователя и классами потоков беседы.

- [C#](#)
- [JavaScript](#)
- [Python](#)



- Класс `UserProfile` для хранения собранных ботом сведений о пользователе.

- Класс `ConversationFlow` для управления состоянием беседы при сборе сведений о пользователе.
- Внутреннее `ConversationFlow.Question` перечисление для отслеживания в диалоге.

Пользовательское состояние будет отформатировать имя пользователя, его возраст и выбранную дату, а состояние беседы будет отводиться на то, что вы только что запросили пользователя. Так как вы не планируете развертывать этот робот, вы настроите состояние пользователя и диалога для использования *хранилища памяти*.

Чтобы управлять потоком диалога и коллекцией входных данных, используйте обработчики передачи сообщений Bot, свойства состояния пользователя и диалога. В программе-роботе вы зарегистрируете сведения о свойстве состояния, полученные во время каждой итерации обработчика поочередных сообщений.

## Создание объектов беседы и пользователя

- [C#](#)
- [JavaScript](#)
- [Python](#)

Создание объектов состояния пользователя и сеанса при запуске и их использование с помощью внедрения зависимостей в конструкторе бота.

### Startup.cs

```
// Create the storage we'll be using for User and Conversation state. (Memory is great for testing purposes.)
services.AddSingleton<IStorage, MemoryStorage>();

// Create the User state.
services.AddSingleton<UserState>();

// Create the Conversation state.
services.AddSingleton<ConversationState>();
```

### Bots/CustomPromptBot.cs

```
private readonly BotState _userState;
private readonly BotState _conversationState;

public CustomPromptBot(ConversationState conversationState, UserState userState)
{
    _conversationState = conversationState;
    _userState = userState;
}
```

## Создание методов доступа к свойствам

- [C#](#)
- [JavaScript](#)
- [Python](#)

Создание методов доступа к свойствам профиля пользователя и потока диалога и вызов `GetAsync` для получения значения свойства из состояния.

### Bots/CustomPromptBot.cs

```
protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    var conversationStateAccessors = _conversationState.CreateProperty<ConversationFlow>
(nameof(ConversationFlow));
    var flow = await conversationStateAccessors.GetAsync(turnContext, () => new ConversationFlow(),
cancellationToken);

    var userStateAccessors = _userState.CreateProperty<UserProfile>(nameof(UserProfile));
    var profile = await userStateAccessors.GetAsync(turnContext, () => new UserProfile(),
cancellationToken);
}
```

Перед завершением шага вызовите `SaveChangesAsync`, чтобы записать изменения состояния в хранилище.

```
// Save changes.
await _conversationState.SaveChangesAsync(turnContext, false, cancellationToken);
await _userState.SaveChangesAsync(turnContext, false, cancellationToken);
}
```

## Обработчик шагов бота для сообщений

При обработке действий с сообщениями обработчик сообщений использует вспомогательный метод для управления диалогом и отправки запроса пользователю. Этот вспомогательный метод описан в следующем разделе.

- [C#](#)
- [JavaScript](#)
- [Python](#)

### Bots/CustomPromptBot.cs

```
protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    var conversationStateAccessors = _conversationState.CreateProperty<ConversationFlow>
(nameof(ConversationFlow));
    var flow = await conversationStateAccessors.GetAsync(turnContext, () => new ConversationFlow(),
cancellationToken);

    var userStateAccessors = _userState.CreateProperty<UserProfile>(nameof(UserProfile));
    var profile = await userStateAccessors.GetAsync(turnContext, () => new UserProfile(),
cancellationToken);

    await FillOutUserProfileAsync(flow, profile, turnContext, cancellationToken);

    // Save changes.
    await _conversationState.SaveChangesAsync(turnContext, false, cancellationToken);
    await _userState.SaveChangesAsync(turnContext, false, cancellationToken);
}
```

## Заполнение профиля пользователя

Бот запрашивает у пользователя сведения на основе вопроса, который был задан ботом на предыдущем шаге (если такой вопрос был задан). Входные данные анализируются с помощью метода проверки.

Каждый метод проверки характеризуется следующим образом:

- Возвращаемое значение позволяет узнать, содержат ли входные данные допустимый ответ на этот вопрос.
- Если проверка проходит успешно, она возвращает извлеченное и нормализованное значение для сохранения.
- Если процедура проверки завершается ошибкой, она создает сообщение, которое бот может отправить для повторного запроса той же информации.

Методы проверки описаны в следующем разделе.

- [C#](#)
- [JavaScript](#)
- [Python](#)

[Bots/CustomPromptBot.cs](#)

```

private static async Task FillOutUserProfileAsync(ConversationFlow flow, UserProfile profile, ITurnContext
turnContext, CancellationToken cancellationToken)
{
    var input = turnContext.Activity.Text?.Trim();
    string message;

    switch (flow.LastQuestionAsked)
    {
        case ConversationFlow.Question.None:
            await turnContext.SendActivityAsync("Let's get started. What is your name?", null, null,
cancellationToken);
            flow.LastQuestionAsked = ConversationFlow.Question.Name;
            break;
        case ConversationFlow.Question.Name:
            if (ValidateName(input, out var name, out message))
            {
                profile.Name = name;
                await turnContext.SendActivityAsync($"Hi {profile.Name}.", null, null, cancellationToken);
                await turnContext.SendActivityAsync("How old are you?", null, null, cancellationToken);
                flow.LastQuestionAsked = ConversationFlow.Question.Age;
                break;
            }
            else
            {
                await turnContext.SendActivityAsync(message ?? "I'm sorry, I didn't understand that.", null,
null, cancellationToken);
                break;
            }
        case ConversationFlow.Question.Age:
            if (ValidateAge(input, out var age, out message))
            {
                profile.Age = age;
                await turnContext.SendActivityAsync($"I have your age as {profile.Age}.", null, null,
cancellationToken);
                await turnContext.SendActivityAsync("When is your flight?", null, null, cancellationToken);
                flow.LastQuestionAsked = ConversationFlow.Question.Date;
                break;
            }
            else
            {
                await turnContext.SendActivityAsync(message ?? "I'm sorry, I didn't understand that.", null,
null, cancellationToken);
                break;
            }
        case ConversationFlow.Question.Date:
            if (ValidateDate(input, out var date, out message))
            {
                profile.Date = date;
                await turnContext.SendActivityAsync($"Your cab ride to the airport is scheduled for
{profile.Date}.");
                await turnContext.SendActivityAsync($"Thanks for completing the booking {profile.Name}.");
                await turnContext.SendActivityAsync($"Type anything to run the bot again.");
                flow.LastQuestionAsked = ConversationFlow.Question.None;
                profile = new UserProfile();
                break;
            }
            else
            {
                await turnContext.SendActivityAsync(message ?? "I'm sorry, I didn't understand that.", null,
null, cancellationToken);
                break;
            }
    }
}

```

# Синтаксический анализ и проверка входных данных

Для проверки входных данных бот использует указанные ниже критерии.

- **Name** (имя) не может быть пустой строкой. Для нормализации выполняется усечение пробелов.
- Значение **age** (возраст) должно находиться в диапазоне от 18 до 120. Для нормализации оно округляется до целого числа.
- **Date** (дата) может быть любой датой или временем в будущем, если разница между указанным значением и текущим временем составляет не менее одного часа. Для нормализации возвращается только дата, содержащаяся во входных данных.

## NOTE

Для ввода возраста и даты используются [текстовые библиотеки Microsoft/распознаватели](#) для выполнения начального анализа. При предоставлении примера кода вы не узнаете, как работают библиотеки распознавателей текста, и это лишь один из способов анализа входных данных. Дополнительные сведения об этих библиотеках можно найти в репозитории README.

- [C#](#)
- [JavaScript](#)
- [Python](#)

## Bots/CustomPromptBot.cs

```
private static bool ValidateName(string input, out string name, out string message)
{
    name = null;
    message = null;

    if (string.IsNullOrWhiteSpace(input))
    {
        message = "Please enter a name that contains at least one character.";
    }
    else
    {
        name = input.Trim();
    }

    return message is null;
}

private static bool ValidateAge(string input, out int age, out string message)
{
    age = 0;
    message = null;

    // Try to recognize the input as a number. This works for responses such as "twelve" as well as "12".
    try
    {
        // Attempt to convert the Recognizer result to an integer. This works for "a dozen", "twelve", "12",
        // and so on.
        // The recognizer returns a list of potential recognition results, if any.

        var results = NumberRecognizer.RecognizeNumber(input, Culture.English);

        foreach (var result in results)
        {
            // The result resolution is a dictionary, where the "value" entry contains the processed string.
            if (result.Resolution.TryGetValue("value", out var value))
            {
                age = Convert.ToInt32(value);
            }
        }
    }
}
```

```

        if (age >= 18 && age <= 120)
        {
            return true;
        }
    }

    message = "Please enter an age between 18 and 120.";
}
catch
{
    message = "I'm sorry, I could not interpret that as an age. Please enter an age between 18 and
120.";
}

return message is null;
}

private static bool ValidateDate(string input, out string date, out string message)
{
    date = null;
    message = null;

    // Try to recognize the input as a date-time. This works for responses such as "11/14/2018", "9pm",
    "tomorrow", "Sunday at 5pm", and so on.
    // The recognizer returns a list of potential recognition results, if any.
    try
    {
        var results = DateTimeRecognizer.RecognizeDateTime(input, Culture.English);

        // Check whether any of the recognized date-times are appropriate,
        // and if so, return the first appropriate date-time. We're checking for a value at least an hour in
        the future.
        var earliest = DateTime.Now.AddHours(1.0);

        foreach (var result in results)
        {
            // The result resolution is a dictionary, where the "values" entry contains the processed input.
            var resolutions = result.Resolution["values"] as List<Dictionary<string, string>>;

            foreach (var resolution in resolutions)
            {
                // The processed input contains a "value" entry if it is a date-time value, or "start" and
                // "end" entries if it is a date-time range.
                if (resolution.TryGetValue("value", out var dateString)
                    || resolution.TryGetValue("start", out dateString))
                {
                    if (DateTime.TryParse(dateString, out var candidate)
                        && earliest < candidate)
                    {
                        date = candidate.ToShortDateString();
                        return true;
                    }
                }
            }
        }

        message = "I'm sorry, please enter a date at least an hour out.";
    }
    catch
    {
        message = "I'm sorry, I could not interpret that as an appropriate date. Please enter a date at
least an hour out.";
    }

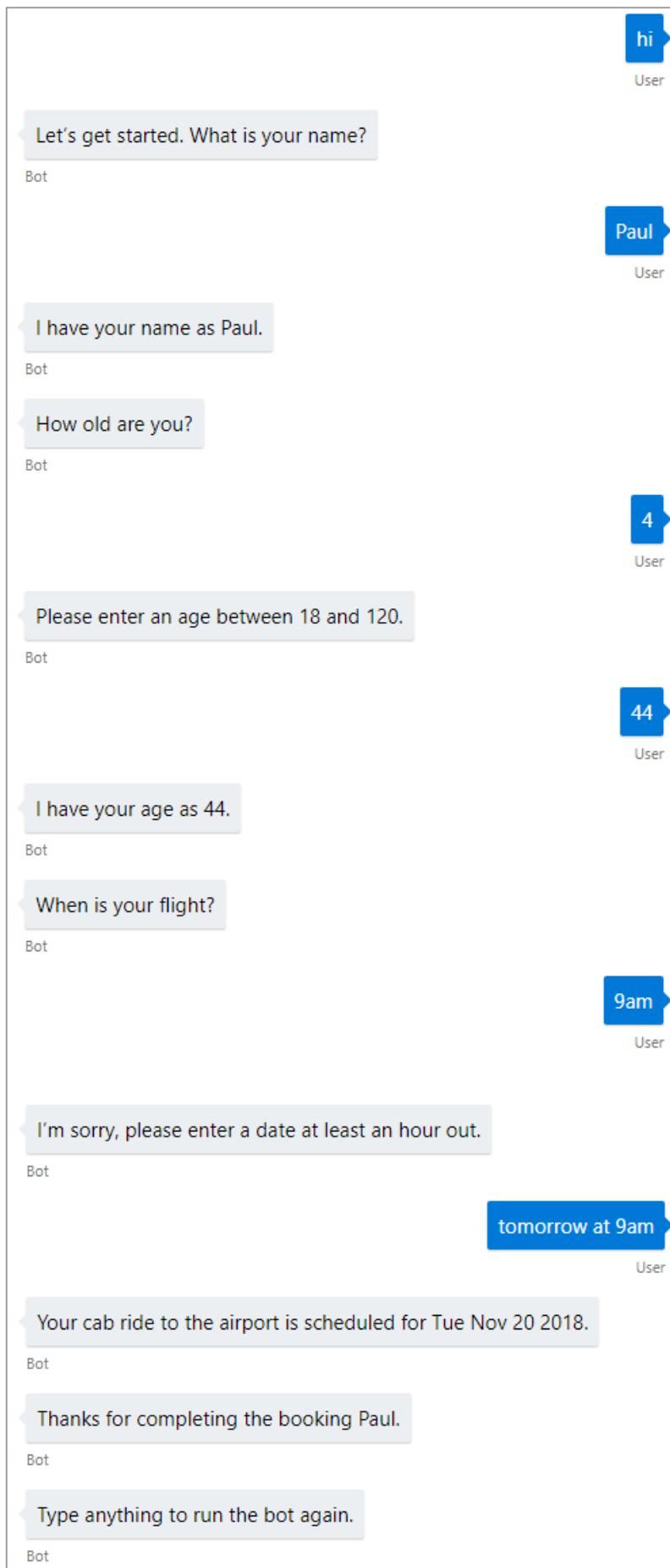
    return false;
}

```

## Локальная проверка бота

Скачайте и установите [Bot Framework Emulator](#) для локального тестирования бота.

1. Выполните этот пример на локальном компьютере. Дополнительные инструкции, включая примеры для [C#](#), [JS](#) и [Python](#), см. в файле README.
2. Протестируйте его с помощью эмулятора, как показано ниже.



## Дополнительные ресурсы

Библиотека [Dialogs](#) предоставляет классы, которые автоматизируют многие аспекты управления беседами.

## Следующий шаг

[Реализация процесса общения](#)

# Отправка приветственного сообщения пользователям

27.03.2021 • 13 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Основная цель создания любого бота — ведение осмысленного диалога с пользователем. Лучший способ достичь этой цели — сделать так, чтобы с момента присоединения к диалогу пользователь знал о предназначении вашего бота, его возможностях и причинах создания. В этой статье представлены примеры кода, которые помогут создать приветствие, отправляемое ботом пользователю.

## Предварительные требования

- Понимание [основных принципов работы ботов](#).
- Копия **примера с приветствием пользователя** для [C#](#), [JavaScript](#) или [Python](#). На примере кода в этой статье мы опишем, как отправлять приветственные сообщения.

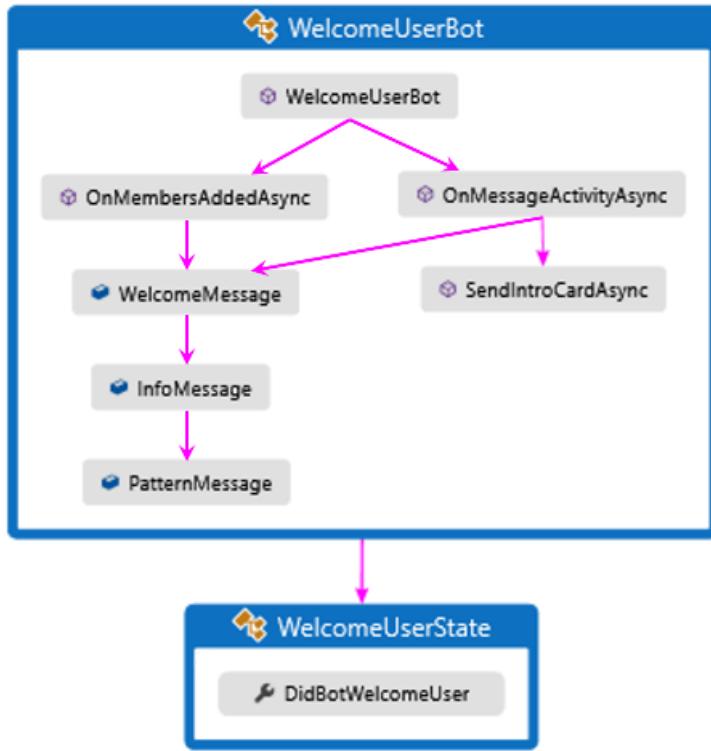
## Сведения о примере кода

Этот пример кода демонстрирует, как обнаруживать и приветствовать новых пользователей, которые впервые подключаются к боту. На следующей схеме показан поток логики для такого бота.

- [C#](#)
- [JavaScript](#)
- [Python](#)

Два основных события, отслеживаемые ботом:

- `OnMembersAddedAsync` вызывается каждый раз, когда к боту подключается новый пользователь;
- `OnMessageActivityAsync` вызывается каждый раз, когда принимаются данные от нового пользователя.



Каждый раз, когда подключается новый пользователь, бот предоставляет ему `WelcomeMessage`, `InfoMessage` и `PatternMessage`. При получении данных от нового пользователя бот проверяет `WelcomeUserState` и выясняет, имеет ли `DidBotWelcomeUser` значение *true*. Если это не так, он возвращает новому пользователю сообщение с приветствием.

## Создание состояния пользователя

- [C#](#)
- [JavaScript](#)
- [Python](#)

В момент запуска создается объект состояния пользователя, а в конструктор бота добавляются зависимости.

`Startup.cs` [!code-csharpdefine state]

`Bots\WelcomeUserBot.cs` [!code-csharpconsume state]

## Создание методов доступа к свойствам

- [C#](#)
- [JavaScript](#)
- [Python](#)

Теперь мы создадим метод доступа к свойству, предоставляющий обработчик `WelcomeUserState` в методе `OnMessageActivityAsync`. Затем мы вызовем метод `GetAsync`, чтобы получить ключ с правильной областью действия. Мы будем сохранять данные о состоянии пользователя после каждого цикла обработки введенных пользователем данных с помощью метода `SaveChangesAsync`.

`Bots\WelcomeUserBot.cs` [!code-csharpGet state]

```
// Save any state changes.
await _userState.SaveChangesAsync(turnContext, cancellationToken);
```

## Обнаружение и приветствие новых подключенных пользователей

- [C#](#)
- [JavaScript](#)
- [Python](#)

В `WelcomeUserBot` мы проверяем обновление действия с помощью `OnMembersAddedAsync()`, чтобы узнать, добавлен ли новый пользователь в диалог. Затем мы отправляем ему набор из трех приветственных сообщений: `WelcomeMessage`, `InfoMessage` и `PatternMessage`. Ниже приведен полный код для этого взаимодействия.

`Bots\WelcomeUserBot.cs` [[!code-csharpDefine messages](#)]

```
protected override async Task OnMembersAddedAsync(IList<ChannelAccount> membersAdded,
ITurnContext<IConversationUpdateActivity> turnContext, CancellationToken cancellationToken)
{
    foreach (var member in membersAdded)
    {
        if (member.Id != turnContext.Activity.Recipient.Id)
        {
            await turnContext.SendActivityAsync($"Hi there - {member.Name}. {WelcomeMessage}",
cancellationToken: cancellationToken);
            await turnContext.SendActivityAsync(InfoMessage, cancellationToken: cancellationToken);
            await turnContext.SendActivityAsync($"{LocaleMessage} Current locale is
'{turnContext.Activity.GetLocale()}'.", cancellationToken: cancellationToken);
            await turnContext.SendActivityAsync(PatternMessage, cancellationToken: cancellationToken);
        }
    }
}
```

## Приветствие нового пользователя и удаление первого входящего сообщения

- [C#](#)
- [JavaScript](#)
- [Python](#)

Важно помнить о том, что введенные пользователем данные могут содержать ценную информацию и что в разных каналах взаимодействие может происходить по-разному. Чтобы облегчить пользователю работу со всеми возможными каналами, нам нужно проверить флаг состояния `didBotWelcomeUser`. Если его значение равно `false`, мы не обрабатываем начальное сообщение, введенное пользователем. Вместо этого мы предоставляем пользователю начальное приветственное сообщение. Затем мы присваиваем значение `true` свойству `welcomedUserProperty`, которое хранится в `UserState`, и с этого момента наш код будет обрабатывать введенные этим пользователем данные от всех дополнительных действий сообщения.

`Bots\WelcomeUserBot.cs` [[!code-csharpDidBotWelcomeUser](#)]

```
// Save any state changes.
await _userState.SaveChangesAsync(turnContext, cancellationToken);
```

## Обработка дополнительных входных данных

Когда новый пользователь получает приветственное сообщение, введенные им данные оцениваются на каждом шаге диалога, и бот предоставляет ответы на основе контекста этих входных данных.

Следующий код демонстрирует логику принятия решений, которая используется для создания ответа.

- [C#](#)
- [JavaScript](#)
- [Python](#)

Ввод команд `intro` или `help` вызывает функцию `SendIntroCardAsync`, которая предоставляет пользователю информационную карту для имиджевого баннера. Этот код рассматривается в следующем разделе этой статьи.

`Bots\WelcomeUserBot.cs` [!code-csharp`SwitchOnUtterance`]

## Использование карты для имиджевого баннера с приветствием

Как упоминалось выше, в ответ на некоторые введенные пользователем данные создается *карта для имиджевого баннера*. См. подробнее в руководстве по [отправке карты с вводными сведениями](#). Ниже представлен код, который предоставляет ответ с картой для имиджевого баннера этого бота.

- [C#](#)
- [JavaScript](#)
- [Python](#)

`Bots\WelcomeUserBot.cs` [!code-csharp`SendHeroCardGreeting`]

## Тестирование бота

Скачайте и установите последнюю версию [Bot Framework Emulator](#).

1. Выполните этот пример на локальном компьютере. Если потребуются дополнительные инструкции, в файле README можно найти [пример кода на C#](#) или [пример кода на JS](#).
2. Примените эмулятор для тестирования бота, как показано ниже.

**Bot Framework Emulator - WelcomeUserBot-1**

File Edit View Conversation Help

BOT EXPLORER ENDPOINT SERVICES

Welcome Live Chat (<http://localhost...>)

Restart conversation Save transcript

http://localhost:3978/api/messages

Hi there - User. This is a simple Welcome Bot sample. This bot will introduce you to welcoming and greeting users. You can say 'intro' to see the introduction card. If you are running this bot in the Bot Framework Emulator, press the 'Start Over' button to simulate user joining a bot or a channel

You are seeing this message because the bot received at least one 'ConversationUpdate' event, indicating you (and possibly others) joined the conversation. If you are using the emulator, pressing the 'Start Over' button to trigger this event again. The specifics of the 'ConversationUpdate' event depends on the channel. You can read more information at: <https://aka.ms/about-botframework-welcome-user>

It is a good pattern to use this event to send general greeting to user, explaining what your bot can do. In this example, the bot handles 'hello', 'hi', 'help' and 'intro'. Try it now, type 'hi'

Just now

hello Just now

You are seeing this message because this was your first message ever to this bot.

It is a good practice to welcome the user and provide personal greeting. For example, welcome User.

Just now

hello Just now

You said hello. Just now

Type your message

INSPECTOR

Click on a log item in the panel below to inspect activity.

You can also inspect the JSON responses from your LUIS and QnA Maker services by selecting a "trace" activity. [Learn More](#).

LOG

```
[14:36:56] POST 201 directline.startConversation
[14:36:56] Emulator listening on http://localhost:55379
[14:36:56] ngrok listening on https://fdf3a3b7.ngrok.io
[14:36:56] ngrok traffic inspector:
[14:36:56] http://127.0.0.1:4040
[14:36:56] Will use ngrok for local addresses
[14:36:56] <-message Hi there - User. This is a simple Welcome Bot samp...
[14:36:56] POST 200 conversations.replyToActivity
[14:36:56] <-message You are seeing this message because the bot receiv...
[14:36:56] POST 200 conversations.replyToActivity
[14:36:56] <-message It is a good pattern to use this event to send gen...
[14:36:56] POST 200 conversations.replyToActivity
[14:37:03] >-message hello
[14:37:03] <-message You are seeing this message because this was your ...
[14:37:03] POST 200 conversations.replyToActivity
[14:37:03] <-message It is a good practice to welcome the user and prov...
[14:37:03] POST 200 conversations.replyToActivity
[14:37:03] POST 200 directline.postActivity
[14:37:07] >-message hello
[14:37:08] <-message You said hello.
[14:37:08] POST 200 conversations.replyToActivity
[14:37:08] POST 200 directline.postActivity
```

Проверьте карту для имиджевого баннера с приветствием.

**Bot Framework Emulator - WelcomeUserBot-1**

File Edit View Conversation Help

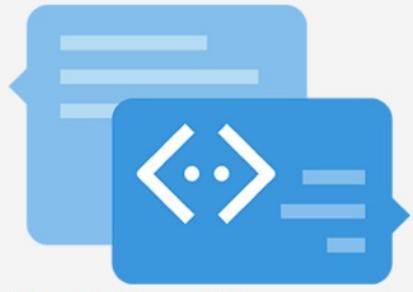
BOT EXPLORER ENDPOINT SERVICES

Welcome Live Chat (<http://localhost...>)

Restart conversation Save transcript

http://localhost:3978/api/messages

help A minute ago



Welcome to Bot Framework!

Welcome to Welcome Users bot sample! This Introduction card is a great way to introduce your Bot to the user and suggest some things to get them started. We use this opportunity to recommend a few next steps for learning more creating and deploying bots.

[Get an overview](#)

[Ask a question](#)

[Learn how to deploy](#)

A minute ago

Type your message

INSPECTOR

Click on a log item in the panel below to inspect activity.

You can also inspect the JSON responses from your LUIS and QnA Maker services by selecting a "trace" activity. [Learn More](#).

LOG

```
[14:35:12] POST 201 directline.startConversation
[14:35:12] Emulator listening on
[14:35:12] http://localhost:55379
[14:35:12] ngrok listening on
[14:35:12] https://fdf3a3b7.ngrok.io
[14:35:12] ngrok traffic inspector:
[14:35:12] http://127.0.0.1:4040
[14:35:12] Will use ngrok for local addresses
[14:35:12] <-message Hi there - User. This is a simple Welcome Bot samp...
[14:35:12] POST 200 conversations.replyToActivity
[14:35:12] <-message You are seeing this message because the bot receiv...
[14:35:12] POST 200 conversations.replyToActivity
[14:35:12] <-message It is a good pattern to use this event to send gen...
[14:35:12] POST 200 conversations.replyToActivity
[14:35:18] >-message help
[14:35:18] <-message You are seeing this message
```

## Дополнительные ресурсы

См. подробнее о [добавлении мультимедиа в сообщения](#).

## Дальнейшие действия

[Сбор вводимых пользователем данных](#)

# Отправка упреждающих уведомлений пользователям

27.03.2021 • 14 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Как правило, Bot отправляет пользователю сообщение непосредственно в ответ на получение сообщения от пользователя. Иногда Bot может потребоваться отправить *упреждающее сообщение*, сообщение в ответ на стимул, не исходящий от пользователя.

Упреждающие сообщения можно использовать в различных сценариях. Например, если пользователь ранее попросил бот отслеживать стоимость продукта, то бот может оповестить пользователя в случае, если стоимость продукта опустилась на 20 %. Если боту требуется какое-то время, чтобы сформировать ответ на вопрос пользователя, он может проинформировать пользователя о задержке и продолжить диалог. Когда бот сформирует ответ на вопрос, он передаст его пользователю.

## NOTE

В этой статье рассматриваются сведения о упреждающих сообщениях для программы-роботы в целом. Сведения об упреждающем сообщении в Microsoft Teams см. в следующих статьях:

- Пример использования **команд** Bot в [C#](#), [JavaScript](#) или [Python](#).
- Документация по Microsoft Teams о том, как [Отправить упреждающие сообщения](#).

## Требования

Прежде чем можно будет отправить упреждающее сообщение, для программы-робота требуется *ссылка на беседу*. Программа-робот может получить ссылку на беседу из любого действия, полученного от пользователя, но для этого обычно требуется, чтобы пользователь взаимодействовал с программой-роботом по крайней мере один раз, прежде чем Bot сможет отправить упреждающее сообщение.

Многие каналы запрещают пользователю-роботу обмен сообщениями, если пользователь не получил по крайней мере один из сообщений. Некоторые каналы допускают исключения. Например, канал команд позволяет роботу отправить упреждающее (или 1-на 1) сообщение отдельным пользователям в уже установленном коллективном сеансе, который включает робот.

## Предварительные требования

- Понимание [основных принципов работы ботов](#).
- Копия примера **упреждающего сообщения** в [C#](#), [JavaScript](#) или [Python](#). Этот пример используется в статье в качестве иллюстрации упреждающего обмена сообщениями.

## Сведения о примере с упреждающими сообщениями

Как правило, программа-робот в качестве приложения имеет несколько уровней:

- Веб-приложение, которое может принимать запросы HTTP и специально поддерживает конечную точку обмена сообщениями.
- Адаптер, который обрабатывает подключения к каналам.
- Обработчик для включения, обычно инкапсулированный в класс *Bot*, который обрабатывает

диалоговые причины для приложения-робота.

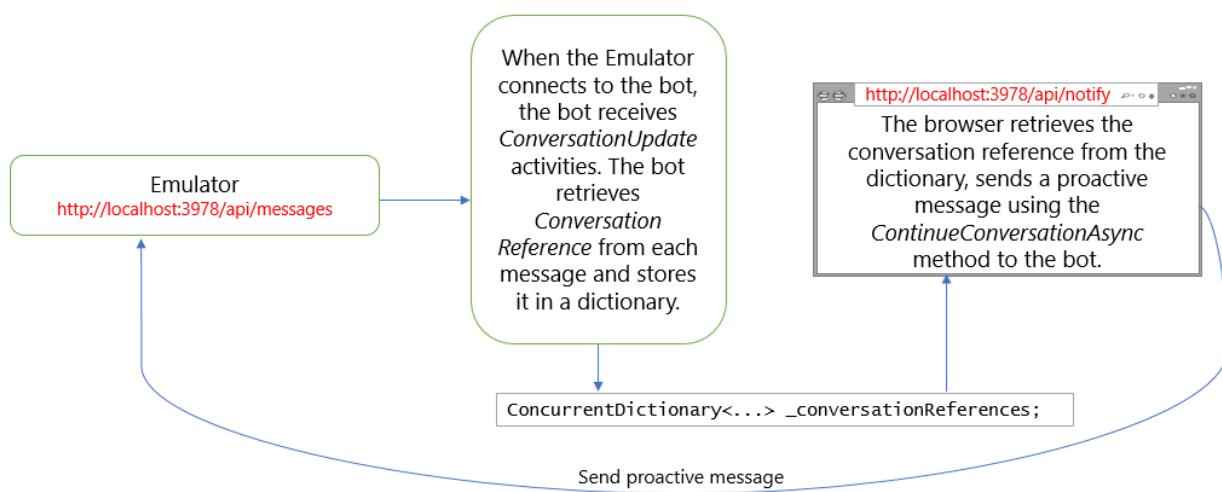
В ответ на входящее сообщение от пользователя приложение вызывает метод *обработки действия* адаптера, который создает контекст включения и выключения, вызывает его конвейер по промежуточного слоя, а затем вызывает обработчик «вращение робота».

Чтобы инициировать упреждающее сообщение, приложение-робот должно иметь возможность получать дополнительные входные данные. Логика приложения для инициации упреждающего сообщения выходит за рамки пакета SDK. В этом примере в дополнение к стандартной конечной точке *сообщений* используется конечная точка *уведомления*, которая служит для активации упреждающего включения.

В ответ на запрос GET для этой конечной точки уведомления приложение вызывает метод *продолжения диалога* адаптера, который работает аналогично методу *действия процесса*. Метод *продолжения диалога*:

- Принимает соответствующую ссылку на диалог для пользователя и метод обратного вызова, который используется для упреждающего включения.
- Создает действие события и переключается на контекст для упреждающего включения.
- Вызывает конвейер по промежуточного слоя адаптера.
- Вызывает предоставленный метод обратного вызова.
- Контекст включения использует ссылку диалога для отправки сообщений пользователю.

В примере есть программа-робот, конечная точка сообщений и дополнительная конечная точка уведомления, которая используется для отправки упреждающих сообщений пользователю, как показано на следующем рисунке.



## Получение и сохранение ссылки на беседу

Когда эмулятор подключается к роботу, Bot получает два действия обновления диалога. В обработчике бота для действий обновления беседы извлекается ссылка на беседу, которая сохраняется в словаре, как показано ниже.

- C#
- JavaScript
- Python

Bots\ProactiveBot.cs

```

private void AddConversationReference(Activity activity)
{
    var conversationReference = activity.GetConversationReference();
    _conversationReferences.AddOrUpdate(conversationReference.User.Id, conversationReference, (key,
    newValue) => conversationReference);
}

protected override Task OnConversationUpdateActivityAsync(ITurnContext<IConversationUpdateActivity>
turnContext, CancellationToken cancellationToken)
{
    AddConversationReference(turnContext.Activity as Activity);

    return base.OnConversationUpdateActivityAsync(turnContext, cancellationToken);
}

```

Ссылка на диалог включает в себя свойство *диалога*, описывающее диалог, в котором существует действие. Диалог включает свойство *User*, содержащее список пользователей, участвующих в диалоге, и свойство *URL-адреса службы*, которое указывает, где можно отправить ответы на текущее действие. Для отправки пользователям упреждающих сообщений нужно получить допустимую ссылку на беседу. (Для канала команд URL-адрес службы сопоставляется с региональным сервером.)

#### **NOTE**

В реальной системе ссылки на беседы следует хранить в базе данных, а не в объекте в памяти.

## Отправка упреждающих сообщений

Второй контроллер, контроллер *уведомления*, отвечает за отправку упреждающего сообщения пользователю. Для создания упреждающего сообщения используются следующие шаги.

- Извлекает ссылку для диалога, в который отправляется упреждающее сообщение.
- Вызывает метод *продолжения диалога* адаптера, предоставляя ссылку на диалог и делегат обработчика событий для использования. (Метод Continue CONVERSATION создает контекст для диалогового окна, на который указывает ссылка, а затем вызывает указанный делегат обработчика событий.)
- В делегате использует контекст переворачивания для отправки упреждающего сообщения. Здесь делегат определяется на контроллере уведомлений и отправляет упреждающее сообщение пользователю.

#### **NOTE**

Хотя каждый канал должен использовать стабильный URL-адрес службы, URL-адрес может меняться со временем. Дополнительные сведения об URL-адресе службы см. в разделах [Базовая структура действий](#) и [URL-адрес службы](#) схемы действий Bot Framework.

При изменении URL-адреса службы предыдущие ссылки на диалог больше не будут действительными, а вызовы для *продолжения диалога* будут вызывать ошибку или исключение. В этом случае программа Bot должна получить новую ссылку на беседу для пользователя, прежде чем она сможет снова отправить упреждающие сообщения.

- [C#](#)
- [JavaScript](#)
- [Python](#)

При каждом запросе страницы уведомления бота соответствующий контроллер извлекает из словаря ссылки на беседы. Этот контроллер выполняет методы `ContinueConversationAsync` и `BotCallback`, чтобы отправить упреждающее сообщение.

```
[Route("api/notify")]
[ApiController]
public class NotifyController : ControllerBase
{
    private readonly IBotFrameworkHttpAdapter _adapter;
    private readonly string _appId;
    private readonly ConcurrentDictionary<string, ConversationReference> _conversationReferences;

    public NotifyController(IBotFrameworkHttpAdapter adapter, IConfiguration configuration,
    ConcurrentDictionary<string, ConversationReference> conversationReferences)
    {
        _adapter = adapter;
        _conversationReferences = conversationReferences;
        _appId = configuration["MicrosoftAppId"] ?? string.Empty;
    }

    public async Task<IActionResult> Get()
    {
        foreach (var conversationReference in _conversationReferences.Values)
        {
            await ((BotAdapter)_adapter).ContinueConversationAsync(_appId, conversationReference,
BotCallback, default(CancellationToken));
        }

        // Let the caller know proactive messages have been sent
        return new ContentResult()
        {
            Content = "<html><body><h1>Proactive messages have been sent.</h1></body></html>",
            ContentType = "text/html",
            StatusCode = (int) HttpStatusCode.OK,
        };
    }

    private async Task BotCallback(ITurnContext turnContext, CancellationToken cancellationToken)
    {
        await turnContext.SendActivityAsync("proactive hello");
    }
}
```

Для отправки упреждающего сообщения адаптеру нужен идентификатор приложения бота. В рабочей среде для этого можно использовать реальный идентификатор приложения бота. Чтобы проверить Bot локально с помощью эмулятора, можно использовать пустую строку ("").

## Тестирование бота

- Установите [Bot Framework Emulator](#), если вы этого еще не сделали.
- Выполните этот пример на локальном компьютере.
- Запустите эмулятор и подключитесь к роботу.
- Загрузите страницу бота `api/notify`. В эмуляторе будет создано упреждающее сообщение.

## Дополнительные сведения

Помимо примера, используемого в этой статье, на сайте [GitHub](#) доступны дополнительные примеры.

### Рекомендации по проектированию

Если вы реализуете в боте упреждающие сообщения, не отправляйте несколько таких сообщений за короткий промежуток времени. Некоторые каналы ограничивают частоту, с которой бот может

отправлять сообщения пользователю, и отключают бот, если он нарушает эти ограничения.

Динамическое упреждающее сообщение — это самый простой тип упреждающих сообщений. Бот просто вставляет сообщение в диалог каждый раз, когда он активируется (вне зависимости от участия пользователя в отдельном разделе диалога с ботом в данный момент), и не пытается изменить диалог каким-либо способом.

Чтобы улучшить обработку уведомлений, рассмотрите другие варианты их интеграции в поток общения, например указав флаг в состоянии диалога или добавив уведомление в очередь.

### О профилактической активации

Метод *Continue CONVERSATION* использует ссылку диалога и обработчик обратного вызова для включения:

1. Создайте ход, в котором приложение-робот может отправить упреждающее сообщение. Адаптер создает `event` действие для этой очереди, а его имя имеет значение "континуеконверсацион".
2. Отправляйте через конвейер по промежуточного слоя адаптера.
3. Вызовите обработчик обратного вызова для выполнения пользовательской логики.

В примере **упреждающего сообщения** обработчик обратного вызова по включению определяется в контроллере уведомлений и отправляет сообщение непосредственно в диалог без отправки упреждающего действия через обработчик обычного включения Bot. Пример кода также не имеет доступа к состоянию Bot и не обновляет его в ходе упреждающего включения.

Многие программы-роботы имеют отслеживание состояния и используют состояние для управления диалогом на нескольких. Когда метод *Continue CONVERSATION* создает контекст «поворот», ему будет назначено правильное состояние пользователя и диалога, и вы можете интегрировать упреждающие преобразования в логику Bot. Если требуется, чтобы логика Bot сознала об упреждающем сообщении, у вас есть несколько вариантов для этого. Вы можете:

- Предоставьте обработчик поворачивания для программы-робота в качестве обработчика обратного вызова. Затем Bot получит действие события "Континуеконверсацион".
- Используйте обработчик обратного вызова, чтобы сначала добавить сведения в контекст для включения, а затем вызвать обработчик поворачивания.

В обоих случаях необходимо разработать логику Bot для обработки упреждающего события.

## Дальнейшие действия

[Реализация процесса общения](#)

# Управление долгосрочными операциями

27.03.2021 • 16 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Правильная обработка длительных операций — важный аспект надежной программы-робота. Когда служба Azure Bot отправляет действие в программу Bot из канала, ожидается, что программа-робот будет обрабатывать действие быстро. Если программа-робот не выполняет операцию в течение 10 – 15 секунд, в зависимости от канала, служба Azure Bot истечет время ожидания и отправит обратно клиенту а `504:GatewayTimeout`, как описано в разделе [как программы-роботы работают](#).

В этой статье описывается использование внешней службы для выполнения операции и уведомления программы-робота о ее завершении.

## Предварительные требования

- Если у вас еще нет подписки Azure, создайте [бесплатную](#) учетную запись Azure, прежде чем начинать работу.
- Знакомство с [запросами в каскадных диалоговых окнах](#) и [упреждающего обмена сообщениями](#).
- Знакомство с [хранилищем очередей Azure](#) и [скриптом C# для функций Azure](#).
- Копия образца [многострочной подсказки](#) в [C#](#).

## Об этом примере

Эта статья начинается с примера кода многофункциональной программы-робота и добавляет код для выполнения длительных операций. В нем также показано, как реагировать на пользователя после завершения операции. В обновленном примере:

- Программа-робот запрашивает у пользователя выполняемую длительную операцию.
- Bot получает действие от пользователя и определяет, какую операцию выполнить.
- Bot уведомляет пользователя, что операция займет некоторое время и отправит операцию в функцию C#.
  - Bot сохраняет состояние, указывая на выполнение операции.
  - Во время выполнения операции программа Bot реагирует на сообщения от пользователя, уведомляя их о том, что операция все еще выполняется.
  - Функции Azure управляют длительной операцией и отправляют `event` действие в Bot, уведомляя его о завершении операции.
- Bot возобновляет диалог и отправляет упреждающее сообщение, чтобы уведомить пользователя о завершении операции. Затем Bot очищает состояние операции, упомянутое ранее.

В этом примере определяется `LongOperationPrompt` класс, производный от абстрактного `ActivityPrompt` класса. Когда `LongOperationPrompt` очередь обрабатывает обрабатываемое действие, оно включает в себя выбора пользователя в свойстве `значения действия`. Затем это действие используется в функциях Azure, изменяется и упаковывается в другое `event` действие перед отправкой обратно в Bot с помощью клиента прямой линии. В Bot действие события используется для возобновления диалога путем вызова метода `продолжения диалога` адаптера. Затем стек диалоговых окон загружается и `LongOperationPrompt` завершается.

В этой статье рассказывается о различных технологиях. Ссылки на соответствующие статьи см. в разделе [Дополнительные ресурсы](#).

## Создание учетной записи хранения Azure

Создайте учетную запись хранения Azure и получите строку подключения. Необходимо добавить строку подключения в файл конфигурации программы-робота.

Дополнительные сведения см. в статьях [Создание учетной записи хранения](#) и [копирование учетных данных из портал Azure](#).

## Создание службы "Регистрация каналов бота"

- Перед созданием регистрации настройте ngrok и получите URL-адрес, который будет использоваться в качестве *конечной точки обмена сообщениями* Bot во время локальной отладки. Конечная точка обмена сообщениями будет содержать URL-адрес перенаправления HTTPS с `/api/messages/` добавлением. Обратите внимание, что портом по умолчанию для New программы-роботы является 3978.

Дополнительные сведения см. в разделе [Отладка Bot с помощью ngrok](#).

- Создайте регистрацию каналов Bot в портал Azure или с помощью Azure CLI. Настройте конечную точку обмена сообщениями Bot, которая была создана с помощью ngrok. После создания ресурса регистрации каналов Bot получите идентификатор и пароль приложения-робота Microsoft. Включите прямой канал линии и получите секрет прямой строки. Вы добавите их в код Bot и функцию C#.

Дополнительные сведения см. в статьях [Управление программой-роботом](#) и [Подключение программы-робота к прямой линии](#).

## Создание функции C#

- Создание приложения "функции Azure" на основе стека .Net Core среды выполнения.

Дополнительные сведения см. в разделе [Создание приложения-функции](#) и Справочник по [скрипту C# для функций Azure](#).

- Добавьте `DirectLineSecret` параметр приложения в приложение-функция.

Дополнительные сведения см. в статье [Управление приложением функции](#).

- В приложение-функция добавьте функцию на основе [шаблона хранилища очередей Azure](#).

Задайте имя нужной очереди и выберите `Azure Storage Account` созданное на предыдущем шаге. Это имя очереди также будет помещено в файл `appsettings.js` Bot в файле.

- Добавьте в функцию файл `Function.proj`.

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>netstandard2.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.Bot.Connector.DirectLine" Version="3.0.2" />
    <PackageReference Include="Microsoft.Rest.ClientRuntime" Version="2.3.4" />
  </ItemGroup>
</Project>
```

- Обновите `Run.csx`, используя следующий код:

```

#r "Newtonsoft.Json"

using System;
using System.Net.Http;
using System.Text;
using Newtonsoft.Json;
using Microsoft.Bot.Connector.DirectLine;
using System.Threading;

public static async Task Run(string queueItem, ILogger log)
{
    log.LogInformation($"C# Queue trigger function processing");

    JsonSerializerSettings jsonSettings = new JsonSerializerSettings() { NullValueHandling = NullValueHandling.Ignore };
    var originalActivity = JsonConvert.DeserializeObject<Activity>(queueItem, jsonSettings);
    // Perform long operation here...
    System.Threading.Thread.Sleep(TimeSpan.FromSeconds(15));

    if(originalActivity.Value.ToString().Equals("option 1", CompareOptions.OrdinalIgnoreCase))
    {
        originalActivity.Value = " (Result for long operation one!)";
    }
    else if(originalActivity.Value.ToString().Equals("option 2", CompareOptions.OrdinalIgnoreCase))
    {
        originalActivity.Value = " (A different result for operation two!)";
    }

    originalActivity.Value = "LongOperationComplete:" + originalActivity.Value;
    var responseActivity = new Activity("event");
    responseActivity.Value = originalActivity;
    responseActivity.Name = "LongOperationResponse";
    responseActivity.From = new ChannelAccount("GenerateReport", "AzureFunction");

    var directLineSecret = Environment.GetEnvironmentVariable("DirectLineSecret");
    using(DirectLineClient client = new DirectLineClient(directLineSecret))
    {
        var conversation = await client.Conversations.StartConversationAsync();
        await client.Conversations.PostActivityAsync(conversation.ConversationId, responseActivity);
    }

    log.LogInformation($"Done...");
}

```

## Создание бота

- Начните с копии примера [множественной командной строки C#](#).
- Добавьте в проект пакет NuGet для [Azure. Storage. Queues](#).
- Добавьте строку подключения для созданной ранее учетной записи хранения Azure и имя очереди хранилища в файл конфигурации программы-робота.

Убедитесь, что имя очереди совпадает с именем, использованным для создания функции триггера очереди ранее. Кроме того, добавьте значения для `MicrosoftAppId` `MicrosoftAppPassword` свойств и, созданных ранее при создании ресурса регистрации каналов Bot.

`appsettings.json`

```
{  
    "MicrosoftAppId": "<your-bot-app-id>",  
    "MicrosoftAppPassword": "<your-bot-app-password>",  
    "StorageQueueName": "<your-azure-storage-queue-name>",  
    "QueueStorageConnectionString": "<your-storage-connection-string>"  
}
```

- Добавьте `IConfiguration` параметр в `DialogBot.cs`, чтобы получить `MicrosoftAppId`. Также добавьте `OnEventActivityAsync` обработчик для объекта `LongOperationResponse` из функции Azure.

`Bots\DialogBot.cs`

```

protected readonly IStatePropertyAccessor<DialogState> DialogState;
protected readonly Dialog Dialog;
protected readonly BotState ConversationState;
protected readonly ILogger Logger;
private readonly string _botId;

/// <summary>
/// Create an instance of <see cref="DialogBot{T}"/>.
/// </summary>
/// <param name="configuration"><see cref=" IConfiguration"/> used to retrieve MicrosoftAppId
/// which is used in ContinueConversationAsync.</param>
/// <param name="conversationState"><see cref=" ConversationState"/> used to store the DialogStack.
/// </param>
/// <param name="dialog">The RootDialog for this bot.</param>
/// <param name="logger"><see cref=" ILogger"/> to use.</param>
public DialogBot(IConfiguration configuration, ConversationState conversationState, T dialog,
ILogger<DialogBot<T>> logger)
{
    _botId = configuration["MicrosoftAppId"] ?? Guid.NewGuid().ToString();
    ConversationState = conversationState;
    Dialog = dialog;
    Logger = logger;
    DialogState = ConversationState.CreateProperty<DialogState>(nameof(DialogState));
}

public override async Task OnTurnAsync(ITurnContext turnContext, CancellationToken cancellationToken
= default)
{
    await base.OnTurnAsync(turnContext, cancellationToken);

    // Save any state changes that might have occurred during the turn.
    await ConversationState.SaveChangesAsync(turnContext, false, cancellationToken);
}

protected override async Task OnEventActivityAsync(ITurnContext<IEventActivity> turnContext,
CancellationToken cancellationToken)
{
    // The event from the Azure Function will have a name of 'LongOperationResponse'
    if (turnContext.Activity.ChannelId == Channels.Directline && turnContext.Activity.Name ==
"LongOperationResponse")
    {
        // The response will have the original conversation reference activity in the .Value
        // This original activity was sent to the Azure Function via Azure.Storage.Queues in
AzureQueuesService.cs.
        var continueConversationActivity = (turnContext.Activity.Value as
JObject)?ToObject<Activity>();
        await turnContext.Adapter.ContinueConversationAsync(_botId,
continueConversationActivity.GetConversationReference(), async (context, cancellationToken) =>
{
            Logger.LogInformation("Running dialog with Activity from LongOperationResponse.");

            // ContinueConversationAsync resets the .Value of the event being continued to Null,
            // so change it back before running the dialog stack. (The .Value contains the response
            // from the Azure Function)
            context.Activity.Value = continueConversationActivity.Value;
            await Dialog.RunAsync(context, DialogState, cancellationToken);

            // Save any state changes that might have occurred during the inner turn.
            await ConversationState.SaveChangesAsync(context, false, cancellationToken);
        }, cancellationToken);
    }
    else
    {
        await base.OnEventActivityAsync(turnContext, cancellationToken);
    }
}

```

5. Создайте службу очередей Azure, чтобы поставить в очередь действия, которые необходимо обработать.

AzureQueuesService.cs

```

///<summary>
/// Service used to queue messages to an Azure.Storage.Queues.
///</summary>
public class AzureQueuesService
{
    private static JsonSerializerSettings jsonSettings = new JsonSerializerSettings()
    {
        Formatting = Formatting.Indented,
        NullValueHandling = NullValueHandling.Ignore
    };

    private bool _createQueueIfNotExists = true;
    private readonly QueueClient _queueClient;

    ///<summary>
    /// Creates a new instance of <see cref="AzureQueuesService"/>.
    ///</summary>
    ///<param name="config"><see cref="IConfiguration"/> used to retrieve
    /// StorageQueueName and QueueStorageConnection from appsettings.json.</param>
    public AzureQueuesService(IConfiguration config)
    {
        var queueName = config["StorageQueueName"];
        var connectionString = config["QueueStorageConnection"];

        _queueClient = new QueueClient(connectionString, queueName);
    }

    ///<summary>
    /// Queue and Activity, with option in the Activity.Value to Azure.Storage.Queues
    ///
    ///<seealso href="https://github.com/microsoft/botbuilder-dotnet/blob/master/libraries/Microsoft.Bot.Builder.Azure/Queues/ContinueConversationLater.cs"/>
    ///</summary>
    ///<param name="referenceActivity">Activity to queue after a call to GetContinuationActivity.
    </param>
    ///<param name="option">The option the user chose, which will be passed within the .Value of the
    activity queued.</param>
    ///<param name="cancellationToken">Cancellation token for the async operation.</param>
    ///<returns>Queued <see cref="Azure.Storage.Queues.Models.SendReceipt.MessageId"/>.</returns>
    public async Task<string> QueueActivityToProcess(Activity referenceActivity, string option,
CancellationToken cancellationToken)
    {
        if (_createQueueIfNotExists)
        {
            _createQueueIfNotExists = false;
            await _queueClient.CreateIfNotExistsAsync().ConfigureAwait(false);
        }

        // create ContinuationActivity from the conversation reference.
        var activity = referenceActivity.GetConversationReference().GetContinuationActivity();
        // Pass the user's choice in the .Value
        activity.Value = option;

        var message =
Convert.ToBase64String(Encoding.UTF8.GetBytes(JsonConvert.SerializeObject(activity, jsonSettings)));

        // Aend ResumeConversation event, it will get posted back to us with a specific value, giving
us
        // the ability to process it and do the right thing.
        var receipt = await _queueClient.SendMessageAsync(message,
cancellationToken).ConfigureAwait(false);
        return receipt.Value.MessageId;
    }
}

```

## Диалоговые окна

Удалите старое диалоговое окно и замените его новыми диалоговыми окнами для поддержки операций.

1. Удалите файл UserProfileDialog.cs .
2. Добавьте диалоговое окно настраиваемого запроса, запрашивающее пользователя о выполняемой операции.

**диалог\лонгоператионпромпт.кс**

```

///<summary>
///<see cref="ActivityPrompt"/> implementation which will queue an activity,
///along with the <see cref="LongOperationPromptOptions.LongOperationOption"/>,
///and wait for an <see cref="ActivityTypes.Event"/> with name of "ContinueConversation"
///and Value containing the text: "LongOperationComplete".
///
///The result of this prompt will be the received Event Activity, which is sent by
///the Azure Function after it finishes the long operation.
///</summary>
public class LongOperationPrompt : ActivityPrompt
{
    private readonly AzureQueuesService _queueService;

    ///<summary>
    ///Create a new instance of <see cref="LongOperationPrompt"/>.
    ///</summary>
    ///<param name="dialogId">Id of this <see cref="LongOperationPrompt"/>.</param>
    ///<param name="validator">Validator to use for this prompt.</param>
    ///<param name="queueService"><see cref="AzureQueuesService"/> to use for Enqueuing the activity
    to process.</param>
    public LongOperationPrompt(string dialogId, PromptValidator<Activity> validator,
AzureQueuesService queueService)
        : base(dialogId, validator)
    {
        _queueService = queueService;
    }

    public async override Task<DialogTurnResult> BeginDialogAsync(DialogContext dc, object options,
CancellationToken cancellationToken = default)
    {
        // When the dialog begins, queue the option chosen within the Activity queued.
        await _queueService.QueueActivityToProcess(dc.Context.Activity, (options as
LongOperationPromptOptions).LongOperationOption, cancellationToken);

        return await base.BeginDialogAsync(dc, options, cancellationToken);
    }

    protected override Task<PromptRecognizerResult<Activity>> OnRecognizeAsync(ITurnContext
turnContext, IDictionary<string, object> state, PromptOptions options, CancellationToken
cancellationToken = default)
    {
        var result = new PromptRecognizerResult<Activity>() { Succeeded = false };

        if(turnContext.Activity.Type == ActivityTypes.Event
            && turnContext.Activity.Name == "ContinueConversation"
            && turnContext.Activity.Value != null
            // Custom validation within LongOperationPrompt.
            // 'LongOperationComplete' is added to the Activity.Value in the Queue consumer (See:
        Azure Function)
            && turnContext.Activity.Value.ToString().Contains("LongOperationComplete",
System.StringComparison.InvariantCultureIgnoreCase))
        {
            result.Succeeded = true;
            result.Value = turnContext.Activity;
        }

        return Task.FromResult(result);
    }
}

```

3. Добавьте класс параметров prompt для пользовательского запроса.

#### **диалог\лонгоператионпромптооптионс.кц**

```

/// <summary>
/// Options sent to <see cref="LongOperationPrompt"/> demonstrating how a value
/// can be passed along with the queued activity.
/// </summary>
public class LongOperationPromptOptions : PromptOptions
{
    /// <summary>
    /// This is a property sent through the Queue, and is used
    /// in the queue consumer (the Azure Function) to differentiate
    /// between long operations chosen by the user.
    /// </summary>
    public string LongOperationOption { get; set; }
}

```

4. Добавьте диалоговое окно, которое использует настраиваемый запрос для получения выбора пользователя и инициации длительной операции.

### **диалог\лонгоператиондиалог.кс**

```

/// <summary>
/// This dialog demonstrates how to use the <see cref="LongOperationPrompt"/>.
///
/// The user is provided an option to perform any of three long operations.
/// Their choice is then sent to the <see cref="LongOperationPrompt"/>.
/// When the prompt completes, the result is received as an Activity in the
/// final Waterfall step.
/// </summary>
public class LongOperationDialog : ComponentDialog
{
    public LongOperationDialog(AzureQueuesService queueService)
        : base(nameof(LongOperationDialog))
    {
        // This array defines how the Waterfall will execute.
        var waterfallSteps = new WaterfallStep[]
        {
            OperationTimeStepAsync,
            LongOperationStepAsync,
            OperationCompleteStepAsync,
        };

        // Add named dialogs to the DialogSet. These names are saved in the dialog state.
        AddDialog(new WaterfallDialog(nameof(WaterfallDialog), waterfallSteps));
        AddDialog(new LongOperationPrompt(nameof(LongOperationPrompt), (vContext, token) =>
        {
            return Task.FromResult(vContext.Recognized.Succeeded);
        }, queueService));
        AddDialog(new ChoicePrompt(nameof(ChoicePrompt)));

        // The initial child Dialog to run.
        InitialDialogId = nameof(WaterfallDialog);
    }

    private static async Task<DialogTurnResult> OperationTimeStepAsync(WaterfallStepContext
stepContext, CancellationToken cancellationToken)
    {
        // WaterfallStep always finishes with the end of the Waterfall or with another dialog; here
it is a Prompt Dialog.
        // Running a prompt here means the next WaterfallStep will be run when the user's response is
received.
        return await stepContext.PromptAsync(nameof(ChoicePrompt),
            new PromptOptions
            {
                Prompt = MessageFactory.Text("Please select a long operation test option."),
                Choices = ChoiceFactory.ToChoices(new List<string> { "option 1", "option 2", "option
3" }),
            }.WithCancellation(cancellationToken));
    }
}

```

```

        , cancellationToken);
    }

    private static async Task<DialogTurnResult> LongOperationStepAsync(WaterfallStepContext
stepContext, CancellationToken cancellationToken)
{
    var value = ((FoundChoice)stepContext.Result).Value;
    stepContext.Values["longOperationOption"] = value;

    var prompt = MessageFactory.Text("...one moment please....");
    // The reprompt will be shown if the user messages the bot while the long operation is being
performed.
    var retryPrompt = MessageFactory.Text($"Still performing the long operation: {value} ... (is
the Azure Function executing from the queue?)");
    return await stepContext.PromptAsync(nameof(LongOperationPrompt),
        new LongOperationPromptOptions
    {
        Prompt = prompt,
        RetryPrompt = retryPrompt,
        LongOperationOption = value,
    }, cancellationToken);
}

private static async Task<DialogTurnResult> OperationCompleteStepAsync(WaterfallStepContext
stepContext, CancellationToken cancellationToken)
{
    stepContext.Values["longOperationResult"] = stepContext.Result;
    await stepContext.Context.SendActivityAsync(MessageFactory.Text($"Thanks for waiting. {(
stepContext.Result as Activity).Value}"), cancellationToken);

    // Start over by replacing the dialog with itself.
    return await stepContext.ReplaceDialogAsync(nameof(WaterfallDialog), null,
cancellationToken);
}
}

```

## Регистрация служб и диалогового окна

В `Startup.cs` обновите `ConfigureServices` метод, чтобы зарегистрировать `LongOperationDialog` и добавить `AzureQueuesService`.

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers().AddNewtonsoftJson();

    // Create the Bot Framework Adapter with error handling enabled.
    services.AddSingleton<IBotFrameworkHttpAdapter, AdapterWithErrorHandler>();

    // In production, this should be a persistent storage provider.
    services.AddSingleton<IStorage>(new MemoryStorage());

    // Create the Conversation state. (Used by the Dialog system itself.)
    services.AddSingleton<ConversationState>();

    // The Dialog that will be run by the bot.
    services.AddSingleton<LongOperationDialog>();

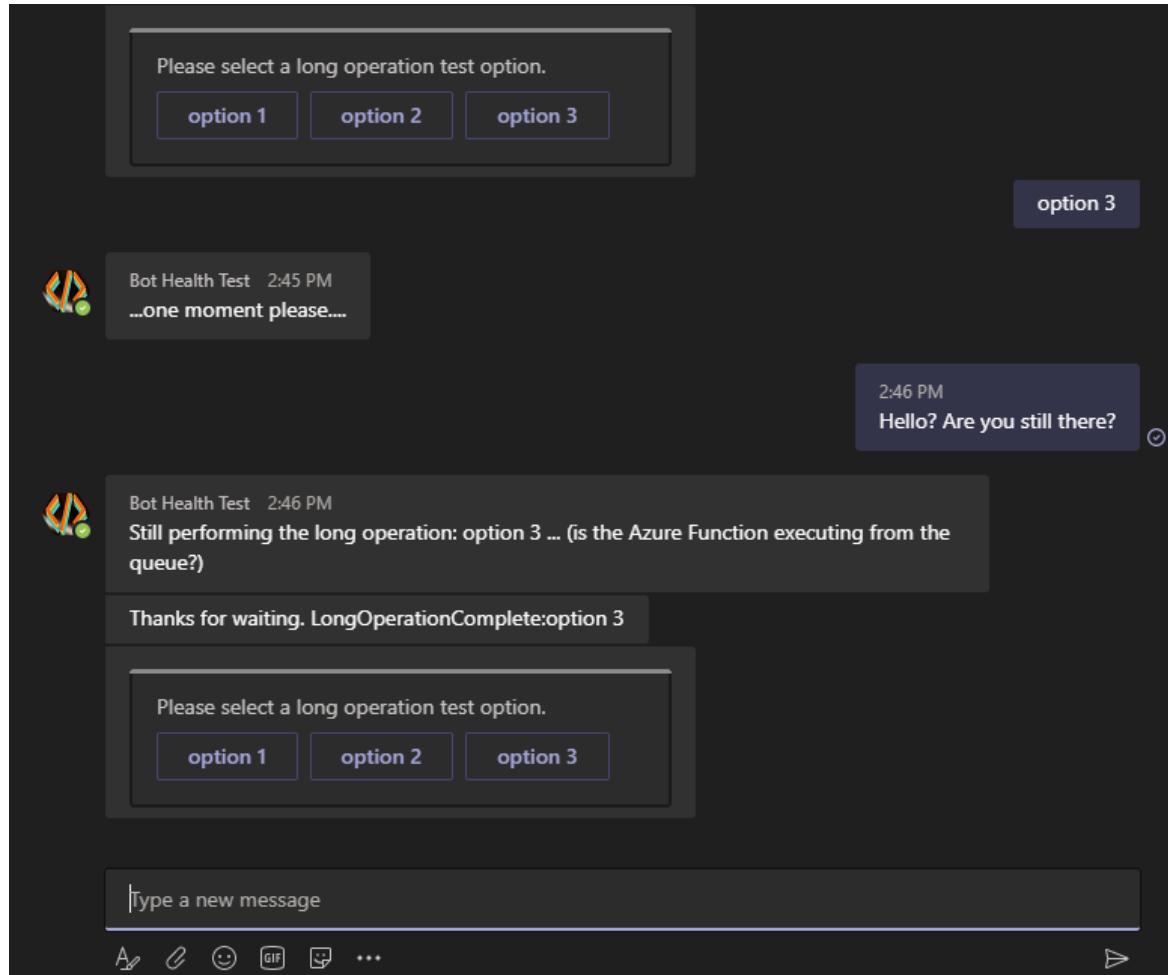
    // Service used to queue into Azure.Storage.Queues
    services.AddSingleton<AzureQueuesService>();

    // Create the bot as a transient. In this case the ASP Controller is expecting an IBot.
    services.AddTransient<IBot, DialogBot<LongOperationDialog>>();
}

```

## Тестирование бота

- Установите [Bot Framework Emulator](#), если вы этого еще не сделали.
- Выполните этот пример на локальном компьютере.
- Запустите эмулятор, подключитесь к боту и отправьте несколько сообщений, как показано ниже.



## Дополнительные ресурсы

СРЕДСТВО ИЛИ КОМПОНЕНТ	РЕСУРСЫ
Функции Azure	<a href="#">Создание приложения-функции</a> <a href="#">Скрипт C# для функций Azure</a> <a href="#">Управление приложением функции</a>
Портал Azure	<a href="#">Управление ботом</a> <a href="#">Подключение бота к Direct Line</a>
Хранилище Azure	<a href="#">Хранилище очередей Azure</a> <a href="#">создать учетную запись хранения;</a> <a href="#">Копирование учетных данных с портала Azure</a> <a href="#">Использование очередей</a>
Основные сведения о ботах	<a href="#">Принципы работы бота</a> <a href="#">Запросы в каскадных диалоговых окнах</a> <a href="#">Упреждающий обмен сообщениями</a>
ngrok	<a href="#">Отладка программы-робота с помощью ngrok</a>

# Реализация процесса общения

27.03.2021 • 34 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Сбор данных путем размещения вопросов — это один из основных способов взаимодействия между ботом и пользователями. Библиотека диалогов предоставляет полезные встроенные функции, такие как классы `запросов`, которые позволяют легко задавать вопросы и проверять ответ, чтобы убедиться в том, что он соответствует определенному типу данных или удовлетворяет пользовательские правила проверки.

С помощью библиотеки диалогов можно управлять простыми и сложными процессами общения. В самом простом сценарии взаимодействия бот выполняет фиксированную последовательность действий и завершает диалог. Диалоговое окно полезно, когда программа Bot должна собирать информацию от пользователя.

В этой статье показано, как реализовать простой поток диалога путем создания запросов и вызова их из диалогового окна каскада.

## TIP

Примеры написания запросов без использования библиотеки диалогов см. в статье о [создании собственных запросов для сбора вводимых пользовательских данных](#).

## Предварительные требования

- Понимание [основных принципов работы ботов, управления состоянием](#) и [библиотеки диалогов](#).
- Копия образца с [несколькими инструкциями по включению](#) в [C#](#), [JavaScript](#) или [Python](#).

## Об этом примере

В примере с несколькими интерактивными запросами используется диалоговое окно каскадом, несколько запросов и диалоговое окно компонента для создания простого взаимодействия, предлагающего пользователю серию вопросов. Код диалога циклически перебирает следующие действия:

ШАГИ	ТИП ЗАПРОСА
Запрос к пользователю о режиме транспортировки	Запрос выбора
Запрос имени пользователя	Запрос текста
Запрос к пользователю, готов ли он указать свой возраст	Запрос подтверждения
Если они ответили да, запросите их возраст.	Номер запроса с проверкой на принятие только возраста больше 0 и меньше 150
Если пользователь не использует Microsoft Teams, запросите изображение профиля	Запрос на вложение с проверкой для разрешения отсутствующего вложения

ШАГИ	ТИП ЗАПРОСА
Спросите, имеет ли собранная информация значение "OK".	Повторный запрос подтверждения

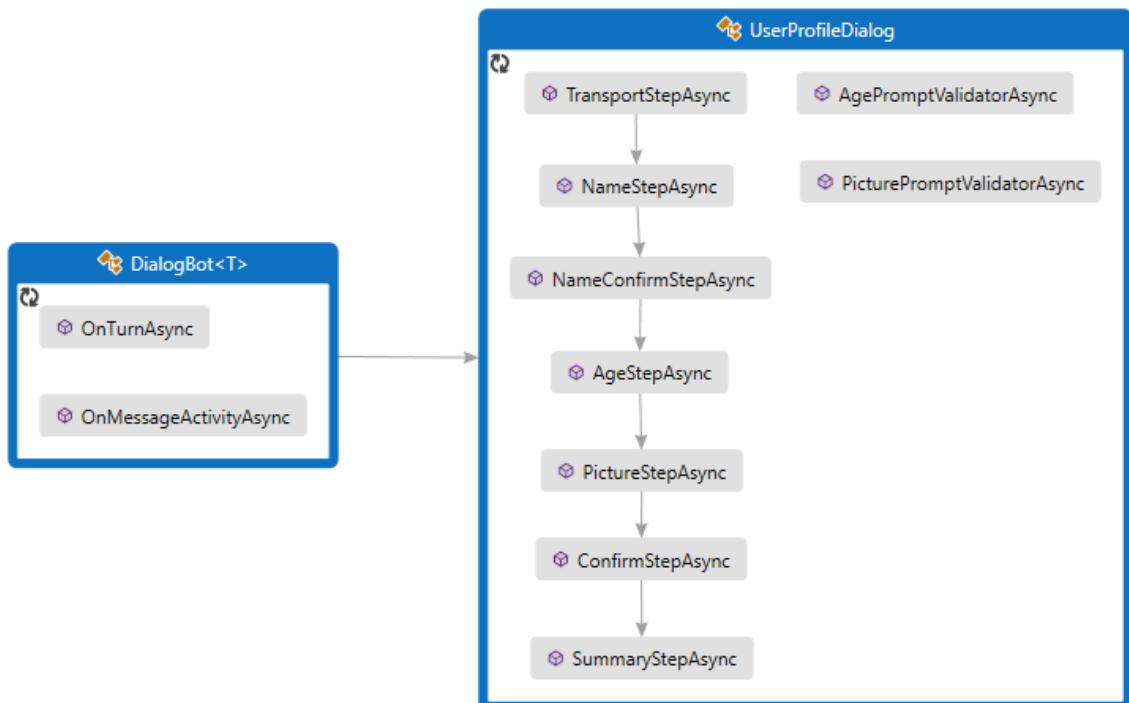
И наконец, если получен положительный ответ, отображается вся собранная информация. В противном случае пользователь получает сообщение о том, что данные не будут сохранены.

## Создание главного диалога

- C#
- JavaScript
- Python

Чтобы использовать диалоги, установите пакет NuGet `Microsoft.Bot.Builder.Dialogs`.

Бот взаимодействует с пользователем через `UserProfileDialog`. При создании `DialogBot` класса Bot в `UserProfileDialog` качестве главного диалогового окна задается. Затем бот применяет вспомогательный метод `Run` для доступа к этому диалогу.



### Dialogs\UserProfileDialog.cs

Сначала создайте объект `UserProfileDialog`, производный от `ComponentDialog` класса, и выполните семь шагов.

В конструкторе `UserProfileDialog` создайте каскадные шаги, запросы и каскадный диалог, затем добавьте их в набор диалогов. Запросы должны находиться в том же наборе диалогов, в котором они используются.

```

public UserProfileDialog(UserState userState)
    : base(nameof(UserProfileDialog))
{
    _userManager = userState.CreateProperty<UserProfile>("UserProfile");

    // This array defines how the Waterfall will execute.
    var waterfallSteps = new WaterfallStep[]
    {
        TransportStepAsync,
        NameStepAsync,
        NameConfirmStepAsync,
        AgeStepAsync,
        PictureStepAsync,
        ConfirmStepAsync,
        SummaryStepAsync,
    };

    // Add named dialogs to the DialogSet. These names are saved in the dialog state.
    AddDialog(new WaterfallDialog(nameof(WaterfallDialog), waterfallSteps));
    AddDialog(new TextPrompt(nameof(TextPrompt)));
    AddDialog(new NumberPrompt<int>(nameof(NumberPrompt<int>), AgePromptValidatorAsync));
    AddDialog(new ChoicePrompt(nameof(ChoicePrompt)));
    AddDialog(new ConfirmPrompt(nameof(ConfirmPrompt)));
    AddDialog(new AttachmentPrompt(nameof(AttachmentPrompt), PicturePromptValidatorAsync));

    // The initial child Dialog to run.
    InitialDialogId = nameof(WaterfallDialog);
}

```

Затем добавьте шаги, которые диалоговое окно использует для запроса входных данных. Чтобы использовать запрос, вызовите его из любого шага диалога и получите результат на следующем шаге с помощью `stepContext.Result`. Запросы данных, по сути, являются диалогами из двух этапов. Во первых, в командной строке запрашиваются входные данные. Затем он возвращает допустимое значение или начинается с начала с момента повторного запроса, пока не получит допустимые входные данные.

Из каскадного шага следует всегда возвращать ненулевое значение `DialogTurnResult`. В противном случае диалоговое окно может работать не так, как задумано. Ниже приведена реализация для `NameStepAsync` в диалоговом окне каскада.

```

private static async Task<DialogTurnResult> NameStepAsync(WaterfallStepContext stepContext,
    CancellationToken cancellationToken)
{
    stepContext.Values["transport"] = ((FoundChoice)stepContext.Result).Value;

    return await stepContext.PromptAsync(nameof(TextPrompt), new PromptOptions { Prompt =
        MessageFactory.Text("Please enter your name.") }, cancellationToken);
}

```

В `AgeStepAsync` Укажите запрос на повторную попытку, когда входные данные пользователя не проверяются, либо потому, что он находится в формате, который не может анализировать запрос, либо если входные данные не прошли проверку. Если строка повторного запроса не указана, в таких случаях пользователю будет повторно предоставляться исходный текст запроса для получения входных данных.

```

private async Task<DialogTurnResult> AgeStepAsync(WaterfallStepContext stepContext, CancellationToken cancellationToken)
{
    if ((bool)stepContext.Result)
    {
        // User said "yes" so we will be prompting for the age.
        // WaterfallStep always finishes with the end of the Waterfall or with another dialog; here it is a
        Prompt Dialog.
        var promptOptions = new PromptOptions
        {
            Prompt = MessageFactory.Text("Please enter your age."),
            RetryPrompt = MessageFactory.Text("The value entered must be greater than 0 and less than
150."),
        };

        return await stepContext.PromptAsync(nameof(NumberPrompt<int>), promptOptions, cancellationToken);
    }
    else
    {
        // User said "no" so we will skip the next step. Give -1 as the age.
        return await stepContext.NextAsync(-1, cancellationToken);
    }
}

```

## UserProfile.cs

Режим транспортировки, имя и возраст пользователя сохраняются в экземпляре класса `UserProfile`.

```

public class UserProfile
{
    public string Transport { get; set; }

    public string Name { get; set; }

    public int Age { get; set; }

    public Attachment Picture { get; set; }
}

```

## Dialogs\UserProfileDialog.cs

На последнем шаге проверьте `stepContext.Result` Возвращаемое диалоговым окном значение, вызываемое в предыдущем каскадном шаге. Если возвращаемое значение равно true, то метод доступа профиля пользователя получает и обновляет профиль пользователя. Чтобы получить профиль пользователя, **вызовите**, `GetAsync` а затем задайте значения `userProfile.Transport` `userProfile.Name` свойств, `userProfile.Age` И `userProfile.Picture`. Наконец, перед вызовом необходимо обобщить сведения о пользователе `EndDialogAsync`, что завершает диалог. Завершенный диалог удаляется из стека диалогов, а его результат (если есть) возвращается в родительский диалог. Родительским считается диалог или метод, в котором был запущен только что завершившийся диалог.

```

private async Task<DialogTurnResult> SummaryStepAsync(WaterfallStepContext stepContext, CancellationToken cancellationToken)
{
    if ((bool)stepContext.Result)
    {
        // Get the current profile object from user state.
        var userProfile = await _userProfileAccessor.GetAsync(stepContext.Context, () => new UserProfile(), cancellationToken);

        userProfile.Transport = (string)stepContext.Values["transport"];
        userProfile.Name = (string)stepContext.Values["name"];
        userProfile.Age = (int)stepContext.Values["age"];
        userProfile.Picture = (Attachment)stepContext.Values["picture"];

        var msg = $"I have your mode of transport as {userProfile.Transport} and your name as {userProfile.Name}";

        if (userProfile.Age != -1)
        {
            msg += $" and your age as {userProfile.Age}";
        }

        msg += ".";

        await stepContext.Context.SendActivityAsync(MessageFactory.Text(msg), cancellationToken);

        if (userProfile.Picture != null)
        {
            try
            {
                await stepContext.Context.SendActivityAsync(MessageFactory.Attachment(userProfile.Picture,
                    "This is your profile picture."), cancellationToken);
            }
            catch
            {
                await stepContext.Context.SendActivityAsync(MessageFactory.Text("A profile picture was saved
but could not be displayed here."), cancellationToken);
            }
        }
        else
        {
            await stepContext.Context.SendActivityAsync(MessageFactory.Text("Thanks. Your profile will not be
kept."), cancellationToken);
        }
    }

    // WaterfallStep always finishes with the end of the Waterfall or with another dialog; here it is the
    end.
    return await stepContext.EndDialogAsync(cancellationToken: cancellationToken);
}

```

## Запуск диалога

- [C#](#)
- [JavaScript](#)
- [Python](#)

### Bots\DialogBot.cs

Обработчик `OnMessageActivityAsync` использует метод `RunAsync`, чтобы начать или продолжить диалог. `OnTurnAsync` использует объекты управления состоянием Bot для сохранения любых изменений состояния в хранилище. Метод `ActivityHandler.OnTurnAsync` вызывает разные методы обработки действий, например `OnMessageActivityAsync`. Таким образом состояние сохраняется после завершения

обработчика сообщений, но перед завершением работы.

```
public override async Task OnTurnAsync(ITurnContext turnContext, CancellationToken cancellationToken = default)
{
    await base.OnTurnAsync(turnContext, cancellationToken);

    // Save any state changes that might have occurred during the turn.
    await ConversationState.SaveChangesAsync(turnContext, false, cancellationToken);
    await UserState.SaveChangesAsync(turnContext, false, cancellationToken);
}

protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
    CancellationToken cancellationToken)
{
    Logger.LogInformation("Running dialog with Message Activity.");

    // Run the Dialog with the new message Activity.
    await Dialog.RunAsync(turnContext, ConversationState.CreateProperty<DialogState>(nameof(DialogState)),
        cancellationToken);
}
```

## Регистрация служб для бота

Этот робот использует следующие службы:

- Основные службы бота: поставщик учетных данных, адаптер и реализация бота.
  - Службы для управления состоянием: хранилище, состояние пользователя и состояние беседы.
  - Диалог, который будет использовать бот.
- [C#](#)
  - [JavaScript](#)
  - [Python](#)

`Startup.cs`.

Зарегистрируйте службы для Бот в `Startup`. Эти службы доступны в других частях кода через механизм внедрения зависимостей.

```
// This method gets called by the runtime. Use this method to add services to the container.
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers().AddNewtonsoftJson();

    // Create the Bot Framework Adapter with error handling enabled.
    services.AddSingleton<IBotFrameworkHttpAdapter, AdapterWithErrorHandler>();

    // Create the storage we'll be using for User and Conversation state. (Memory is great for testing purposes.)
    services.AddSingleton<IStorage, MemoryStorage>();

    // Create the User state. (Used in this bot's Dialog implementation.)
    services.AddSingleton<UserState>();

    // Create the Conversation state. (Used by the Dialog system itself.)
    services.AddSingleton<ConversationState>();

    // The Dialog that will be run by the bot.
    services.AddSingleton<UserProfileDialog>();

    // Create the bot as a transient. In this case the ASP Controller is expecting an IBot.
    services.AddTransient<IBot, DialogBot<UserProfileDialog>>();
}
```

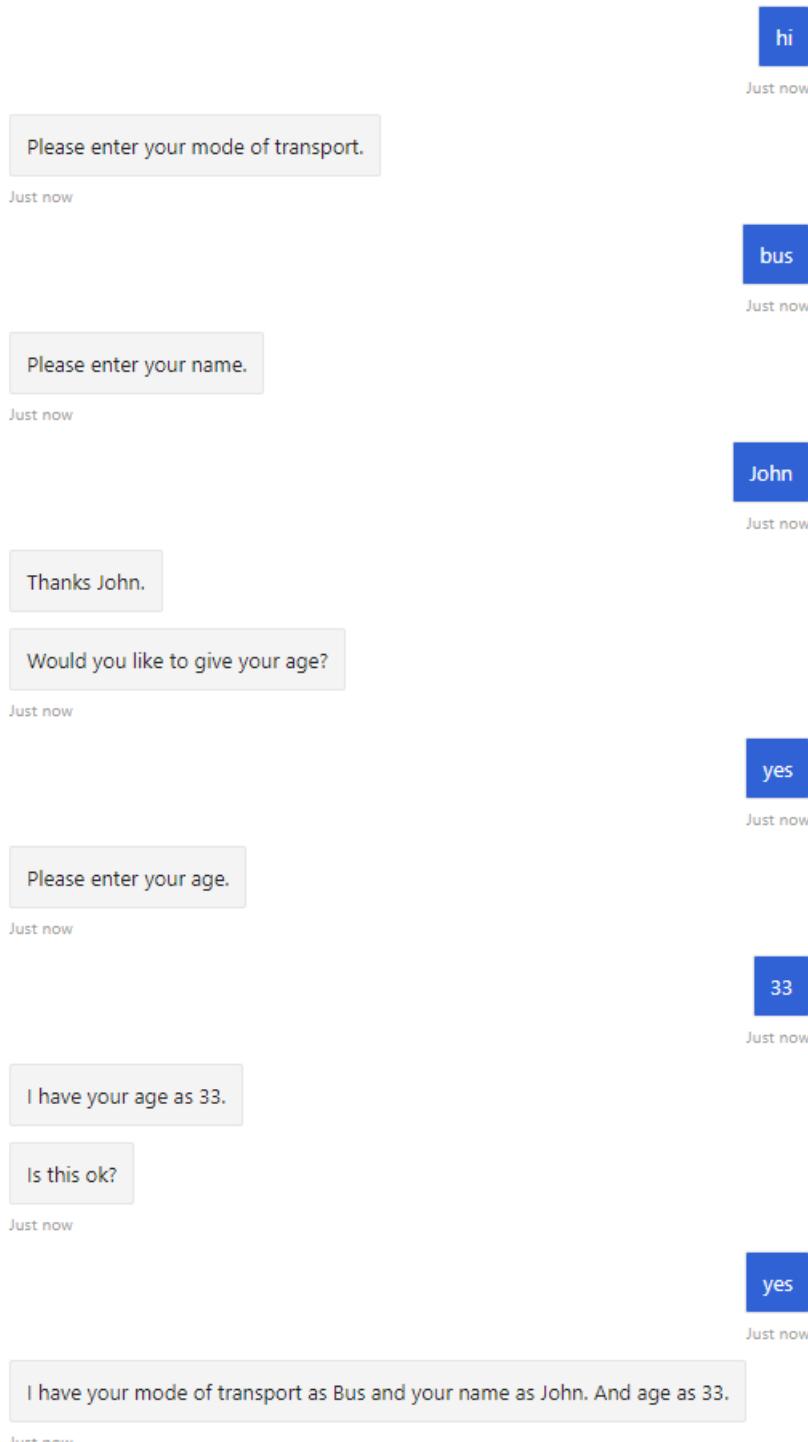
#### NOTE

Хранилище в памяти используется только для тестирования и не предназначено для рабочей среды. Для ботов в рабочей среде обязательно используйте постоянное хранилище любого типа.

## Тестирование бота

1. Установите [эмулятор Bot Framework](#), если вы еще этого не сделали.
2. Выполните этот пример на локальном компьютере.
3. Запустите эмулятор, подключитесь к боту и отправьте несколько сообщений, как показано ниже.

<http://localhost:3978/api/messages>



## Дополнительные сведения

### Сведения о диалоге и состоянии бота

В этом Bot определены два свойства доступа к свойству состояния:

- Один из них создается в состоянии беседы для свойства состояния диалога. Состояние диалогового окна отслеживает, где пользователь находится в диалоговых окнах набора диалоговых окон и обновляется контекстом диалогового окна, например при вызове диалогового окна *Begin* или *Continue Methods*.
- Второй создается в состоянии пользователя для свойства профиля пользователя. Робот использует это для отслеживания сведений о пользователе, и вам необходимо явно управлять этим состоянием в коде диалогового окна.

Методы доступа *get* и *set* для свойства состояния позволяют получить и сохранить значение этого

свойства в кэше объекта управления состоянием. Кэш заполняется автоматически при первом обращении к значению свойства состояния в течение шага, но сохранять его нужно явным образом. Чтобы сохранить изменения в обоих этих свойствах состояния, выполняется вызов метода *Save Changes* соответствующего объекта управления состоянием.

В этом примере обновляется состояние профиля пользователя из диалога. Эта практика может работать для простого робота, но она не будет работать, если вы хотите повторно использовать диалоговое окно в программы-роботы.

Есть несколько подходов, позволяющих отделить этапы диалога от состояний бота. Например, после сбора информации вы можете сделать следующее:

- Используйте метод *End Dialog* для предоставления собранных данных в качестве возвращаемого значения обратно в родительский контекст. Это может быть обработчик «поверх» программы-робота или предыдущее активное диалоговое окно в стеке диалоговых окон, в котором разрабатываются классы запросов.
- Создайте запрос к соответствующей службе. Это может быть хорошим вариантом, если бот выступает в роли интерфейса для более крупной службы.

### Определение метода для проверяющего элемента управления запроса

- [C#](#)
- [JavaScript](#)
- [Python](#)

#### UserProfileDialog.cs

Ниже приведен пример кода проверяющего элемента управления для `AgePromptValidatorAsync` определения метода. `promptContext.Recognized.Value` содержит анализируемое значение (в нашем примере это целое число из запроса числа). `promptContext.Recognized.Succeeded` указывает, удалось ли в запросе выполнить синтаксический анализ введенных пользователем данных. Проверяющий элемент управления должен возвращать значение `false`, если значение сочтено недопустимым, и в этом случае диалоговое окно запроса снова отображается пользователю. В противном случае должно вернуться значение `true`, чтобы принять введенные данные и завершить работу диалогового окна запроса. Обратите внимание, что это значение можно изменить в проверяющем элементе управления в соответствии с вашим сценарием.

```
private static Task<bool> AgePromptValidatorAsync(PromptValidatorContext<int> promptContext,
CancellationToken cancellationToken)
{
    // This condition is our validation rule. You can also change the value at this point.
    return Task.FromResult(promptContext.Recognized.Succeeded && promptContext.Recognized.Value > 0 &&
promptContext.Recognized.Value < 150);
}
```

## Дальнейшие действия

[Добавление возможности распознавания естественного языка в функционал бота](#)

# Управление сложностью диалогов

27.03.2021 • 10 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Компонентные диалоги позволяют создавать независимые диалоги для обработки определенных сценариев, разбивая большие наборы диалогов на более управляемые фрагменты. Каждый из этих компонентов имеет отдельный набор диалогов, что позволяет избежать конфликтов имен с внешними наборами диалогов. Компонентные диалоги можно использовать многократно для:

- добавления в другой `ComponentDialog` или `DialogSet` в боте;
- экспорта как части пакета;
- использования в других ботах.

## Предварительные требования

- Опыт работы с [ботами, библиотеками диалогов и управлением сообщениями](#).
- Копия примера запроса с несколькими шагами для [C#, JavaScript](#) или [Python](#).

## Сведения о примере

В примере диалога с несколькими запросами мы применим каскадный диалог, несколько запросов и компонентный диалог для реализации простого взаимодействия, в рамках которого пользователю предлагается несколько вопросов. Код диалога циклически перебирает следующие действия:

ШАГИ	ТИП ЗАПРОСА
Запрос к пользователю о режиме транспортировки	Запрос выбора
Запрос имени пользователя	Запрос текста
Запрос к пользователю, готов ли он указать свой возраст	Запрос подтверждения
Если получен положительный ответ, запрос возраста пользователя	Запрос числа с проверкой, при которой принимается возраст только в диапазоне от 0 до 150.
Запрос на подтверждение собранной информации	Повторный запрос подтверждения

И наконец, если получен положительный ответ, отображается вся собранная информация. В противном случае пользователь получает сообщение о том, что данные не будут сохранены.

## Реализация компонентного диалога

В примере диалога с несколькими запросами мы применим *каскадный диалог*, несколько запросов и *компонентный диалог* для создания простого взаимодействия, в рамках которого пользователю предлагается несколько вопросов.

Компонентный диалог включает один или несколько диалогов. Компонентный диалог содержит внутренний набор диалогов, в котором все добавляемые к диалоги и запросы имеют собственные идентификаторы, доступные только для компонентного диалога.

- [C#](#)
- [JavaScript](#)
- [Python](#)

Чтобы использовать диалоги, установите пакет NuGet **Microsoft.Bot.Builder.Dialogs**.

### Dialogs\UserProfileDialog.cs

Класс `UserProfileDialog` является производным от класса `ComponentDialog`.

```
public class UserProfileDialog : ComponentDialog
```

В конструкторе метод `AddDialog` добавляет диалоги и запросы в компонентный диалог. Первый элемент, добавленный с помощью этого метода, настраивается как начальный диалог. Но это можно изменить, явно задав свойство `InitialDialogId`. При запуске компонентного диалога будет запущен *начальный диалог*.

```
public UserProfileDialog(UserState userState)
    : base(nameof(UserProfileDialog))
{
    _userProfileAccessor = userState.CreateProperty<UserProfile>("UserProfile");

    // This array defines how the Waterfall will execute.
    var waterfallSteps = new WaterfallStep[]
    {
        TransportStepAsync,
        NameStepAsync,
        NameConfirmStepAsync,
        AgeStepAsync,
        PictureStepAsync,
        ConfirmStepAsync,
        SummaryStepAsync,
    };

    // Add named dialogs to the DialogSet. These names are saved in the dialog state.
    AddDialog(new WaterfallDialog(nameof(WaterfallDialog), waterfallSteps));
    AddDialog(new TextPrompt(nameof(TextPrompt)));
    AddDialog(new NumberPrompt<int>(nameof(NumberPrompt<int>), AgePromptValidatorAsync));
    AddDialog(new ChoicePrompt(nameof(ChoicePrompt)));
    AddDialog(new ConfirmPrompt(nameof(ConfirmPrompt)));
    AddDialog(new AttachmentPrompt(nameof(AttachmentPrompt), PicturePromptValidatorAsync));

    // The initial child Dialog to run.
    InitialDialogId = nameof(WaterfallDialog);
}
```

Это реализация первого шага каскадного диалога.

```
private static async Task<DialogTurnResult> NameStepAsync(WaterfallStepContext stepContext,
CancellationToken cancellationToken)
{
    stepContext.Values["transport"] = ((FoundChoice)stepContext.Result).Value;

    return await stepContext.PromptAsync(nameof(TextPrompt), new PromptOptions { Prompt =
MessageFactory.Text("Please enter your name.") }, cancellationToken);
}
```

См. подробнее о [реализации последовательного потока диалога](#).

Во время выполнения компонентный диалог поддерживает собственный стек диалогов. При запуске

компонентного диалога происходит следующее:

- Создается экземпляр, который добавляется во внешний стек диалогов.
- Создается внутренний стек диалогов, который добавляется в свое состояние.
- Запускается начальный диалог, который добавляется во внутренний стек диалогов.

Из родительского контекста активным диалогом будет считаться компонентный диалог. Внутри него активным диалогом будет считаться начальный диалог.

#### Реализация остальной части диалога и добавление кода в бот

Во внешнем наборе диалогов, в который вы добавили компонентный диалог, такому диалогу присвоен идентификатор, с которым он был создан. Во внешнем наборе компонентный диалог выглядят как один диалог, похожий на обычные запросы.

Чтобы использовать компонентный диалог, добавьте его экземпляр в набор диалогов бота, то есть во внешний набор диалогов.

- [C#](#)
- [JavaScript](#)
- [Python](#)

#### Bots\DialogBot.cs

В нашем примере это выполняется с помощью метода `RunAsync`, вызываемого из метода `OnMessageActivityAsync` бота.

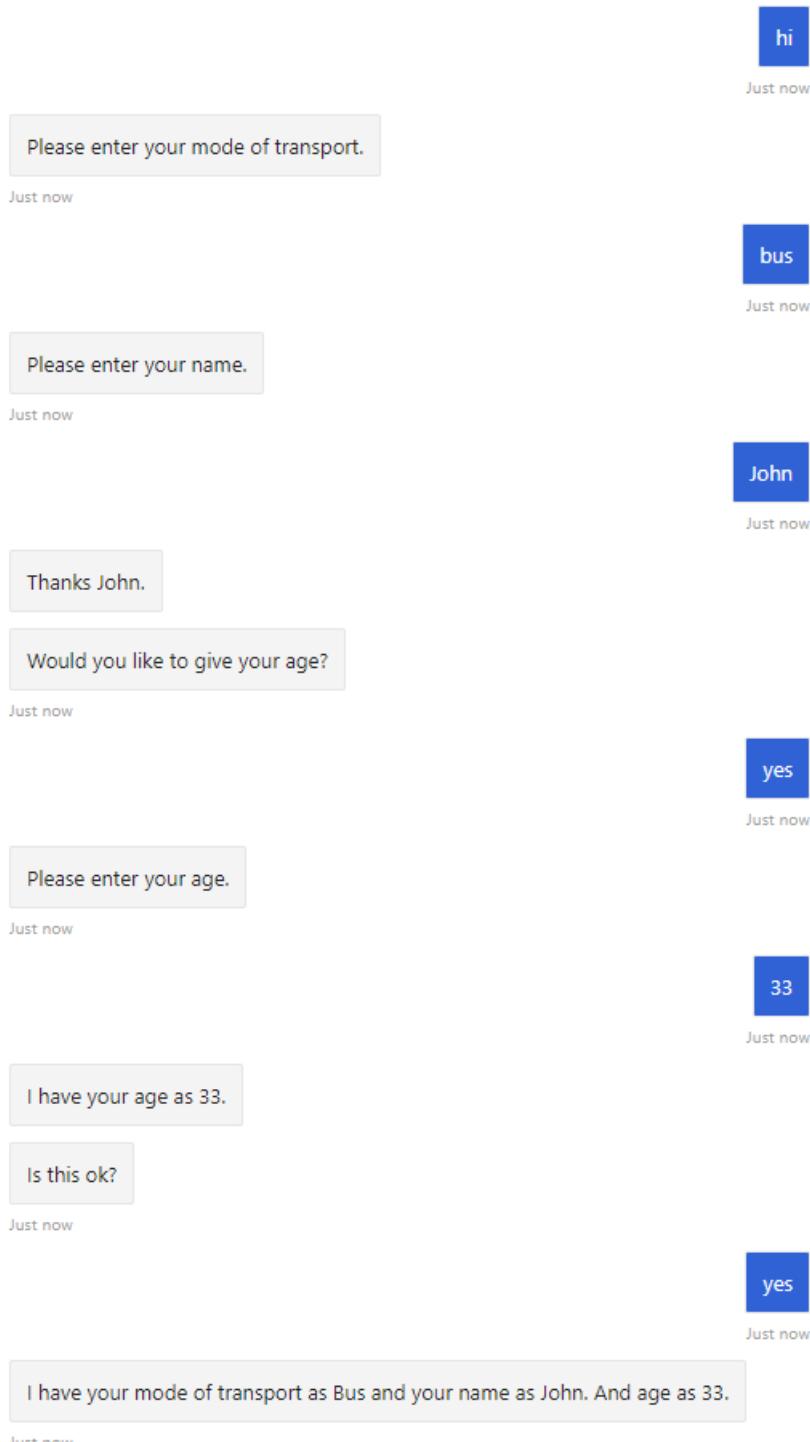
```
protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    Logger.LogInformation("Running dialog with Message Activity.");

    // Run the Dialog with the new message Activity.
    await Dialog.RunAsync(turnContext, ConversationState.CreateProperty<DialogState>(nameof(DialogState)),
cancellationToken);
}
```

## Тестирование бота

1. Установите [Bot Framework Emulator](#), если вы этого еще не сделали.
2. Выполните этот пример на локальном компьютере.
3. Запустите эмулятор, подключитесь к боту и отправьте несколько сообщений, как показано ниже.

<http://localhost:3978/api/messages>



## Дополнительные сведения

### Как работает отмена для компонентных диалогов

Если вы вызовете метод `отмены всех диалогов` из контекста компонентного диалога, этот метод отменит все диалоги во внутреннем стеке и завершит работу, передав управление следующему диалогу во внешнем стеке.

Если вы вызовете метод `отмены всех диалогов` из внешнего контекста, компонентный диалог будет отменен вместе со всеми остальными диалогами во внешнем контексте.

Не забывайте об этом, организуя управление вложенными компонентными диалогами в боте.

## Дальнейшие действия

Узнайте, как управлять сложными диалогами с ветвлениями и циклами.

[Обработка прерываний диалога пользователем](#)

# Создание сложного потока беседы с использованием ветвления и циклов

27.03.2021 • 18 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

С помощью библиотеки диалогов можно создавать сложные потоки беседы. В этой статье объясняется, как управлять сложными беседами с ответвлением и циклами, а также как передавать аргументы между разными частями диалога.

## Предварительные требования

- Понимание [основных принципов работы ботов, управления состоянием, библиотек диалогов и реализации последовательного процесса общения](#).
- Копия примера сложного диалога для [C#](#), [JavaScript](#) или [Python](#).

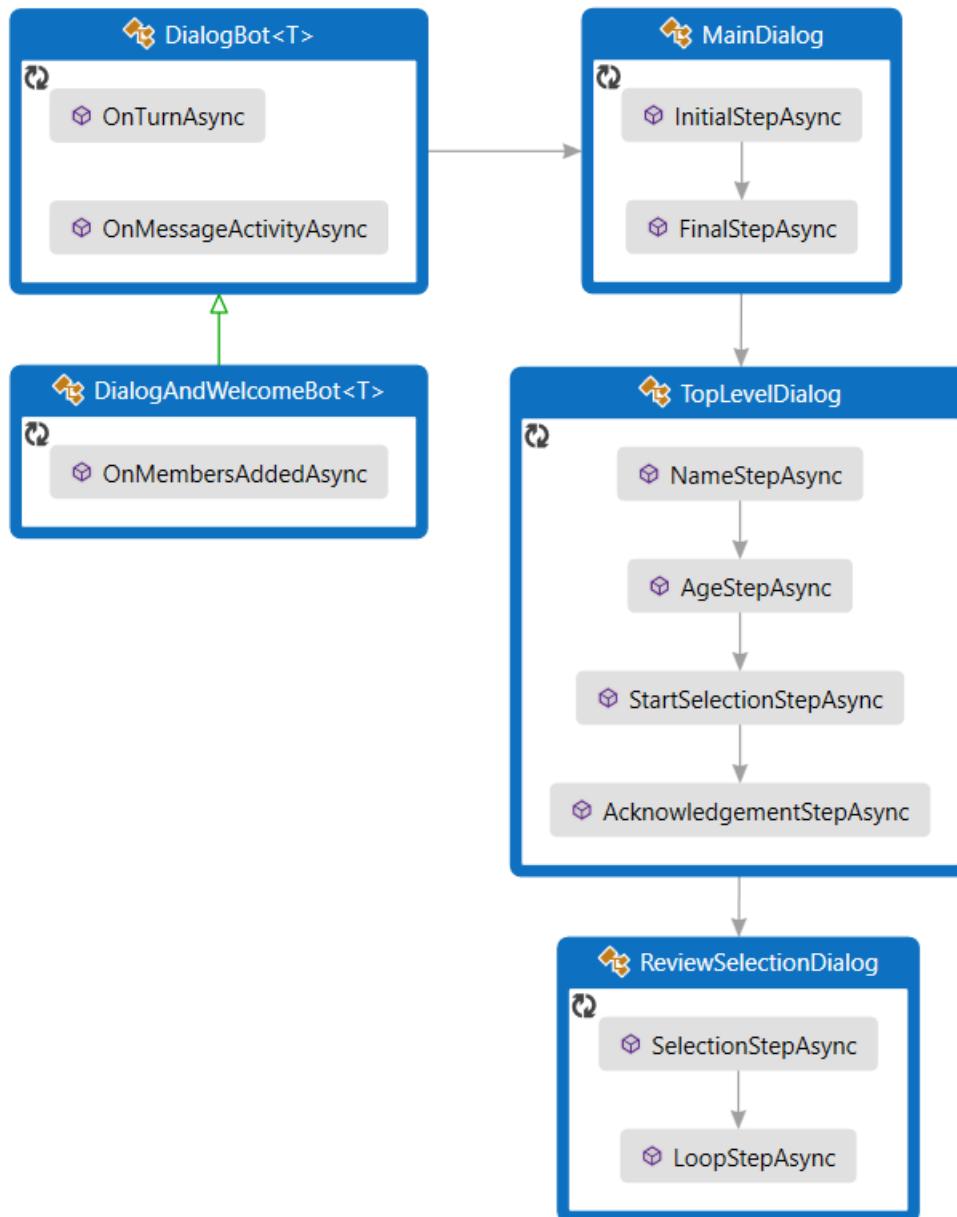
## Об этом примере

Этот пример содержит бот, который может выполнять регистрацию пользователей для оценки одной или двух компаний из заданного списка. Для управления потоком беседы в боте используются три диалога-компоненты. Каждый диалог-компонент содержит каскадный диалог и все запросы, необходимые для сбора данных, вводимых пользователем. Эти диалоги подробно описаны в следующих разделах. Бот использует состояние беседы для управления диалогами и состояние пользователя для сохранения сведений о нем и тех компаниях, которые пользователь хочет оценить.

Бот разработан на основе обработчика действий. Как и во многих примерах ботов, этот бот отправляет пользователю приветствие, использует диалоги для обработки сообщений от пользователя, а также сохраняет состояние пользователя и диалога до окончания реплики.

- [C#](#)
- [JavaScript](#)
- [Python](#)

Чтобы использовать диалоги, установите пакет NuGet `Microsoft.Bot.Builder.Dialogs`.



## Определение профиля пользователя

Профиль пользователя будет содержать такие собранные в диалогах сведения, как имя пользователя, возраст и выбранные для оценки компании.

- C#
- JavaScript
- Python

### UserProfile.cs

```

/// <summary>Contains information about a user.</summary>
public class UserProfile
{
    public string Name { get; set; }
    public int Age { get; set; }

    // The list of companies the user wants to review.
    public List<string> CompaniesToReview { get; set; } = new List<string>();
}

```

# Создание диалогов

Этот бот содержит три диалога:

- В основном диалоге запускается общий процесс общения и создается сводка собранных сведений.
- В диалоге верхнего уровня происходит сбор сведений пользователя. Здесь включена логика ветвления на основе возраста пользователя.
- В диалоге оценки и выбора пользователь может последовательно выбрать компании для оценки. Для этого используется логика цикла.

## Основной диалог

Основной диалог состоит из двух шагов:

1. Запуск диалога верхнего уровня.
2. Получение и формирование сводки данных в профиле пользователя, собранных в диалоге верхнего уровня, сохранение данных о состоянии пользователя и информирование о завершении основного диалога.

- [C#](#)
- [JavaScript](#)
- [Python](#)

### Dialogs\MainDialog.cs

```
private async Task<DialogTurnResult> InitialStepAsync(WaterfallStepContext stepContext, CancellationToken cancellationToken)
{
    return await stepContext.BeginDialogAsync(nameof(TopLevelDialog), null, cancellationToken);
}

private async Task<DialogTurnResult> FinalStepAsync(WaterfallStepContext stepContext, CancellationToken cancellationToken)
{
    var userInfo = (UserProfile)stepContext.Result;

    string status = "You are signed up to review "
        + (userInfo.CompaniesToReview.Count == 0 ? "no companies" : string.Join(" and ",
userInfo.CompaniesToReview))
        + ".";

    await stepContext.Context.SendActivityAsync(status);

    var accessor = _userState.CreateProperty<UserProfile>(nameof(UserProfile));
    await accessor.SetAsync(stepContext.Context, userInfo, cancellationToken);

    return await stepContext.EndDialogAsync(null, cancellationToken);
}
```

## Диалог верхнего уровня

Диалог верхнего уровня состоит из четырех шагов:

1. запрос имени пользователя;
2. запрос возраста пользователя;
3. Запуск диалога оценки и выбора или переход к следующему шагу в зависимости от возраста пользователя.
4. отправка пользователю благодарности за участие и возвращение собранных сведений.

На первом шаге в рамках сохранения состояния диалога создается пустой профиль пользователя. В

начале диалога создается пустой профиль, который заполняется в ходе ведения беседы. В конце диалога возвращаются собранные сведения.

На третьем шаге (начало выбора) в потоке беседы создается ветвь на основе возраста пользователя.

- [C#](#)
- [JavaScript](#)
- [Python](#)

[Dialogs\TopLevelDialog.cs](#)

```

private static async Task<DialogTurnResult> NameStepAsync(WaterfallStepContext stepContext,
CancellationToken cancellationToken)
{
    // Create an object in which to collect the user's information within the dialog.
    stepContext.Values[UserInfo] = new UserProfile();

    var promptOptions = new PromptOptions { Prompt = MessageFactory.Text("Please enter your name.") };

    // Ask the user to enter their name.
    return await stepContext.PromptAsync(nameof(TextPrompt), promptOptions, cancellationToken);
}

private async Task<DialogTurnResult> AgeStepAsync(WaterfallStepContext stepContext, CancellationToken cancellationToken)
{
    // Set the user's name to what they entered in response to the name prompt.
    var userProfile = (UserProfile)stepContext.Values[UserInfo];
    userProfile.Name = (string)stepContext.Result;

    var promptOptions = new PromptOptions { Prompt = MessageFactory.Text("Please enter your age.") };

    // Ask the user to enter their age.
    return await stepContext.PromptAsync(nameof(NumberPrompt<int>), promptOptions, cancellationToken);
}

private async Task<DialogTurnResult> StartSelectionStepAsync(WaterfallStepContext stepContext,
CancellationToken cancellationToken)
{
    // Set the user's age to what they entered in response to the age prompt.
    var userProfile = (UserProfile)stepContext.Values[UserInfo];
    userProfile.Age = (int)stepContext.Result;

    if (userProfile.Age < 25)
    {
        // If they are too young, skip the review selection dialog, and pass an empty list to the next step.
        await stepContext.Context.SendActivityAsync(
            MessageFactory.Text("You must be 25 or older to participate."),
            cancellationToken);
        return await stepContext.NextAsync(new List<string>(), cancellationToken);
    }
    else
    {
        // Otherwise, start the review selection dialog.
        return await stepContext.BeginDialogAsync(nameof(ReviewSelectionDialog), null, cancellationToken);
    }
}

private async Task<DialogTurnResult> AcknowledgementStepAsync(WaterfallStepContext stepContext,
CancellationToken cancellationToken)
{
    // Set the user's company selection to what they entered in the review-selection dialog.
    var userProfile = (UserProfile)stepContext.Values[UserInfo];
    userProfile.CompaniesToReview = stepContext.Result as List<string> ?? new List<string>();

    // Thank them for participating.
    await stepContext.Context.SendActivityAsync(
        MessageFactory.Text($"Thanks for participating,
{((UserProfile)stepContext.Values[UserInfo]).Name}."),
        cancellationToken);

    // Exit the dialog, returning the collected user information.
    return await stepContext.EndDialogAsync(stepContext.Values[UserInfo], cancellationToken);
}

```

## Диалог выбора элементов для обзора

Диалог оценки и выбора состоит из двух шагов:

1. Предложение пользователю выбрать компанию для оценки или ввести `done`, чтобы завершить процесс.

- Если в начале диалога были известны какие-либо сведения, их можно получить с помощью свойства *options* контекста каскадного шага. Диалог оценки и выбора поддерживает автоматический перезапуск. Благодаря этому пользователь может выбрать несколько компаний для оценки.
- Если пользователь уже выбрал компанию для оценки, ее название удаляется из доступных вариантов.
- Вариант `done` добавлен, чтобы пользователь мог завершить цикл на раннем этапе.

2. повтор того же диалога или выход, в зависимости от некоторых условий.

- Если пользователь выбрал компанию для оценки, ее нужно добавить в список.
- Если пользователь выбрал две компании или решил выйти, нужно завершить диалог и вывести список собранных сведений.
- В противном случае перезапустите диалог, инициализируя его с содержимым списка.

• [C#](#)

• [JavaScript](#)

• [Python](#)

`Dialogs\ReviewSelectionDialog.cs`

```

private async Task<DialogTurnResult> SelectionStepAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken)
{
    // Continue using the same selection list, if any, from the previous iteration of this dialog.
    var list = stepContext.Options as List<string> ?? new List<string>();
    stepContext.Values[CompaniesSelected] = list;

    // Create a prompt message.
    string message;
    if (list.Count is 0)
    {
        message = $"Please choose a company to review, or `'{DoneOption}` to finish.";
    }
    else
    {
        message = $"You have selected **{list[0]}**. You can review an additional company, " +
            $"or choose `'{DoneOption}` to finish.";
    }

    // Create the list of options to choose from.
    var options = _companyOptions.ToList();
    options.Add(DoneOption);
    if (list.Count > 0)
    {
        options.Remove(list[0]);
    }

    var promptOptions = new PromptOptions
    {
        Prompt = MessageFactory.Text(message),
        RetryPrompt = MessageFactory.Text("Please choose an option from the list."),
        Choices = ChoiceFactory.ToChoices(options),
    };

    // Prompt the user for a choice.
    return await stepContext.PromptAsync(nameof(ChoicePrompt), promptOptions, cancellationToken);
}

private async Task<DialogTurnResult> LoopStepAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken)
{
    // Retrieve their selection list, the choice they made, and whether they chose to finish.
    var list = stepContext.Values[CompaniesSelected] as List<string>;
    var choice = (FoundChoice)stepContext.Result;
    var done = choice.Value == DoneOption;

    if (!done)
    {
        // If they chose a company, add it to the list.
        list.Add(choice.Value);
    }

    if (done || list.Count >= 2)
    {
        // If they're done, exit and return their list.
        return await stepContext.EndDialogAsync(list, cancellationToken);
    }
    else
    {
        // Otherwise, repeat this dialog, passing in the list from this iteration.
        return await stepContext.ReplaceDialogAsync(nameof(ReviewSelectionDialog), list, cancellationToken);
    }
}

```

## Запуск диалогов

Класс *dialog bot* позволяет расширить возможности обработчика действия. Этот класс содержит логику выполнения диалогов. Класс *dialog and welcome bot* позволяет расширить возможности чат-бота, а также добавить приветствие пользователя, когда он присоединяется к беседе.

Обработчик реплик бота повторяет поток беседы, который определен в трех диалогах. При получении сообщения от пользователя происходит следующее:

1. Запускается основной диалог.
  - Если стек диалога пуст, начнется основной диалог.
  - Если это не так, диалог все еще выполняется, а значит, активный диалог будет продолжен.
2. Сохраняется состояние пользователя, беседы и диалога, чтобы не потерять новые сведения о состоянии.
  - [C#](#)
  - [JavaScript](#)
  - [Python](#)

### Bots\DialogBot.cs

```
public override async Task OnTurnAsync(ITurnContext turnContext, CancellationToken cancellationToken = default(CancellationToken))
{
    await base.OnTurnAsync(turnContext, cancellationToken);

    // Save any state changes that might have occurred during the turn.
    await ConversationState.SaveChangesAsync(turnContext, false, cancellationToken);
    await UserState.SaveChangesAsync(turnContext, false, cancellationToken);
}

protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    Logger.LogInformation("Running dialog with Message Activity.");

    // Run the Dialog with the new message Activity.
    await Dialog.RunAsync(turnContext, ConversationState.CreateProperty<DialogState>(nameof(DialogState)),
cancellationToken);
}
```

## Регистрация служб для бота

При необходимости создайте и зарегистрируйте службы:

- Основные службы бота: адаптер и реализация бота.
  - Службы для управления состоянием: хранилище, состояние пользователя и состояние беседы.
  - Корневой диалог, который будет использоваться ботом.
- [C#](#)
  - [JavaScript](#)
  - [Python](#)

### Startup.cs.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers().AddNewtonsoftJson();

    // Create the Bot Framework Adapter with error handling enabled.
    services.AddSingleton<IBotFrameworkHttpAdapter, AdapterWithErrorHandler>();

    // Create the storage we'll be using for User and Conversation state. (Memory is great for testing purposes.)
    services.AddSingleton<IStorage, MemoryStorage>();

    // Create the User state. (Used in this bot's Dialog implementation.)
    services.AddSingleton<UserState>();

    // Create the Conversation state. (Used by the Dialog system itself.)
    services.AddSingleton<ConversationState>();

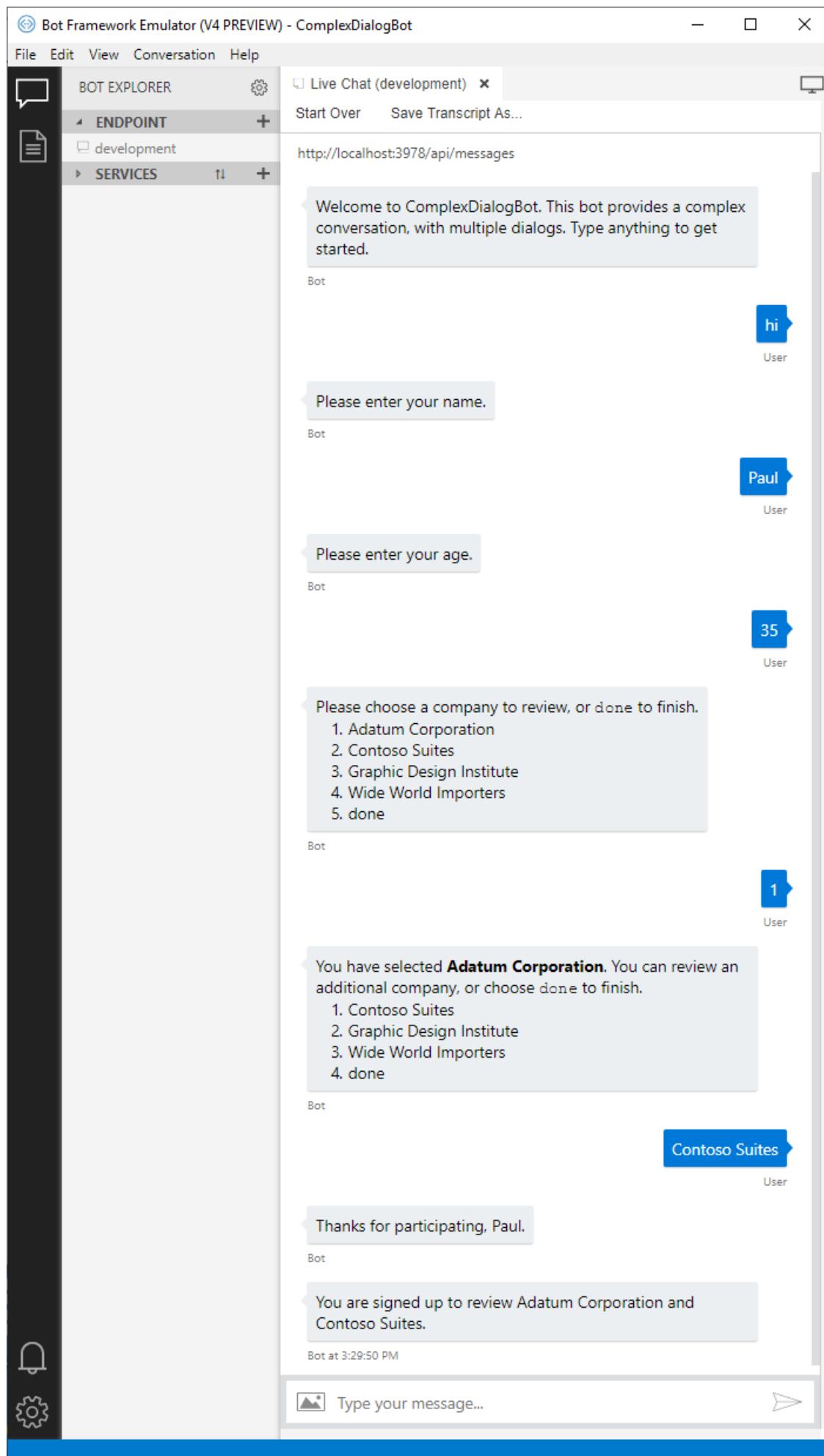
    services.AddSingleton<MainDialog>();
    // Create the bot as a transient. In this case the ASP Controller is expecting an IBot.
    services.AddTransient<IBot, DialogAndWelcomeBot<MainDialog>>();
}
```

#### NOTE

Хранилище в памяти используется только для тестирования и не предназначено для рабочей среды. Для ботов в рабочей среде обязательно используйте постоянное хранилище любого типа.

## Тестирование бота

1. Установите [Bot Framework Emulator](#), если вы этого еще не сделали.
2. Выполните этот пример на локальном компьютере.
3. Запустите эмулятор, подключитесь к боту и отправьте несколько сообщений, как показано ниже.



## Дополнительные ресурсы

Общие сведения о создании диалогов можно получить в статье [о реализации последовательного потока диалога](#), в котором на основе одного каскадного диалога и нескольких запросов создается простой пример взаимодействия, который задает пользователю набор вопросов.

Библиотека диалогов выполняет простую проверку запросов. Вы также можете добавить любую собственную проверку. Дополнительные сведения о сборе данных от пользователя с помощью запросов диалога см. в [этой статье](#).

Чтобы упростить код диалога и использовать его повторно в нескольких ботах, следует определить компоненты набора диалогов в отдельном классе. Подробнее об этом см. в статье [о повторном использовании диалогов](#).

## Дальнейшие действия

[Повторное использование диалогов](#)

# Обработка прерываний со стороны пользователя

27.03.2021 • 22 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Обработка прерываний является важным аспектом для создания надежного бота. Пользователи не всегда будут строго соблюдать определенный вами процесс общения. Они могут задать вопрос в середине процесса или отменить процесс вместо завершения. В этом разделе рассматривается несколько распространенных способов обработки прерываний со стороны пользователя.

## Предварительные требования

- Понимание [основных принципов работы ботов, управления состоянием, библиотек диалогов и повторного использования диалогов](#).
- Копия основного примера кода Bot в [C#, JavaScript](#) или [Python](#).

## Об этом примере

Пример в этой статье моделирует работу бота для бронирования авиабилетов, который использует диалоги для получения от пользователя информации о нужном рейсе. В любой момент общения с ботом пользователь может выдать команду *help* (помощь) или *cancel* (отмена), что должно вызывать прерывание. Здесь мы будем использовать два вида прерываний.

- **На уровне шага.** Обработка на уровне шага отменяется, но сам диалог сохраняется в стеке вместе с предоставленной информацией. Следующий шаг будет продолжен с того места, где остановился диалог.
- **На уровне диалога.** Полная отмена обработки, позволяющая боту начать работу сначала.

## Определение и реализация логики прерывания

Сначала нам нужно определить и реализовать прерывания по командам *help* и *cancel*.

- [C#](#)
- [JavaScript](#)
- [Python](#)

Чтобы использовать диалоги, установите пакет NuGet [Microsoft.Bot.Builder.Dialogs](#).

### Dialogs\CancelAndHelpDialog.cs

Реализуйте класс `CancelAndHelpDialog` для обработки прерываний со стороны пользователя. Отменяемые диалоги `BookingDialog` и `DateResolverDialog` являются производными от этого класса.

```
public class CancelAndHelpDialog : ComponentDialog
```

В классе `CancelAndHelpDialog` метод `OnContinueDialogAsync` вызывает метод `InterruptAsync`, чтобы проверить наличие прерываний со стороны пользователя. Если процесс прерывается, вызываются методы базового класса. В противном случае возвращается значение, полученное из `InterruptAsync`.

```

protected override async Task<DialogTurnResult> OnContinueDialogAsync(DialogContext innerDc,
CancellationToken cancellationToken = default)
{
    var result = await InterruptAsync(innerDc, cancellationToken);
    if (result != null)
    {
        return result;
    }

    return await base.OnContinueDialogAsync(innerDc, cancellationToken);
}

```

Если пользователь вводит слово help, метод `InterruptAsync` отправляет сообщение и вызывает `DialogTurnResult (DialogTurnStatus.Waiting)`, чтобы обозначить ожидание ответа от пользователя в диалоге верхнего уровня. В этом случае поток общения прерывается только на один шаг, а на следующем шаге продолжается с того места, где остановился диалог.

Если пользователь вводит cancel, то вызывается метод `CancelAllDialogsAsync` в контексте внутреннего диалога. Это действие очищает стек диалогов и приводит к выходу из диалога с состоянием отмены и без результирующего значения. С точки зрения `MainDialog` (см. далее) все будет выглядеть так, как будто диалог бронирования завершился и вернул значение NULL. Этот равнозначно ситуации, когда пользователь отказался подтвердить бронирование.

```

private async Task<DialogTurnResult> InterruptAsync(DialogContext innerDc, CancellationToken
cancellationToken)
{
    if (innerDc.Context.Activity.Type == ActivityTypes.Message)
    {
        var text = innerDc.Context.Activity.Text.ToLowerInvariant();

        switch (text)
        {
            case "help":
            case "?":
                var helpMessage = MessageFactory.Text(HelpMsgText, HelpMsgText, InputHints.ExpectingInput);
                await innerDc.Context.SendActivityAsync(helpMessage, cancellationToken);
                return new DialogTurnResult(DialogTurnStatus.Waiting);

            case "cancel":
            case "quit":
                var cancelMessage = MessageFactory.Text(CancelMsgText, CancelMsgText,
InputHints.IgnoringInput);
                await innerDc.Context.SendActivityAsync(cancelMessage, cancellationToken);
                return await innerDc.CancelAllDialogsAsync(cancellationToken);
        }
    }

    return null;
}

```

## Проверка прерываний на каждом шаге

После реализации класса обработки прерываний проверьте, что происходит, когда бот получает новое сообщение от пользователя.

- [C#](#)
- [JavaScript](#)
- [Python](#)

При поступлении действия с новым сообщением бот выполняет `MainDialog`. В `MainDialog` у пользователя спрашивается, какая помочь тому требуется. Затем запускается `BookingDialog` в методе `MainDialog.ActStepAsync` с помощью вызова `BeginDialogAsync`, как показано ниже.

```
private async Task<DialogTurnResult> ActStepAsync(WaterfallStepContext stepContext, CancellationToken cancellationToken)
{
    if (!_luisRecognizer.IsConfigured)
    {
        // LUIS is not configured, we just run the BookingDialog path with an empty BookingDetailsInstance.
        return await stepContext.BeginDialogAsync(nameof(BookingDialog), new BookingDetails(),
cancellationToken);
    }

    // Call LUIS and gather any potential booking details. (Note the TurnContext has the response to the
prompt.)
    var luisResult = await _luisRecognizer.RecognizeAsync<FlightBooking>(stepContext.Context,
cancellationToken);
    switch (luisResult.TopIntent().intent)
    {
        case FlightBooking.Intent.BookFlight:
            await ShowWarningForUnsupportedCities(stepContext.Context, luisResult, cancellationToken);

            // Initialize BookingDetails with any entities we may have found in the response.
            var bookingDetails = new BookingDetails()
            {
                // Get destination and origin from the composite entities arrays.
                Destination = luisResult.ToEntities.Airport,
                Origin = luisResult.FromEntities.Airport,
                TravelDate = luisResult.TravelDate,
            };

            // Run the BookingDialog giving it whatever details we have from the LUIS call, it will fill out
the remainder.
            return await stepContext.BeginDialogAsync(nameof(BookingDialog), bookingDetails,
cancellationToken);

        case FlightBooking.Intent.GetWeather:
            // We haven't implemented the GetWeatherDialog so we just display a TODO message.
            var getWeatherMessageText = "TODO: get weather flow here";
            var getWeatherMessage = MessageFactory.Text(getWeatherMessageText, getWeatherMessageText,
InputHints.IgnoringInput);
            await stepContext.Context.SendActivityAsync(getWeatherMessage, cancellationToken);
            break;

        default:
            // Catch all for unhandled intents
            var didntUnderstandMessageText = $"Sorry, I didn't get that. Please try asking in a different
way (intent was {luisResult.TopIntent().intent})";
            var didntUnderstandMessage = MessageFactory.Text(didntUnderstandMessageText,
didntUnderstandMessageText, InputHints.IgnoringInput);
            await stepContext.Context.SendActivityAsync(didntUnderstandMessage, cancellationToken);
            break;
    }

    return await stepContext.NextAsync(null, cancellationToken);
}
```

После этого в методе `FinalStepAsync` класса `MainDialog` завершается диалог бронирования — оно считается завершенным или отмененным.

```
private async Task<DialogTurnResult> FinalStepAsync(WaterfallStepContext stepContext, CancellationToken cancellationToken)
{
    // If the child dialog ("BookingDialog") was cancelled, the user failed to confirm or if the intent
    // wasn't BookFlight
    // the Result here will be null.
    if (stepContext.Result is BookingDetails result)
    {
        // Now we have all the booking details call the booking service.

        // If the call to the booking service was successful tell the user.

        var timeProperty = new TimexProperty(result.TravelDate);
        var travelDateMsg = timeProperty.ToNaturalLanguage(DateTime.Now);
        var messageText = $"I have you booked to {result.Destination} from {result.Origin} on
{travelDateMsg}";
        var message = MessageFactory.Text(messageText, messageText, InputHints.IgnoringInput);
        await stepContext.Context.SendActivityAsync(message, cancellationToken);
    }

    // Restart the main dialog with a different message the second time around
    var promptMessage = "What else can I do for you?";
    return await stepContext.ReplaceDialogAsync(InitialDialogId, promptMessage, cancellationToken);
}
```

Код в `BookingDialog` здесь не показан, так как он не имеет прямого отношения к обработке прерывания. Он используется, чтобы запросить у пользователя сведения о бронировании. Вы можете найти этот код в файле `Dialogs\BookingDialogs.cs`.

## Обработка непредвиденных ошибок

Обработчик ошибок адаптера обрабатывает все исключения, которые не были перехвачены в боте.

- [C#](#)
- [JavaScript](#)
- [Python](#)

### AdapterWithErrorHandler.cs

В нашем примере обработчик `OnTurnError` в адаптере получает все исключения, создаваемые в соответствии с логикой шага в боте. Если создано исключение, обработчик удаляет состояние текущей беседы, чтобы бот не застрял в цикле ошибки из-за неправильного состояния.

```

OnTurnError = async (turnContext, exception) =>
{
    // Log any leaked exception from the application.
    // NOTE: In production environment, you should consider logging this to
    // Azure Application Insights. Visit https://aka.ms/bottelemetry to see how
    // to add telemetry capture to your bot.
    logger.LogError(exception, $"[OnTurnError] unhandled error : {exception.Message}");

    // Send a message to the user
    var errorMessageText = "The bot encountered an error or bug.";
    var errorMessage = MessageFactory.Text(errorMessageText, errorMessageText, InputHints.IgnoringInput);
    await turnContext.SendActivityAsync(errorMessage);

    errorMessageText = "To continue to run this bot, please fix the bot source code.";
    errorMessage = MessageFactory.Text(errorMessageText, errorMessageText, InputHints.ExpectingInput);
    await turnContext.SendActivityAsync(errorMessage);

    if (conversationState != null)
    {
        try
        {
            // Delete the conversationState for the current conversation to prevent the
            // bot from getting stuck in a error-loop caused by being in a bad state.
            // ConversationState should be thought of as similar to "cookie-state" in a Web pages.
            await conversationState.DeleteAsync(turnContext);
        }
        catch (Exception e)
        {
            logger.LogError(e, $"Exception caught on attempting to Delete ConversationState : {e.Message}");
        }
    }

    // Send a trace activity, which will be displayed in the Bot Framework Emulator
    await turnContext.TraceActivityAsync("OnTurnError Trace", exception.Message,
    "https://www.botframework.com/schemas/error", "TurnError");
};


```

## Регистрация служб

- [C#](#)
- [JavaScript](#)
- [Python](#)

### Startup.cs.

Наконец, в `Startup.cs` создается временный бот, то есть на каждом шаге создается новый экземпляр бота.

```

// Create the bot as a transient. In this case the ASP Controller is expecting an IBot.
services.AddTransient<IBot, DialogAndWelcomeBot<MainDialog>>();

```

Для справки ниже приведены определения классов, которые используются в описанном выше вызове для создания бота.

```
public class MainDialog : ComponentDialog
```

```
public class DialogAndWelcomeBot<T> : DialogBot<T>
```

```
public class DialogBot<T> : ActivityHandler  
    where T : Dialog
```

## Тестирование бота

1. Установите [Bot Framework Emulator](#), если вы этого еще не сделали.
2. Выполните этот пример на локальном компьютере.
3. Запустите эмулятор, подключитесь к боту и отправьте несколько сообщений, как показано ниже.

## Дополнительные сведения

- Пример 24. Bot-Authentication-мсграф в [C#](#), [JavaScript](#)или [Python](#) демонстрирует обработку запроса на выход. Для обработки прерываний в нем используется шаблон, аналогичный показанному здесь.
- Вместо бездействия следует отправить ответ по умолчанию, чтобы пользователь не оставался в недоумении от молчания бота. Ответ по умолчанию должен сообщать, какие команды понимает бот, чтобы пользователь мог вернуться в нужное русло.
- В любой точке шага свойство контекста *responded* указывает, отправлял ли бот пользователю сообщение на этом шаге. Следите за тем, чтобы бот обязательно отправил пользователю хоть что-то до завершения шага, хотя бы просто подтверждение ввода.

# Завершение срока действия диалога

27.03.2021 • 12 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Иногда боту требуется перезапустить беседу с самого начала. Например, если пользователь не отвечает по истечении определенного периода времени. В этой статье описываются два способа истечения срока действия диалога:

- Отследить время последнего получения сообщения от пользователя и очистить состояние, если время больше, чем предварительно настроенная длина при получении следующего сообщения от пользователя. Дополнительные сведения см. в разделе [срок действия взаимодействия с пользователем](#).
- Используйте компонент уровня хранилища, например Cosmos DB срок жизни (TTL), чтобы автоматически очищать состояние после предварительно настроенного периода времени. Дополнительные сведения см. в разделе [срок действия хранилища](#).

## Предварительные требования

- Если у вас еще нет подписки Azure, создайте [бесплатную](#) учетную запись Azure, прежде чем начинать работу.
- Понимание [основных принципов работы ботов, управления состоянием и библиотеки диалогов](#).
- Копия примера [запроса с несколькими шагами](#) для [C#](#), [JavaScript](#) и [Python](#).

## Об этом примере

Пример кода в этой статье начинается со структуры многофункционального робота и расширяет ее функциональность, добавляя дополнительный код (приведенный в следующих разделах). В этом расширенном коде показано, как очистить состояние диалога после прохождения определенного периода времени.

## Срок действия взаимодействия с пользователем

Этот тип диалога с истекающим сроком действия достигается путем добавления свойства *времени последнего доступа* в состояние диалога Bot. Это значение свойства затем сравнивается с текущим временем в *обработчике действия* перед обработкой действий.

### NOTE

В этом примере для простоты тестирования этого шаблона используется время ожидания в 30 секунд.

- [C#](#)
- [JavaScript](#)
- [Python](#)

### appsettings.json

Сначала добавьте `ExpireAfterSeconds` параметр в appsettings.json:

```
{  
    "MicrosoftAppId": "",  
    "MicrosoftAppPassword": "",  
    "ExpireAfterSeconds": 30  
}
```

## Bots\DialogBot.cs

Затем добавьте `ExpireAfterSeconds` поля, `LastAccessedTimeProperty` и `DialogStateProperty` в класс Bot и инициализируйте их в конструкторе Bot. Также добавьте `IConfiguration` в конструктор параметр, который будет использоваться для получения `ExpireAfterSeconds` значения.

Обратите внимание, что вместо создания встроенного метода доступа к свойству состояния диалогового окна в `OnMessageActivityAsync` методе создается и записывается во время инициализации. Роботу требуется метод доступа к свойству состояния не только для запуска диалогового окна, но также для очистки состояния диалогового окна.

```
protected readonly int ExpireAfterSeconds;  
protected readonly IStatePropertyAccessor<DateTime> LastAccessedTimeProperty;  
protected readonly IStatePropertyAccessor<DialogState> DialogStateProperty;  
  
// Existing fields omitted...  
  
public DialogBot(IConfiguration configuration, ConversationState conversationState, UserState userState, T dialog, ILogger<DialogBot<T>> logger)  
{  
    ConversationState = conversationState;  
    UserState = userState;  
    Dialog = dialog;  
    Logger = logger;  
  
    TimeoutSeconds = configuration.GetValue<int>("ExpireAfterSeconds");  
    DialogStateProperty = ConversationState.CreateProperty<DialogState>(nameof(DialogState));  
    LastAccessedTimeProperty = ConversationState.CreateProperty<DateTime>(nameof(LastAccessedTimeProperty));  
}
```

Наконец, добавьте код в метод Bot, `OnTurnAsync` чтобы очистить состояние диалога, если диалог слишком стар.

```

public override async Task OnTurnAsync(ITurnContext turnContext, CancellationToken cancellationToken = default)
{
    // Retrieve the property value, and compare it to the current time.
    var lastAccess = await LastAccessedTimeProperty.GetAsync(turnContext, () => DateTime.UtcNow,
    cancellationToken).ConfigureAwait(false);
    if ((DateTime.UtcNow - lastAccess) >= TimeSpan.FromSeconds(ExpireAfterSeconds))
    {
        // Notify the user that the conversation is being restarted.
        await turnContext.SendActivityAsync("Welcome back! Let's start over from the
beginning.").ConfigureAwait(false);

        // Clear state.
        await ConversationState.ClearStateAsync(turnContext, cancellationToken).ConfigureAwait(false);
    }

    await base.OnTurnAsync(turnContext, cancellationToken).ConfigureAwait(false);

    // Set LastAccessedTime to the current time.
    await LastAccessedTimeProperty.SetAsync(turnContext, DateTime.UtcNow,
    cancellationToken).ConfigureAwait(false);

    // Save any state changes that might have occurred during the turn.
    await ConversationState.SaveChangesAsync(turnContext, false, cancellationToken).ConfigureAwait(false);
    await UserState.SaveChangesAsync(turnContext, false, cancellationToken).ConfigureAwait(false);
}

```

## Срок действия хранилища

Cosmos DB предоставляет функцию срока жизни (TTL), которая позволяет удалять элементы автоматически из контейнера по истечении определенного периода времени. Это можно настроить в портал Azure или во время создания контейнера (с помощью пакетов SDK для конкретного языка Cosmos DB).

Пакет SDK для Bot Framework не предоставляет параметр конфигурации TTL. Однако Инициализация контейнера может быть переопределена, а пакет SDK для Cosmos DB можно использовать для настройки TTL до инициализации хранилища ленты.

- [C#](#)
- [JavaScript](#)
- [Python](#)

Начните с свежей копии образца с **несколькими пересказками** и добавьте

`Microsoft.Bot.Builder.Azure` пакет NuGet в проект.

### appsettings.json

Обновите `appsettings.js`, чтобы включить параметры хранения Cosmos DB:

```
{
    "MicrosoftAppId": "",
    "MicrosoftAppPassword": "",

    "CosmosDbTimeToLive": 30,
    "CosmosDbEndpoint": "<endpoint-for-your-cosmosdb-instance>",
    "CosmosDbAuthKey": "<your-cosmosdb-auth-key>",
    "CosmosDbDatabaseId": "<your-database-id>",
    "CosmosDbUserStateContainerId": "<no-ttl-container-id>",
    "CosmosDbConversationStateContainerId": "<ttl-container-id>"
}
```

Обратите внимание на два Контейнеридс, один для `UserState` и один для `ConversationState`. Это связано с тем, что настройка времени жизни по умолчанию в `ConversationState` контейнере выполняется, но не на `UserState`.

### CosmosDbStorageInitializerHostedService.cs

Затем создайте `CosmosDbStorageInitializerHostedService` класс, который создаст контейнер с настроенным временем жизни.

```
// Add required using statements...

public class CosmosDbStorageInitializerHostedService : IHostedService
{
    readonly CosmosDbPartitionedStorageOptions _storageOptions;
    readonly int _cosmosDbTimeToLive;

    public CosmosDbStorageInitializerHostedService(IConfiguration config)
    {
        _storageOptions = new CosmosDbPartitionedStorageOptions()
        {
            CosmosDbEndpoint = config["CosmosDbEndpoint"],
            AuthKey = config["CosmosDbAuthKey"],
            DatabaseId = config["CosmosDbDatabaseId"],
            ContainerId = config["CosmosDbConversationStateContainerId"]
        };

        _cosmosDbTimeToLive = config.GetValue<int>("CosmosDbTimeToLive");
    }

    public async Task StartAsync(CancellationToken cancellationToken)
    {
        using (var client = new CosmosClient(
            _storageOptions.CosmosDbEndpoint,
            _storageOptions.AuthKey,
            _storageOptions.CosmosClientOptions ?? new CosmosClientOptions()))
        {
            // Create the container with the provided TTL
            var containerResponse = await client
                .GetDatabase(_storageOptions.DatabaseId)
                .DefineContainer(_storageOptions.ContainerId, "/id")
                .WithDefaultTimeToLive(_cosmosDbTimeToLive)
                .WithIndexingPolicy().WithAutomaticIndexing(false).Attach()
                .CreateIfNotExistsAsync(_storageOptions.ContainerThroughput)
                .ConfigureAwait(false);
        }
    }

    public Task StopAsync(CancellationToken cancellationToken) => Task.CompletedTask;
}
```

### Startup.cs

Наконец, обновите, `Startup.cs` чтобы использовать инициализатор хранилища, и Cosmos DB для State:

```

// Existing code omitted...

// commented out MemoryStorage, since we are using CosmosDbPartitionedStorage instead
// services.AddSingleton<IStorage, MemoryStorage>();

// Add the Initializer as a HostedService (so it is called during the app service startup)
services.AddHostedService<CosmosDbStorageInitializerHostedService>();

// Create the storage options for User state
var userStorageOptions = new CosmosDbPartitionedStorageOptions()
{
    CosmosDbEndpoint = Configuration["CosmosDbEndpoint"],
    AuthKey = Configuration["CosmosDbAuthKey"],
    DatabaseId = Configuration["CosmosDbDatabaseId"],
    ContainerId = Configuration["CosmosDbUserStateContainerId"]
};

// Create the User state. (Used in this bot's Dialog implementation.)
services.AddSingleton(new UserState(new CosmosDbPartitionedStorage(userStorageOptions)));

// Create the storage options for Conversation state
var conversationStorageOptions = new CosmosDbPartitionedStorageOptions()
{
    CosmosDbEndpoint = Configuration["CosmosDbEndpoint"],
    AuthKey = Configuration["CosmosDbAuthKey"],
    DatabaseId = Configuration["CosmosDbDatabaseId"],
    ContainerId = Configuration["CosmosDbConversationStateContainerId"]
};

// Create the Conversation state. (Used by the Dialog system itself.)
services.AddSingleton(new ConversationState(new CosmosDbPartitionedStorage(conversationStorageOptions)));

// Existing code omitted...

```

Cosmos DB теперь будет автоматически удалять записи о состоянии беседы через 30 секунд бездействия.

Дополнительные сведения см. [в разделе Настройка срока жизни в Azure Cosmos DB](#)

## Тестирование бота

1. Установите [Bot Framework Emulator](#), если вы этого еще не сделали.
2. Выполните этот пример на локальном компьютере.
3. Запустите эмулятор, подключитесь к роботу и отправьте в него сообщение.
4. После одного из приглашений подождите 30 секунд, прежде чем отвечать на запросы.

# Добавление возможности распознавания естественного языка в функционал бота

27.03.2021 • 19 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Возможность понимать, что пользователь хочет сказать и какой вкладывает контекст, может быть сложной задачей, но также может способствовать более естественной беседе с ботом. *API распознавания речи* — это облачная служба API, которая позволяет сделать так, чтобы бот мог распознавать намерения пользовательских сообщений, использовать более естественный язык пользователя и лучше направлять поток общения.

В этом разделе рассматривается добавление LUIS в приложение для бронирования авиабилетов, чтобы распознавать намерения и сущности в введенных пользователем данных.

## Предварительные требования

- Учетная запись [LUIS](#).
- Копия примера [Core Bot](#) на языке [C#](#), [JavaScript](#) или [Python](#).
- Понимание [основных принципов работы ботов, обработки естественного языка и управления ресурсами бота](#).

## Об этом примере

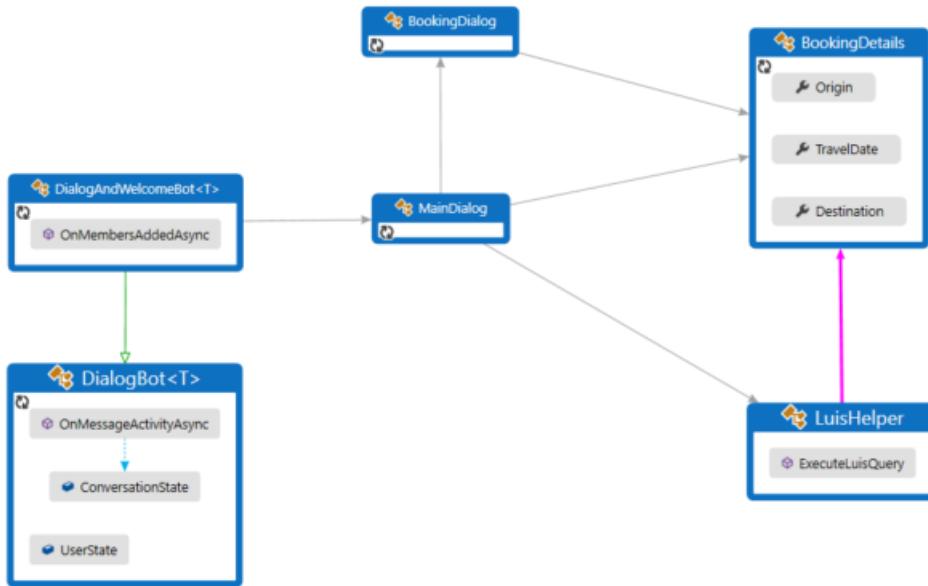
Этот пример основного бота реализует логику приложения для бронирования авиабилетов. С помощью службы LUIS он распознает пользовательский ввод и возвращает наиболее вероятное из обнаруженных LUIS намерений.

Языковая модель содержит три намерения: `Book Flight`, `Cancel` И `None`. Служба LUIS будет использовать эти намерения для распознавания желаний пользователя в полученном от него сообщении. Языковая модель также определяет сущности, которые LUIS может извлекать из входных данных пользователя, таких как аэропорты вылета или назначения.

- [C#](#)
- [JavaScript](#)
- [Python](#)

После каждой обработки введенных пользователем данных `DialogBot` сохраняет текущее состояние `UserState` И `ConversationState`. После сбора всех необходимых сведений в этом примере кода создается демонстрационное резервирование авиабилетов. В этой статье мы будем рассматривать те элементы примера, которые имеют отношение к LUIS. Но в целом поток логических действий в примере выглядит примерно так.

- Когда подключается новый пользователь, вызывается `OnMembersAddedAsync` И отображается приветственная карточка.
- `OnMessageActivityAsync` вызывается для каждого полученного блока данных, введенных пользователем.



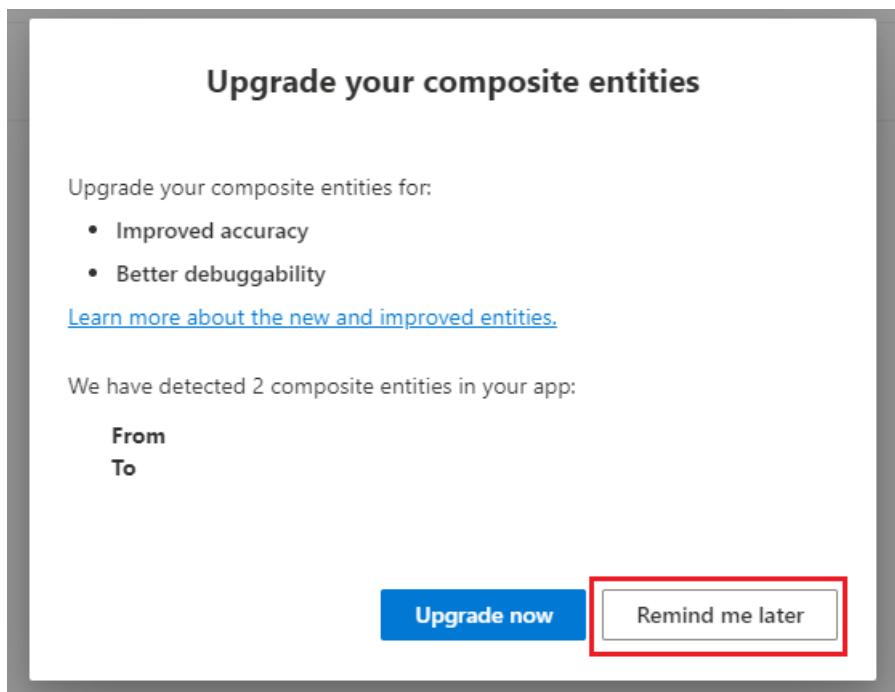
Модуль `OnMessageActivityAsync` запускает соответствующий диалог с помощью метода расширения диалога `Run`. Затем главный диалог вызывает вспомогательный метод LUIS для поиска самого вероятного намерения пользователя. Если таким намерением является BookFlight (бронирование авиабилетов), вспомогательный метод сохраняет полученные от LUIS данные пользователя. Затем главный диалог запускает `BookingDialog` для сбора требуемых дополнительных сведений от пользователя, например:

- `Origin` — город вылета;
- `TravelDate` — дата для бронирования авиабилетов;
- `Destination` — город прилета;

В этой статье описывается добавление LUIS в бот. Дополнительные сведения об использовании диалогов или состояния можно получить, ознакомившись со [сбором данных, которые вводит пользователь, с помощью запроса в диалоге](#) или [сохранением данных пользователя и диалога](#).

## Создание приложения LUIS на портале LUIS

1. Войдите на портал [LUIS](#).
  - Создайте учетную запись, если у вас ее еще нет.
  - Если у вас еще нет ресурса разработки, создайте его.
  - Дополнительные сведения см. в документации LUIS о том, как [войти на портал LUIS](#).
2. На странице `My Apps` (Мои приложения) выберите `+ New app for conversation` (Создать приложение для беседы), а затем выберите `Import as JSON` (Импортировать как JSON).
3. В диалоговом окне `Import new app` (Импорт нового приложения) сделайте следующее.
  - а. Выберите файл `FlightBooking.json` в папке `CognitiveModels` примера.
  - б. Введите `FlightBooking` в качестве необязательного имени приложения и нажмите кнопку `Done` (Готово).
4. Возможно, вам будет предложено обновить составные сущности. Можно проигнорировать это и нажать кнопку **напомнить позже**.



5. Обучите и опубликуйте свое приложение. Дополнительные сведения см. в документации LUIS по [обучению](#) и [публикации](#) приложения в рабочей среде.

#### Для чего нужны сущности

Сущности LUIS позволяют боту лучше понимать некоторые факты или события, которые отличаются от стандартных намерений. Это позволяет получить от пользователя дополнительную информацию, которая позволит боту точнее реагировать на действия пользователя или пропустить некоторые вопросы, в которых от пользователя запрашивается уже полученная информация. В файле FlightBooking.json определены не только три намерения LUIS (Book Flight, Cancel и None), но и набор дополнительных сущностей, таких как From.Airport и To.Airport. Эти сущности позволяют LUIS обнаруживать и возвращать дополнительные сведения из данных, которые пользователь предоставляет при запросе нового бронирования.

## Получение значений для подключения к приложению LUIS

После публикации приложения LUIS ваш бот сможет обратиться к нему. Для доступа к приложению LUIS из кода бота потребуется записать несколько значений. Нужные сведения можно получить с помощью портала LUIS.

#### Получение сведений о приложении на портале LUIS.ai

Файл параметров (`appsettings.json`, `.env` или `config.py`) используется, чтобы собрать в одном расположении ссылки на все службы. Полученные данные будут добавлены в этот файл в следующем разделе.

1. Выберите опубликованное приложение LUIS на сайте [luis.ai](https://luis.ai).
2. Открыв опубликованное приложение LUIS, выберите в нем вкладку **MANAGE** (Управление).
3. Перейдите на вкладку **Параметры** в левой части экрана и запишите значение, показанное в поле **идентификатор приложения** как `<YOUR_APP_ID>`.

FlightBooking (V0.1) ▾

DASHBOARD BUILD MANAGE Train Publish Test

Settings

Publish Settings

Azure Resources

Versions

### Application Settings

App name

FlightBooking

App description (optional)

Luis Model for CoreBot

App ID

Culture

en-us

Make endpoints public ?  Off

- Перейдите на вкладку **Azure Resources** (Ресурсы Azure) в левой части страницы и выберите группу **Authoring Resource** (Ресурс разработки). Запишите значение *Location* (Расположение) как <YOUR\_REGION>, а значение *Primary Key* (Первичный ключ) — как <YOUR\_AUTHORING\_KEY>.

Settings

Publish Settings

Azure Resources

Versions

### Azure Resources

LUIS uses two types of resources: authoring and prediction. The authoring key is needed for authoring, publishing, managing collaborators, and versioning. An authoring resource is created for you when you create your azure cognitive service account. When you are ready to publish your LUIS app, you need to create the prediction resource key and use the prediction key with endpoint queries. [Learn more](#).

Prediction Resources Authoring Resource

Resource group:

Location: westus

Primary Key:

Secondary Key:

Endpoint URL:  https://████████cognitiveservices.azure.com/

Pricing Tier: F0 (Free)

## Обновление файла параметров

- [C#](#)
- [JavaScript](#)
- [Python](#)

Добавьте в файл `appsettings.json` необходимые сведения для доступа к приложению LUIS, включая идентификатор приложения, ключ разработки и регион. Все эти данные вы ранее сохранили из опубликованного приложения LUIS. Обратите внимание, что имя узла API должно иметь формат `<your region>.api.cognitive.microsoft.com`.

## appsetting.json

```
{
  "MicrosoftAppId": "",
  "MicrosoftAppPassword": "",
  "LuisAppId": "",
  "LuisAPIKey": "",
  "LuisAPIHostName": ""
}
```

# Настройка бота для работы с приложением LUIS

- C#
- JavaScript
- Python

Убедитесь, что для вашего проекта установлен пакет NuGet Microsoft.Bot.Builder.AI.Luis.

Для подключения к службе LUIS бот извлекает сведения, которые вы ранее добавили в файл appsetting.json. Класс `FlightBookingRecognizer` содержит код с параметрами из файла appsetting.json и отправляет запрос к службе LUIS, вызывая метод `RecognizeAsync`.

## FlightBookingRecognizer.cs

```
public class FlightBookingRecognizer : IRecognizer
{
    private readonly LuisRecognizer _recognizer;

    public FlightBookingRecognizer(IConfiguration configuration)
    {
        var luisIsConfigured = !string.IsNullOrEmpty(configuration["LuisAppId"]) &&
        !string.IsNullOrEmpty(configuration["LuisAPIKey"]) &&
        !string.IsNullOrEmpty(configuration["LuisAPIHostName"]);
        if (luisIsConfigured)
        {
            var luisApplication = new LuisApplication(
                configuration["LuisAppId"],
                configuration["LuisAPIKey"],
                "https://" + configuration["LuisAPIHostName"]);
            // Set the recognizer options depending on which endpoint version you want to use.
            // More details can be found in https://docs.microsoft.com/en-gb/azure/cognitive-
            services/luis/luis-migration-api-v3
            var recognizerOptions = new LuisRecognizerOptionsV3(luisApplication)
            {
                PredictionOptions = new Bot.Builder.AI.LuisV3.LuisPredictionOptions
                {
                    IncludeInstanceData = true,
                }
            };
            _recognizer = new LuisRecognizer(recognizerOptions);
        }
    }

    // Returns true if luis is configured in the appsettings.json and initialized.
    public virtual bool IsConfigured => _recognizer != null;

    public virtual async Task<RecognizerResult> RecognizeAsync(ITurnContext turnContext, CancellationToken cancellationToken)
        => await _recognizer.RecognizeAsync(turnContext, cancellationToken);

    public virtual async Task<T> RecognizeAsync<T>(ITurnContext turnContext, CancellationToken cancellationToken)
        where T : IRecognizerConvert, new()
        => await _recognizer.RecognizeAsync<T>(turnContext, cancellationToken);
}
```

`FlightBookingEx.cs` содержит логику для извлечения *From*, *To* и *TravelDate*. Это расширение разделяемого класса `FlightBooking.cs`, используемого для хранения результатов LUIS при вызове `FlightBookingRecognizer.RecognizeAsync<FlightBooking>` из `MainDialog.cs`.

CognitiveModels\FlightBookingEx.cs

```

// Extends the partial FlightBooking class with methods and properties that simplify accessing entities in
// the luis results
public partial class FlightBooking
{
    public (string From, string Airport) FromEntities
    {
        get
        {
            var fromValue = Entities?._instance?.From?.FirstOrDefault()?.Text;
            var fromAirportValue =
                Entities?.From?.FirstOrDefault()?.Airport?.FirstOrDefault()?.FirstOrDefault();
            return (fromValue, fromAirportValue);
        }
    }

    public (string To, string Airport) ToEntities
    {
        get
        {
            var toValue = Entities?._instance?.To?.FirstOrDefault()?.Text;
            var toAirportValue =
                Entities?.To?.FirstOrDefault()?.Airport?.FirstOrDefault()?.FirstOrDefault();
            return (toValue, toAirportValue);
        }
    }

    // This value will be a TIMEX. And we are only interested in a Date so grab the first result and drop
    // the Time part.
    // TIMEX is a format that represents DateTime expressions that include some ambiguity. e.g. missing a
    // Year.
    public string TravelDate
        => Entities.datetime?.FirstOrDefault()?.Expressions.FirstOrDefault()?.Split('T')[0];
}

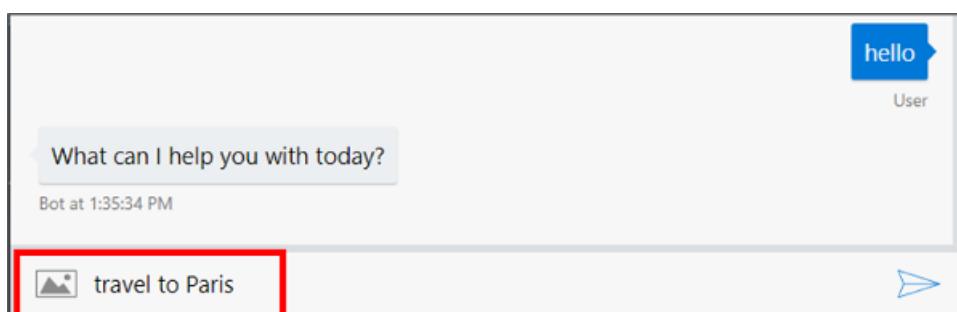
```

Теперь служба LUIS для бота полностью настроена и подключена.

## Тестирование бота

Скачайте и установите последнюю версию [Bot Framework Emulator](#).

- Выполните этот пример на локальном компьютере. Дополнительные инструкции, включая примеры для [C#](#), [JS](#) или [Python](#), см. в файле README.
- В эмуляторе введите сообщение, например "Путешествие в Париж" или "переход от Париж к Берлин". Используйте все речевые фрагменты, включенные в файл FlightBooking.json, для обучения намерения Book flight (Бронирование авиабилетов).



Если наиболее вероятное намерение LUIS соответствует элементу Book flight (Бронирование авиабилетов), бот будет задавать дополнительные вопросы, пока не получит достаточно сохраненных сведений для создания бронирования. После этого он возвращает всю собранную информацию о бронировании пользователю.

I have you booked to Paris from Berlin on today

Bot at 1:40:39 PM

Type your message... 

На этом этапе логика кода обнуляет состояние бота, и вы можете создать новое резервирование.

## Дополнительные сведения

Дополнительные сведения о LUIS см. в документации по LUIS:

- [Что такое служба "Распознавание речи" \(LUIS\)?](#)
- [Создание приложения LUIS на портале LUIS](#)
- [Проектирование с помощью моделей намерения и сущности](#)
- [Переход на версии 3 API для создания](#)
- [Переход на версию 3 интерфейсов API для прогнозирования](#)

## Дальнейшие действия

[Использование QnA Maker для ответов на вопросы](#)

# Развертывание ресурсов LUIS с помощью команд CLI Bot Framework для LUIS

27.03.2021 • 25 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Интерфейс командной строки Bot Framework позволяет автоматизировать управление LUIS приложениями. Из командной строки или скрипта можно создавать, обновлять и удалять свойства LUIS.

В этой статье объясняется, как развернуть ресурс LUIS. Сведения о том, как обновить существующий ресурс LUIS с помощью интерфейса командной строки BF, см. в разделе [обновление ресурсов Luis с помощью команд интерфейса командной строки "Bot Framework Luis"](#).

## Предварительные условия

- Знание [шаблонов Lu](#).
- Иметь проект Bot с файлами. lu.
- При работе с адаптивными диалогами необходимо иметь представление о следующих аспектах:
  - [Обработка естественного языка в адаптивных диалоговых окнах](#).
  - [Понимание языка в адаптивных диалоговых окнах](#).
  - как используется [распознаватель Luis](#).

## Использование команд интерфейса командной строки LUIS для включения LUIS в Bot

В этой статье описывается, как выполнять некоторые распространенные задачи с помощью интерфейса командной строки Bot Framework.

После создания ресурсов по языку проекта Bot вы можете выполнить действия, описанные в этой статье, чтобы получить LUIS работу.

## Создание ресурса для создания LUIS в Azure

Ресурс для разработки LUIS — это ресурс [azure Cognitive Services](#), созданный с помощью страницы [создания Cognitive Services](#) Azure. Вы можете считать это контейнером для приложений LUIS и модели или моделей, из которых состоят эти приложения LUIS. Ресурс разработки LUIS обеспечивает безопасный способ создания ресурсов LUIS. Это необходимо для выполнения необходимых действий, таких как создание, обновление, обучение и публикация приложения LUIS. Группа LUIS в интерфейсе командной строки Bot Framework предоставляет команды, необходимые для выполнения этих задач.

1. Перейдите на страницу [создания Cognitive Services](#) Azure.
2. В разделе **Параметры создания** выберите **Создание ресурса для создания Luis**.

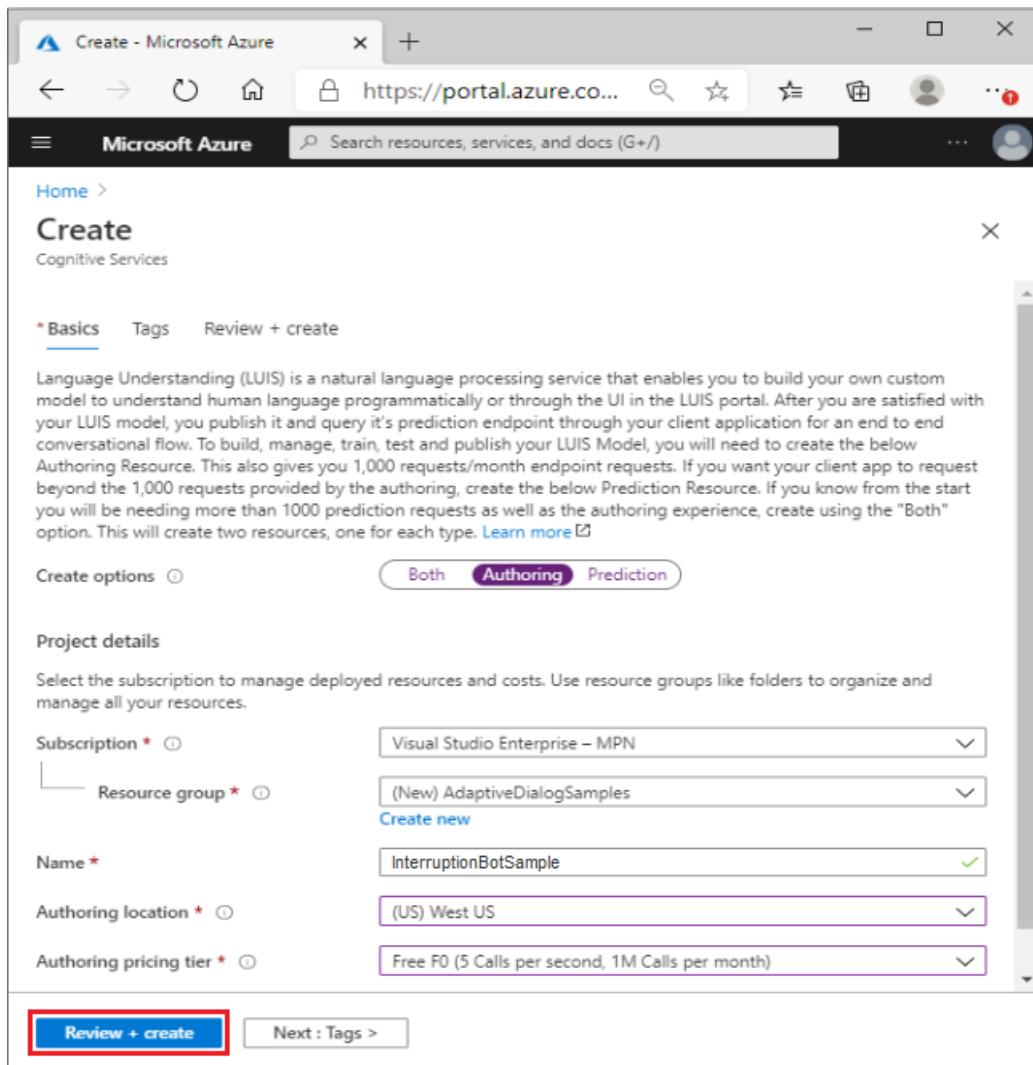
Create options

Both

Authoring

Prediction

3. Введите значения для каждого поля, а затем нажмите кнопку "**проверить и создать**".



#### NOTE

При вводе **группы ресурсов** и **имени** Помните, что эти значения нельзя изменить позже. Также обратите внимание, что значение, присваиваемое **имени**, будет являться частью URL-адреса **конечной точки**.

4. Проверьте значения, чтобы убедиться, что они верны, а затем нажмите кнопку **создать**.

Ресурс для разработки LUIS включает сведения, которые будут использоваться программой Bot для доступа к приложению LUIS:

- **Ключи.** Они называются *ключами подписки*, которые иногда называются *ключами разработки* при ссылке на ключи в ресурсе разработки Luis. Они создаются автоматически. Вам потребуется ключ разработки при ссылке на ресурс разработки LUIS для любого действия, например при создании приложения и моделей LUIS, которые будут подробно описаны в этой статье. Ключи можно найти в колонке **ключи и конечная точка** в ресурсе разработки Luis.
- **Конечная точка.** Он создается автоматически с помощью имени ресурса для создания LUIS, которое вы задаете при его создании. Он имеет следующий формат:  
`https://<luis-resource-name>.cognitiveservices.azure.com/`. При ссылке на ресурс разработки LUIS для любого действия, например при создании приложения и моделей LUIS, которые будут подробно описаны в этой статье. Ключ можно найти в колонке **ключи и конечная точка** в ресурсе разработки Luis.
- **Расположение.** Это регион Azure, который содержит ресурс для разработки LUIS. Этот параметр выбирается при создании ресурса для создания LUIS.

The screenshot shows the Microsoft Azure Cognitive Services Keys and Endpoint page for the 'interruptbotsample' resource. On the left, a sidebar lists various management options like Overview, Activity log, Access control (IAM), Tags, and Diagnose and solve problems. Under RESOURCE MANAGEMENT, 'Keys and Endpoint' is selected. The main content area displays two keys: KEY 1 and KEY 2, each with a copy icon. Below them are the ENDPOINT and LOCATION fields, both also with copy icons. A tip box at the top right provides guidance on key security and regeneration.

#### TIP

Существует два типа ключей подписки, связанных с ресурсом LUIS в зависимости от типа ресурса LUIS, на который вы ссылаетесь. У ресурса, создающего LUIS, есть ключ создания, а LUISный ресурс-прогноз имеет ключ прогноза, и оба они являются ключами подписки, и их можно найти в колонке *ключи и конечная точка*.

Дополнительные сведения см. в разделе [CREATE Luis Resources](#).

## Установка интерфейса командной строки для Bot Framework

Для работы с Bot Framework CLI требуется [Node.js](#).

1. Убедитесь, что установлена последняя версия NPM:

```
npm i -g npm
```

2. С помощью Node.js установите последнюю версию Bot Framework CLI из командной строки.

```
npm i -g @microsoft/botframework-cli
```

Дополнительные сведения см. в статье [Bot Framework CLI Tool](#).

## Создание модели LUIS

После создания всех отдельных файлов .lu, необходимых в проекте, их можно объединить для создания модели LUIS с помощью `luis:convert` команды. Это приводит к созданию JSON-файла, на который вы будете ссылаться при создании приложения LUIS, размещенного в Azure Cognitive Services в созданном ранее *ресурсе Luis Authoring*.

```
bf luis:convert -i <input-folder-name> -o <output-file-name> -r
```

В приведенном ниже примере команда выполняется в командной строке в корневом каталоге проекта.

Он выполнит поиск всех файлов. `lu` в каталоге *диалоговых окон* и из-за `-r` параметра все вложенные каталоги. Он сохранит файл с именем `LUISModel.json` в *выходном* каталоге.

```
bf luis:convert -i dialogs -o .\output\LUISModel.json -r
```

## Создание приложения LUIS

Созданный ранее *ресурс создания Luis* состоит из ваших ключей и конечных точек разработки, которые необходимы при создании приложения LUIS (Luis App). У вас может быть несколько приложений LUIS, связанных с одним ресурсом создания LUIS, и каждое приложение LUIS будет иметь собственное значение `appId`, которое будет предоставлено вам в рамках процесса создания. Это потребуется вам `appId` при ссылке на это приложение Luis в будущем. Приложение LUIS предоставит работу все функции, предоставляемые LUIS, в сочетании с данными приложения, которые вы указали в модели LUIS, созданной ранее из файлов Projects. `lu`.

Чтобы создать приложение LUIS, сделайте следующее:

```
luis:application:import --in <luis-model-json-file> --endpoint <endpoint> --subscriptionKey <subscription-key> --name <name> --versionId <initial-version-id>
```

В приведенном выше примере `luis:application:import` команда показывает команду с требуемыми параметрами. Дополнительные сведения о всех параметрах, доступных для этой команды, см. в разделе [BF Luis: Application: Import](#) файла `readme` интерфейса командной строки Luis.

Чтобы создать приложение LUIS без включения модели LUIS, см. команду [BF Luis: Application: Create](#).

The screenshot shows the Microsoft Azure portal interface. The top navigation bar includes the Microsoft Azure logo, a search bar, and various navigation icons. Below the navigation bar, the URL bar shows the path: Home > Microsoft.CognitiveServicesLUISAllInOne > MyLUISApplication. The main content area is titled "LUIS-Authoring-Resource | Keys and Endpoint". On the left, there is a sidebar with several options: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, RESOURCE MANAGEMENT (Quick start, Keys and Endpoint, Pricing tier, Networking, Identity, Billing By Subscription), and a "Hide Keys" button. The "Keys and Endpoint" option is highlighted with a red box. The main content area displays two keys (KEY 1 and KEY 2) and their corresponding endpoint and location. The endpoint is listed as `https://luis-authoring-resource.cognitiveservices.azure.com/` and the location as `westus`. The entire "KEY 1" row and the "ENDPOINT" row are also highlighted with red boxes.

На приведенном выше рисунке для создания приложения LUIS использовалась следующая команда, предполагая, что JSON-файл модели LUIS находится в подкаталоге с именем *Output*.

```
luis:application:import --in .\output\LUISModel.json --endpoint https://LUIS-Authoring-Resource.cognitiveservices.azure.com/ --subscriptionKey xxxxxxxxxxxxxxxxxxxxxxxx --name LUISApplication --versionId 0.1
```

## NOTE

Ключи на рисунке выше не являются допустимыми ключами. Они показаны только в демонстрационных целях. Эти ключи разработки обеспечивают безопасный способ доступа к ресурсам разработки LUIS. Не предоставляйте доступ к ключам другим пользователям.

## Получение appId из приложения LUIS

Вам потребуются `appId`, которые будут возвращены, когда `luis:application:import` команда завершится успешно, также потребуется `versionId`, что вы указали при создании приложения Luis. В сценариях, где эти сведения отсутствуют, или при создании скриптов для автоматизации этого процесса можно использовать команду [Luis: Application: показывать](#), чтобы получить эти сведения.

```
bf luis:application:show --appId <application-id> --endpoint <endpoint> --subscriptionKey <subscription-key>
```

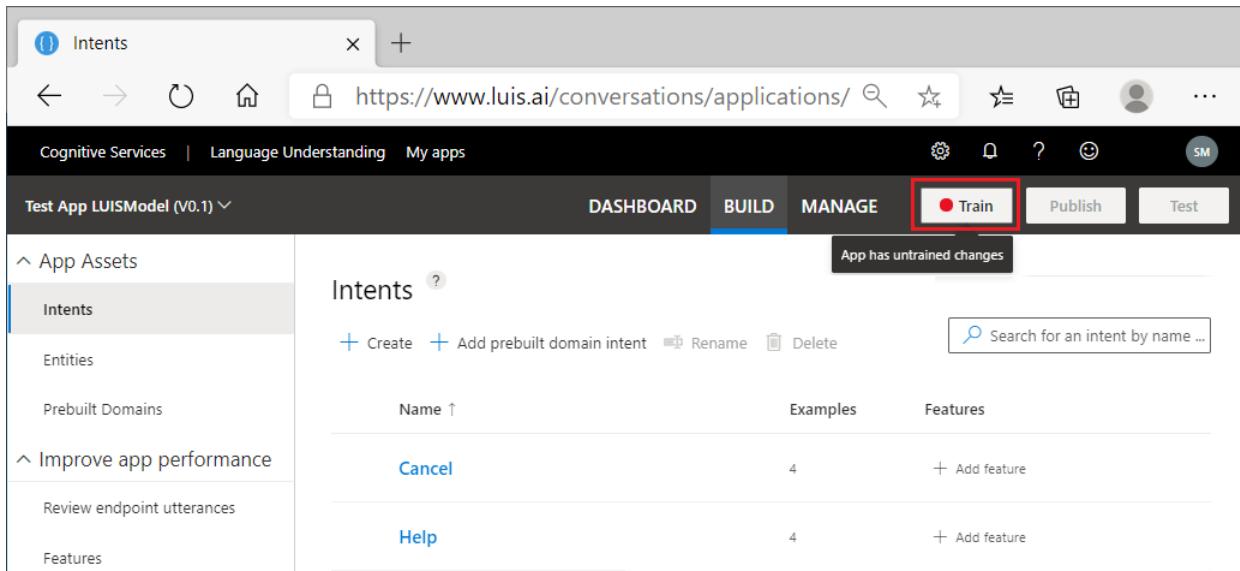
## Обучение приложения LUIS

Обучение — это процесс обучения приложения LUIS, чтобы улучшить понимание естественного языка. После внесения в модель обновлений, таких как добавление, изменение, маркировка или удаление сущностей, целей или фразы продолжительностью, необходимо обучить приложение LUIS. Дополнительные сведения см. в статье [обучение активной версии приложения Luis](#) в документации Luis.

Чтобы обучить приложение LUIS, используйте `luis:train:run` команду:

```
bf luis:train:run --appId <application-id> --versionId <version-id> --endpoint <endpoint> --subscriptionKey <subscription-key>
```

Это позволяет автоматизировать то, что вы обычно делаете на сайте LUIS:



The screenshot shows the Microsoft LUIS web interface. At the top, there's a navigation bar with links for 'Cognitive Services', 'Language Understanding', and 'My apps'. Below the navigation is a header for 'Test App LUISModel (V0.1)'. The main menu has tabs for 'DASHBOARD', 'BUILD' (which is selected), 'MANAGE', 'Train' (highlighted with a red box), 'Publish', and 'Test'. On the left, there's a sidebar with sections for 'App Assets' (selected), 'Intents', 'Entities', 'Prebuilt Domains', and 'Improve app performance' (with sub-options like 'Review endpoint utterances' and 'Features'). The main content area is titled 'Intents' and shows a table with two rows: 'Cancel' and 'Help'. Each row has columns for 'Name', 'Examples', and 'Features'. There are buttons for 'Create', 'Add prebuilt domain intent', 'Rename', and 'Delete' at the top of the table, and a search bar below it. A message 'App has untrained changes' is displayed above the table.

Дополнительные сведения об использовании этой команды см. в разделе [BF Luis: поезд: Run](#) в BF CLI Luis readme.

**TIP**

После обучения приложения LUIS следует [протестировать](#) его с помощью примера фразы продолжительностью, чтобы проверить правильность распознавания целей и сущностей. Если распознавание выполняется неправильно, внесите изменения в приложение, а затем обучите его и протестируйте еще раз. Это тестирование можно выполнить вручную на сайте LUIS. Дополнительные сведения см. в статье [тестирование utterance](#).

## Публикация приложения LUIS

После завершения сборки, обучения и тестирования активного приложения LUIS сделайте его доступным для клиентского приложения, опубликовав его в конечной точке. Это можно сделать с помощью `luis:application:publish` команды.

```
bf luis:application:publish --appId <application-id> --versionId <version-id> --endpoint <endpoint> --subscriptionKey <subscription-key>
```

Дополнительные сведения об использовании этой команды см. в разделе [BF Luis: Application: Publish](#) в BF CLI Luis readme.

Сведения о публикации приложения LUIS см. в статье [Публикация активного, обученного приложения в промежуточной или производственной конечной точке](#).

## Создание исходного кода

### Создание класса C# для результатов модели

`luis:generate:cs` Команда создает строго типизированный исходный код C# из модели Luis (JSON).

Выполните следующую команду, чтобы создать CS-представление модели LUIS:

```
bf luis:generate:cs -i <luis-model-file> -o <output-file-name> --className <class-name>
```

Дополнительные сведения об использовании этой команды см. в разделе [BF Luis: Generate: CS](#) в файле readme платформы Bot CLI Luis.

### Создание типа TypeScript для результатов модели

`luis:generate:ts` Команда создает строго типизированный исходный код TypeScript из модели Luis (JSON).

Выполните следующую команду, чтобы создать представление. TS модели LUIS:

```
bf luis:generate:ts -i <luis-model-file> -o <output-file-name> --className <class-name>
```

Дополнительные сведения об использовании этой команды см. в разделе [BF Luis: Generate: TS](#) в файле readme интерфейса командной строки BF Luis.

## Создайте и обучите приложение LUIS, а затем опубликуйте его с помощью команды Build.

Полезно понимать, как работает процесс развертывания приложения LUIS, и после завершения этой статьи до этого момента вы получите более полное представление о процессах, связанных с созданием модели LUIS, используя эту модель для создания приложения LUIS в ресурсе Azure Cognitive Services, а затем Обучите и опубликуйте его с помощью команд интерфейса командной строки Bot Framework.

Эти команды обеспечивают большую гибкость при адаптации сценариев к конкретным потребностям. Вы можете использовать `luis:build` команду, чтобы создать или обновить, а затем обучить и опубликовать приложение Luis. Однако использование `luis:build` команды предоставляет меньше параметров для управления процессом.

Для каждого Lu-файла, включая файлы с расширением Lu для каждого языкового стандарта, команда Build объединяет все следующие действия в одну команду:

1. Создает одну модель LUIS для [каждого языкового стандарта](#), найденного с помощью существующих файлов. lu.
2. С помощью этой модели создается новое приложение LUIS в указанном ресурсе Azure Cognitive Services, если оно не существует, в противном случае будет обновлено существующее приложение LUIS.
3. При обновлении существующего приложения LUIS оно автоматически увеличивает значение versionId и при необходимости удаляет старую версию.
4. Выполните обучение нового или обновленного приложения LUIS, а затем опубликуйте его.
5. Если включить `--dialog` параметр, будут выведены `.dialog` файлы определений, которые могут использоваться [распознавателем Luis](#) при разработке с использованием [декларативного подхода](#). Это объясняется в разделе [файл диалогового окна](#).

## Использование команды Build

Команда LUIS Build с необходимыми параметрами:

```
bf luis:build --in <input-file-or-folder> --out <output-file-or-folder> --botName <bot-name> --authoringKey <subscription-key> --region <authoring-region>
```

`luis:build` Команда создаст все необходимые ресурсы из локальных файлов. lu. При использовании `--in` параметра `luis:build` создает одно приложение Luis для каждого файла. lu, найденного для каждого языкового стандарта.

### Обязательный Luis: параметры сборки

- `--in` : Каталог, включая подкаталоги, в котором будут искаться файлы. lu.
- `--out` : Каталог для сохранения выходных файлов. Сюда входят все файлы распознавателя, а также файл параметров. Если опустить `--out` параметр, файлы не будут сохранены на диск, а в консоль будут записываться только ключи разработки и конечная точка из файла параметров.
- `--botName` — Имя программы-робота. Он будет использоваться в качестве префикса для имени создаваемых приложений LUIS.
- `--authoringKey` — То же значение, что и в Субскриптионкэй, используемом во всех предыдущих командах, обсуждаемых в этой статье.
- `--region` : Определяет регион для публикации приложений LUIS. по умолчанию используется значение *westus*, если не указано.

Дополнительные сведения о дополнительных параметрах см. в разделе [BF Luis: Build](#) в файле readme интерфейса командной строки BF.

Кроме того, можно включить эти обязательные данные, а также любые другие параметры в файле конфигурации и обращаться к ним с помощью `--luConfig` параметра.

### Файл конфигурации сборки LUIS

Ниже приведен пример `luconfig.json` в файле, на который можно сослаться с помощью `--luConfig` параметра.

```
{
  "in": "dialogs",
  "out": "generated",
  "botName": "MyProject",
  "authoringKey": "<your-32-digit-subscription-key>",
  "region": "westus",
  "schema": "app.schema",
  "defaultCulture": "en-us",
  "deleteOldVersion": true,
  "dialog": "multiLanguage",
  "fallbackLocale": "en-us",
  "force": true,
  "suffix": "<value-to-replace-username>"
}
```

Кроме того, `models` в файле `luconfig.json` для файла можно указать, какие файлы. `.lu` в проекте соответствуют приложению Luis. Это особенно полезно, если вы используете внешние ссылки в файлах. `.lu`, поэтому не каждый отдельный Lu-файл рассматривается как приложение LUIS. Каждый файл, перечисленный в `models` разделе, станет Luis приложением, а все файлы. `.lu`, отсутствующие в списке, станут частью приложения Luis, которое создается из любого файла с расширением. `.lu`, который ссылается на него. Файлы, упоминаемые в нескольких файлах. `.lu`, будут дублироваться в каждом LUIS приложении, которое ссылается на него. Любой файл. `.lu`, не указанный явно в разделе Models `luconfig.json` в файле или упоминаемом в одном из перечисленных файлов. `.lu`, будет проигнорирован.

В примере ниже показан раздел `моделей` `luconfig.json` в файле:

```
// Each model is a LUIS application.
"models": [
    // Each line is to an lu file and corresponds to a LUIS application.
    // relative paths here are relative to the luconfig.json file itself.
    "./dialog/AddToDoDialog/AddToDoDialog.lu",
    "./dialog/Common/Common.lu",
    "./dialog/DeleteToDoDialog/DeleteToDoDialog.lu",
    "./dialog/GetUserProfileDialog/GetUserProfileDialog.lu",
    "./dialog/RootDialog/RootDialog.lu",
    "./dialog/ViewToDoDialog/ViewToDoDialog.lu"
]
```

После создания этого файла конфигурации необходимо просто сослаться на него в `luis:build` команде. Например:

```
bf luis:build --luConfig luconfig.json
```

## LU и несколько вариантов языка

Каждый [файл](#). `lu` может иметь несколько вариантов языка, и `luis:build` команда будет создавать приложение Luis для каждого поддерживаемого языка.

Шаблон для имени Lu-файла, когда используются дополнительные языковые стандарты, выглядит следующим образом:

```
<file-name>.<locale>.lu
```

Например:

```
RootDialog.en-us.lu  
RootDialog.fr-fr.lu  
RootDialog.de-de.lu  
etc.
```

В приведенном выше примере для каждого из файлов. `lu` будет создана собственная уникальная модель для конкретного языка, что приведет к созданию одного LUIS приложения для каждого из трех файлов. `lu`.

#### TIP

- В качестве альтернативы включению языкового стандарта в имя файла его можно включить в файл. `lu` в качестве части сведений о конфигурации. Это полезно, если требуется более гибкое соглашение об именовании файлов. Например, в файле. `lu` для французского языка можно добавить:  
`> !# @app.culture = fr-fr`. Дополнительные сведения см. в [описании модели](#) в справочнике по [формату файлов Lu](#).
- Если языковой стандарт не может быть определен из имени файла и он не включен в сведения о конфигурации файлов. `lu`, будет использоваться значение, указанное в `--defaultCulture` параметре команды сборки. Если `--defaultCulture` параметр не указан, то языковой стандарт будет установлен в значение `en-us`.

## Созданные приложения LUIS

Каждое приложение LUIS, созданное от вашего имени, будет называться с помощью сочетания `{botName}` значения, указанного при выполнении `luis:build` команды, имени пользователя, выполнившего вход, и имени логического файла, включая языковой стандарт.

Имена приложений LUIS будут использовать следующий формат:

```
{botName}{{suffix}}-{file-name}-{locale}.lu
```

Например, если Ботнаме — *MyProject*, а имя пользователя — *юуританака*, а имя файла является *расширением*, то имена приложений Luis будут выглядеть следующим образом:

```
MyProject(YuuriTanaka)-GetAddresss.en-us.lu  
MyProject(YuuriTanaka)-GetAddresss.fr-fr.lu  
MyProject(YuuriTanaka)-GetAddresss.de-de.lu
```

Одно и то же имя приложения LUIS будет использоваться в каждом регионе Azure с внутренними конечными точками.

#### TIP

Включение имени пользователя в имя приложения LUIS позволяет нескольким разработчикам работать независимо. Это значение создается автоматически с использованием имени пользователя, выполнившего вход, однако его можно переопределить с помощью `--suffix` параметра.

## Файл параметров, созданный с помощью команды сборки

Все файлы с расширением `Lu` для каждого языкового стандарта будут иметь одно приложение LUIS, а выходные данные `luis:build` команды будут содержать один файл параметров, содержащий список всех идентификаторов приложений Luis, которые были созданы для каждого языкового стандарта.

Например, если имя пользователя, выполнившего вход в систему, — *юуританака* и вы используете область разработки *westus*, ваше имя файла будет выглядеть так:

## **luis.settings.YuuriTanaka.westus.js на**

Пример файла параметров:

```
{  
  "luis": {  
    "RootDialog_en_us_lu": "<LUIS-App-ID-for-en-us-locale>",  
    "RootDialog_fr_fr_lu": "<LUIS-App-ID-for-fr-fr-locale>"  
  }  
}
```

### **Файл диалогового окна**

При включении `--dialog` параметра будет создан файл Dialog для каждого из файлов. Iu, по одному для каждого языкового стандарта.

#### **IMPORTANT**

`--schema` Параметр используется вместе с `- - dialog` параметром. Включение `--schema` параметра гарантирует, что каждый созданный файл диалога будет иметь ссылку на файл корневой схемы проектов. Этот файл схемы содержит схемы всех компонентов, используемых программой Bot. Каждому потребителю декларативных файлов, включая [Composer] [Composer], требуется файл схемы. Если в проекте нет файла схемы, его можно создать с помощью `dialog:merge` команды. Перед выполнением команды необходимо выполнить эту команду `luis:build`. Дополнительные сведения см. в статье об [использовании декларативных ресурсов в адаптивных диалоговых окнах](#).

Диалоговое окно Luis: сборка и параметры схемы:

- **диалоговое окно.** Существует два допустимых значения параметра диалогового окна `multiLanguage` и `crosstrained`.
- **схема.** Это принимает относительный путь и имя файла, указывающего на файл схемы Bot.

Эти файлы будут записаны в каталог, указанный в `out` параметре. Например:

```
RootDialog.en-us.lu.dialog <- LuisRecognizer for en-us locale  
RootDialog.fr-fr.lu.dialog <- LuisRecognizer for fr-fr locale  
RootDialog.lu.dialog      <- MultiLanguageRecognizer configured to use all locales
```

Ниже приведен пример файла мультиязычного распознавателя:

```
{  
  "$schema": "app.schema",  
  "$kind": "Microsoft.MultiLanguageRecognizer",  
  "id": "lu_RootDialog",  
  "recognizers": {  
    "en-us": "RootDialog.en-us.lu",  
    "fr-fr": "RootDialog.fr-fr.lu",  
    "": "RootDialog.en-us.lu"  
  }  
}
```

Эти файлы будут использоваться, если вы используете декларативный подход к разработке программы Bot, и вам нужно будет добавить ссылку на этот распознаватель в файл адаптивных диалогов `.dialog`. В следующем примере `"recognizer": "RootDialog.lu"` выполняется поиск распознавателя, определенного в файле **рутдиалог. Iu. Dialog**:

```
{  
    "$schema": "app.schema",           ← The schema file created when the schema option is used  
    "$kind": "Microsoft.AdaptiveDialog", ← This is a dialog  
    "recognizer": "RootDialog.lu",      ← This is referencing the file named RootDialog.lu.dialog  
    "triggers": [  
        {$kind: "OnIntent"}           ← The OnIntent Trigger is required to react to the  
    ]                                    RecognizedIntent event  
}
```

Дополнительные сведения см. [в разделе Использование декларативных ресурсов в адаптивных диалоговых окнах](#).

## Дополнительные сведения

- [Обновление моделей LUIS](#)

# Обновление ресурсов LUIS с помощью команд CLI Bot Framework для LUIS

27.03.2021 • 11 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Интерфейс командной строки Bot Framework (BF CLI) позволяет автоматизировать управление ресурсами LUIS. Из командной строки или скрипта можно создавать, обновлять и удалять свойства LUIS.

В этой статье объясняется, как обновить существующий ресурс LUIS. Сведения о начале работы и о развертывании ресурсов LUIS с помощью BF CLI см. в статье [развертывание ресурсов Luis с помощью команд CLI для Bot Framework](#).

## Предварительные требования

- Знание [шаблонов Lu](#).
- Иметь проект Bot с файлами. lu.
- При работе с адаптивными диалогами необходимо иметь представление о следующих аспектах:
  - [Обработка естественного языка в адаптивных диалоговых окнах](#).
  - [Понимание языка в адаптивных диалоговых окнах](#).
  - как используется [распознаватель Luis](#).

## Использование команд интерфейса командной строки LUIS для обновления ресурсов LUIS, используемых в Bot

В этой статье описаны следующие шаги по обновлению существующих ресурсов разработки LUIS в Azure с помощью интерфейса командной строки Bot Framework. В этих инструкциях объясняется, как использовать версии приложений LUIS для создания резервной копии активной версии приложения LUIS перед созданием новой активной версии.

1. [Установка интерфейса командной строки для Bot Framework](#)
2. [Получение параметров из приложения LUIS](#)
3. [Создание модели LUIS](#)
4. [Удалить версию резервной копии, если она существует](#)
5. [Сохранить текущую версию в качестве резервной копии](#)
6. [Импорт новой версии модели LUIS](#)
7. [Обучение приложения LUIS](#)
8. [Публикация приложения LUIS](#)
9. [Создание исходного кода](#)

## Установка интерфейса командной строки для Bot Framework

Для работы с Bot Framework CLI требуется [Node.js](#).

1. Убедитесь, что установлена последняя версия NPM:

```
npm i -g npm
```

2. С помощью Node.js установите последнюю версию Bot Framework CLI из командной строки.

```
npm i -g @microsoft/botframework-cli
```

Дополнительные сведения см. в статье [Bot Framework CLI Tool](#).

## Получение параметров из приложения LUIS

Для завершения процесса обновления ресурсов LUIS вам потребуется *идентификатор приложения Luis* и *идентификатор активной версии*. Эту информацию можно получить двумя способами. В этом разделе будут объяснены и, и при использовании каждого из них. Независимо от того, какой подход вы используете, вам потребуется ключ подписки и конечная точка, с которыми связано ваше приложение LUIS. Эти сведения можно найти в *ресурсе создания Luis* в Azure в колонке **ключи и конечная точка**.

The screenshot shows the Microsoft Azure portal interface. In the top navigation bar, the 'Microsoft Azure' logo and search bar are visible. Below the navigation bar, the URL 'Home > Microsoft.CognitiveServicesLUISAllInOne > MyLUISApplication' is shown. The main content area has a title 'LUIS-Authoring-Resource | Keys and Endpoint'. On the left, there's a sidebar with 'Cognitive Services' and a list of options: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, RESOURCE MANAGEMENT (Quick start, Keys and Endpoint, Pricing tier, Networking, Identity, Billing By Subscription), and a 'Hide Keys' button. The 'Keys and Endpoint' option is highlighted with a red box. The main panel displays two keys ('KEY 1' and 'KEY 2') and their corresponding endpoint ('ENDPOINT'). The endpoint value 'https://luis-authoring-resource.cognitiveservices.azure.com/' is also highlighted with a red box. A note at the top right says: 'These keys are used to access your Cognitive Service API. Do not share your keys. Store them securely—for example, using Azure Key Vault. We also recommend regenerating these keys regularly. Only one key is necessary to make an API call. When regenerating the first key, you can use the second key for continued access to the service.' Below the endpoint, the location 'westus' is listed.

Если вы знакомы с ИДЕНТИФИКАТОРом приложения LUIS, но вам нужно получить идентификатор активной версии, можно использовать `luis:application:show` команду. Будет возвращена только информация для указанного приложения LUIS.

```
bf luis:application:show --appId <application-id> --endpoint <endpoint> --subscriptionKey <subscription-key>
```

Дополнительные сведения об использовании этой команды см. в разделе [BF Luis: Application: Показать](#) в BF CLI Luis readme.

Если вы не знакомы с ИДЕНТИФИКАТОРом приложения LUIS, можно использовать команду `Luis: Application: List`, чтобы получить его вместе с идентификатором активной версии. Эта команда выводит список всех приложений LUIS, созданных в указанном ресурсе разработки LUIS. Идентификатор приложения LUIS возвращается как `id`, а идентификатор активной версии возвращается в виде `activeVersion`.

```
bf luis:application:list --endpoint <endpoint> --subscriptionKey <subscription-key>
```

Дополнительные сведения об использовании этой команды см. в разделе [BF Luis: Application: List](#) статьи

## Создание модели LUIS

Каждый раз, когда вы вносите изменения в любой из отдельных файлов. luis, используемых в проекте, необходимо создать новую модель LUIS с помощью `luis:convert` команды. Эта новая модель будет использоваться для обновления приложения LUIS, размещенного в Azure. Это позволит сделать эти изменения действительными в коде робота.

```
bf luis:convert -i <input-folder-name> -o <output-file-name> -r --name <name>
```

Дополнительные сведения об использовании этой команды см. в разделе [BF Luis: Convert](#) в BF CLI Luis readme.

### TIP

Этот `name` параметр не является обязательным, однако если этот параметр не включен, необходимо вручную обновить код JSON модели Luis перед импортом, иначе возникнет ошибка:

```
Failed to import app version: Error: Application name cannot be null or empty.
```

## Удалить версию резервной копии

Перед созданием новой версии модели LUIS можно создать резервную копию активной версии. При следующем создании нового обновления модели LUIS может потребоваться удалить старую резервную копию перед созданием новой резервной копии. Для этого используйте команду `luis:version:delete`.

```
bf luis:version:delete --appId <application-id> --versionId <version-id> --endpoint <endpoint> --subscriptionKey <subscription-key>
```

Дополнительные сведения об использовании этой команды см. в разделе [BF Luis: версия: Delete](#) в файле readme интерфейса командной строки BF Luis.

### IMPORTANT

Следите за тем, чтобы команда не предобразовала ошибку `luis:application:delete` `luis:version:delete`.  
`luis:application:delete` Команда окончательно удалит приложение Luis вместе со всеми версиями всех связанных с ним моделей Luis. `luis:version:delete` Команда удалит только указанную версию. Эта команда удалит версию без предупреждения, даже если это единственная версия модели.

## Сохранить текущую версию в качестве резервной копии

Перед импортом новой версии модели LUIS можно создать резервную копию активной версии. Это можно сделать с помощью команды `luis:version:rename`. Вам потребуется версия `versionId` активной версии, полученная из предыдущего раздела [Получение параметров из приложения Luis](#), и можно задать `newVersionId` значение "Backup", чтобы указать, что это ваша версия резервной копии.

```
bf luis:version:rename --appId <application-id> --versionId <version-id> --newVersionId <new-version-id> --endpoint <endpoint> --subscriptionKey <subscription-key>
```

Дополнительные сведения об использовании этой команды см. в разделе [BF Luis: Version: Rename](#) в BF CLI Luis readme.

**TIP**

Вы не ограничены числовыми символами для versionId. Если вы назначите имя резервной копии "Backup", это упростит процесс удаления версии резервной копии.

## Импорт новой версии модели LUIS

Теперь вы можете импортировать новую версию модели, созданную в разделе [Создание модели Luis](#) этой статьи. Для этого используется `luis:version:import`.

Чтобы обновить приложение LUIS, выполните следующие действия.

```
luis:version:import --in <luis-model-json-file> --endpoint <endpoint> --subscriptionKey <subscription-key> -  
-appId <app-id> --versionId <version-id>
```

Дополнительные сведения о всех параметрах, доступных для этой команды, см. в разделе [BF Luis: Application: Import](#) файла readme интерфейса командной строки Luis.

## Обучение приложения LUIS

Обучение — это процесс обучения приложения LUIS, чтобы улучшить понимание естественного языка. После внесения обновлений в модель необходимо обучить приложение LUIS. Дополнительные сведения см. в статье [обучение активной версии приложения Luis](#) в документации Luis.

Чтобы обучить приложение LUIS, используйте `luis:train:run` команду:

```
bf luis:train:run --appId <application-id> --versionId <version-id> --endpoint <endpoint> --subscriptionKey  
<subscription-key>
```

Дополнительные сведения об использовании этой команды см. в разделе [BF Luis: поезд: Run](#) в BF CLI Luis readme.

**TIP**

После обучения приложения LUIS следует [протестировать](#) его с помощью примера фразы продолжительностью, чтобы проверить правильность распознавания целей и сущностей. Если это не так, обновите приложение LUIS, выполните импорт, обучение и тестирование еще раз. Это тестирование можно выполнить вручную на сайте LUIS. Дополнительные сведения см. в статье [тестирование utterance](#).

## Публикация приложения LUIS

После завершения сборки, обучения и тестирования активного приложения LUIS сделайте его доступным для клиентского приложения, опубликовав его в конечной точке. Это можно сделать с помощью `luis:application:publish` команды.

```
bf luis:application:publish --appId <application-id> --versionId <version-id> --endpoint <endpoint> --  
subscriptionKey <subscription-key>
```

Дополнительные сведения об использовании этой команды см. в разделе [BF Luis: Application: Publish](#) в BF CLI Luis readme.

Сведения о публикации приложения LUIS см. в статье [Публикация активного, обученного приложения в](#)

промежуточной или производственной конечной точке.

## Создание исходного кода

### Создание класса C# для результатов модели

`luis:generate:cs` Команда создает строго типизированный исходный код C# из модели Luis (JSON).

Выполните следующую команду, чтобы создать CS-представление модели LUIS:

```
bf luis:generate:cs -i <luis-model-file> -o <output-file-name> --className <class-name>
```

Дополнительные сведения об использовании этой команды см. в разделе [BF Luis: Generate: CS](#) в файле `readme` платформы Bot CLI Luis.

### Создание типа TypeScript для результатов модели

`luis:generate:ts` Команда создает строго типизированный исходный код TypeScript из модели Luis (JSON).

Выполните следующую команду, чтобы создать представление TS модели LUIS:

```
bf luis:generate:ts -i <luis-model-file> -o <output-file-name> --className <class-name>
```

Дополнительные сведения об использовании этой команды см. в разделе [BF Luis: Generate: TS](#) в файле `readme` интерфейса командной строки BF Luis.

# Использование QnA Maker для ответов на вопросы

27.03.2021 • 12 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

QnA Maker создает слой вопросов и ответов для диалога на основе ваших данных. Это позволяет вашему роботу отправить вопрос в QnA Maker и получить ответ без необходимости анализировать и интерпретировать цель вопроса.

Одна из основных сложностей при создании собственной службы QnA Maker — заполнить ее начальным набором вопросов и ответов. Во одних случаях вопросы и ответы уже существуют в таком содержимом, как разделы с вопросами и ответами или другая документация. В других случаях вы можете настроить ответы на вопросы в более естественном разговорном стиле.

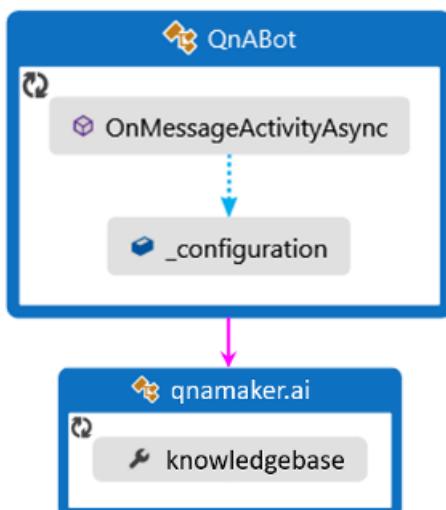
## Предварительные требования

- Учетная запись [QnA Maker](#).
- Знания о [работе ботов](#), [QnA Maker](#) и [управлении ресурсами бота](#).
- Копия образца QnA Maker (Simple) в [C#](#), [JavaScript](#) или [Python](#).

## Об этом примере

Чтобы использовать QnA Maker в боте, создайте базу знаний на портале [QnA Maker](#), как показано в следующем разделе. После этого бот сможет использовать базу знаний, чтобы отвечать на вопросы пользователя.

- [C#](#)
- [JavaScript](#)
- [Python](#)



`OnMessageActivityAsync` вызывается для каждого полученного блока данных, введенных пользователем. При вызове он обращается к `_configuration` информации, хранящейся в `appsetting.json` файле примера кода, чтобы найти значение для подключения к предварительно настроенной базе знаний QnA Maker.

Введенные пользователем данные передаются в эту базу знаний, а наиболее точный полученный ответ отображается для пользователя.

## Создание службы QnA Maker и публикация базы знаний

1. Создайте службу QnA Maker.
2. Создайте базу знаний на основе файла `smartLightFAQ.tsv`, который расположен в папке `CognitiveModels` этого примера проекта. Назовите базу знаний QnA и используйте файл **смартлигтфак. tsv**, чтобы заполнить его.

Этот способ можно также использовать для доступа к собственным базам знаний QnA Maker.

### NOTE

В документации по QnA Maker содержатся инструкции по [созданию](#) службы в Azure, а также по [созданию, обучению и публикации базы знаний](#).

## Получение значений для подключения бота к базе знаний

1. На сайте [QnA Maker](#) выберите свою базу знаний.
2. Откройте базу знаний, перейдите на вкладку **Параметры**. Запишите значение, отображаемое для параметра *имя службы*. Это значение используется для поиска нужной базы знаний в интерфейсе портала QnA Maker. Он не используется для подключения приложения-робота к этой базе знаний.
3. Прокрутите вниз, чтобы найти **сведения о развертывании**, и запишите следующие значения из образца HTTP-запроса POST.
  - POST /knowledgebases/<knowledge-base-id>/generateAnswer
  - Host: <your-host-url>
  - Авторизация: EndpointKey <your-endpoint-key>

URL-адрес узла будет начинаться с `https://` и заканчиваться `/кнамакер`, например `https://Azure.NET/qnamaker`. Чтобы подключиться к базе знаний QnA Maker, боту потребуется идентификатор базы знаний, URL-адрес узла и ключ конечной точки.

## Обновление файла параметров

Во-первых, добавьте в файл параметров сведения, необходимые для доступа к базе знаний: имя узла, ключ конечной точки и идентификатор базы знаний (`kbld`). Это значения, сохраненные на вкладке **Параметры** базы знаний в QnA Maker.

Если это развертывание не предназначено для рабочей среды, поля идентификатора приложения и пароля можно оставить пустыми.

### NOTE

Чтобы добавить базу знаний QnA Maker в существующее приложение бота, обязательно создайте понятные описательные заголовки для записей QnA. Параметр `name` в этом разделе предоставляет ключ для доступа к этой информации из приложения.

- [C#](#)
- [JavaScript](#)
- [Python](#)

`appsettings.json`

```
{  
    "MicrosoftAppId": "",  
    "MicrosoftAppPassword": "",  
    "QnAKnowledgebaseId": "",  
    "QnAEndpointKey": "",  
    "QnAEndpointHostName": ""  
}
```

## Настройка экземпляра QnA Maker

Для начала мы создадим объект для доступа к базе знаний QnA Maker.

- [C#](#)
- [JavaScript](#)
- [Python](#)

Убедитесь, что пакет NuGet **Microsoft.Bot.Builder.AI.QnA** установлен для вашего проекта.

В **QnABot.cs** в `OnMessageActivityAsync` методе создайте экземпляр **QnAMaker**. **QnABot** Класс также используется в тех местах, где имена сведений о соединении, сохраненные в **appsettings.js** выше, извлекаются в. Если вы выбрали другие имена для сведений о подключении к базе знаний в файле параметров, обязательно обновите имена в этом методе, чтобы они соответствовали выбранным.

### Bots/QnABot.cs

```
var qnaMaker = new QnAMaker(new QnAMakerEndpoint  
{  
    KnowledgeBaseId = _configuration["QnAKnowledgebaseId"],  
    EndpointKey = _configuration["QnAEndpointKey"],  
    Host = _configuration["QnAEndpointHostName"]  
},  
null,  
httpClient);
```

## Вызов QnA Maker из кода бота

- [C#](#)
- [JavaScript](#)
- [Python](#)

Когда программа-роботу требуется ответ от **QnAMaker**, вызовите `GetAnswersAsync` метод из кода **Bot**, чтобы получить соответствующий ответ на основе текущего контекста. Если вы используете собственную базу знаний, измените приведенное ниже сообщение *без ответов*, чтобы предоставить полезные инструкции для пользователей.

### Bots/QnABot.cs

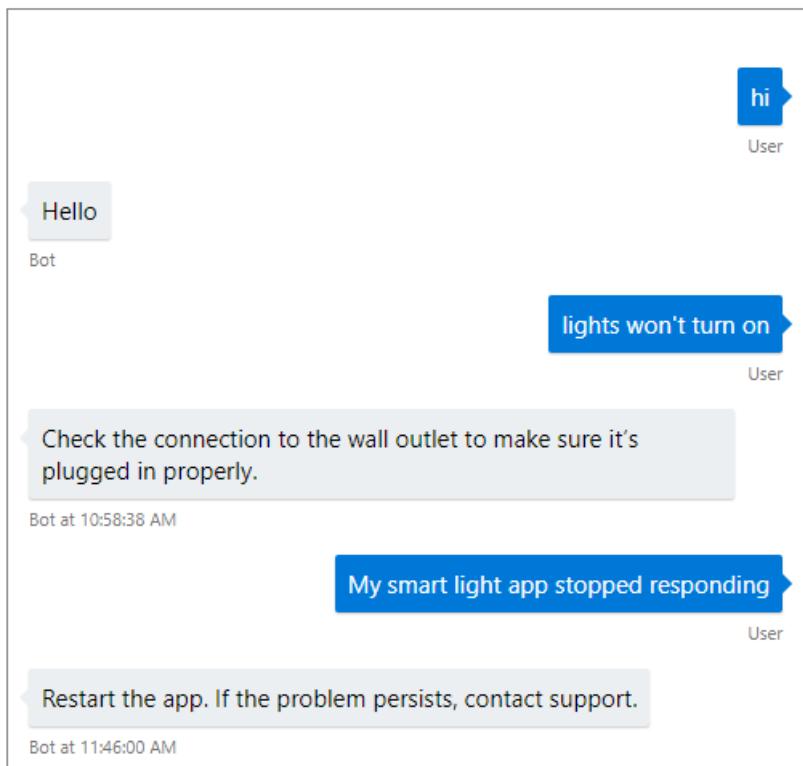
```
var options = new QnAMakerOptions { Top = 1 };

// The actual call to the QnA Maker service.
var response = await qnaMaker.GetAnswersAsync(turnContext, options);
if (response != null && response.Length > 0)
{
    await turnContext.SendActivityAsync(MessageFactory.Text(response[0].Answer), cancellationToken);
}
else
{
    await turnContext.SendActivityAsync(MessageFactory.Text("No QnA Maker answers were found."),
cancellationToken);
}
```

## Тестирование бота

Выполните этот пример на локальном компьютере. Установите [эмulateor Bot Framework](#), если вы еще этого не сделали. Дальнейшие инструкции см. в файле сведений примера ([C#, JavaScript, Python](#)).

Запустите эмулятор, подключитесь к роботу и отправьте сообщение, как показано ниже.



## Дополнительные сведения

В примере **многоэтапного диалога** QnA Maker ([на C#, JavaScript и Python](#)) показывается, как использовать диалог QnA Maker для поддержки запросов для дальнейших действий и функций активного обучения QnA Maker.

- QnA Maker поддерживает дополнительные (многоэтапные) подсказки. Если база знаний QnA Maker требует дополнительных сведений от пользователя, QnA Maker отправляет сведения о контексте, которые можно использовать при отправке запросов пользователю. Эти сведения затем используются для дополнительных обращений к службе QnA Maker. Поддержка этой функции добавлена в пакет SDK для Bot Framework версии 4.6.

Чтобы создать такую базу знаний, воспользуйтесь статьей [Use follow-up prompts to create multiple turns of a conversation](#) (Использование дальнейших подсказок для создания диалога с несколькими

шагами) из документации по QnA Maker.

- QnA Maker также поддерживает предложения для активного обучения, позволяя усовершенствовать базу знаний со временем. Диалог QnA Maker поддерживает явные отзывы для функции активного обучения.

Сведения о том, как включить эту функцию в базе знаний, см. в документации QnA Maker по [предложениям для активного обучения](#).

## Дальнейшие действия

QnA Maker можно объединять с другими службами Cognitive Services, чтобы сделать бота еще более мощным. Средство подготовки к отправке предоставляет способ объединения QnA с распознаванием речи (LUIS) в боте.

[Combine LUIS apps and QnA services using the Dispatch tool](#) (Объединение приложений LUIS и служб QnA с помощью средства Dispatch)

# Развертывание базы знаний QnA Maker с помощью команд интерфейса командной строки Bot Framework qnamaker

27.03.2021 • 25 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Интерфейс командной строки (CLI) для платформы Bot позволяет автоматизировать управление QnA Maker базой знаний (KB). Она позволяет создавать, обновлять и удалять QnA Maker KB из командной строки или скрипта. В этой статье объясняется, как развернуть базу знаний QnA Maker KB в QnA Maker KB в Azure.

## Предварительные требования

- Основные сведения о QnA Maker.
- Знание [формата файла QnA](#).
- Иметь проект Bot с QnA-файлами.
- При работе с адаптивными диалогами необходимо иметь представление о следующих аспектах:
  - [Обработка естественного языка в адаптивных диалоговых окнах](#).
  - как используется [распознаватель QnA Maker](#).

## Использование команд интерфейса командной строки qnamaker для включения QnA Maker в Bot

В этой статье описывается, как выполнять некоторые распространенные задачи, используемые для развертывания QnA Maker KB с помощью интерфейса командной строки Bot Framework.

- [Развертывание базы знаний QnA Maker с помощью команд интерфейса командной строки Bot Framework qnamaker](#)
  - [Необходимые компоненты](#)
  - [Использование команд интерфейса командной строки qnamaker для включения QnA Maker в Bot](#)
  - [Создание ресурса QnA Maker в Azure Cognitive Services](#)
  - [Установка интерфейса командной строки для Bot Framework](#)
  - [Создание файла инициализации QnA Maker](#)
  - [Создание модели QnA Maker](#)
  - [Создание базы знаний QnA Maker](#)
  - [Тестирование базы знаний QnA Maker](#)
  - [Публикация базы знаний QnA Maker](#)
  - [Создание базы знаний QnA Maker и ее публикация в рабочей среде с помощью команды Build](#)
    - [Использование команды Build](#)
      - [Qnamaker: параметры сборки](#)
      - [Файл конфигурации qnamaker](#)
      - [QnA и несколько вариантов языка](#)
      - [QnA Maker созданных баз знаний](#)

- Файл параметров, созданный с помощью команды сборки
- Файл диалогового окна
- Дополнительные сведения

После создания файлов QnA Maker KB. QnA в проекте Bot вы можете выполнить действия, описанные в этой статье, чтобы создать QnA Maker КБ. Если у вас нет проекта с файлами QnA Maker KB. QnA, можно использовать [QnAMaker](#). Сведения о [получении примеров](#) см. в файле сведений в образце репозитория.

## Создание ресурса QnA Maker в Azure Cognitive Services

Ресурс QnA Maker — это ресурс [Azure Cognitive Services](#), создаваемый с помощью страницы [создания Cognitive Services](#) Azure. Это предоставляет ключи безопасности и конечную точку, необходимые для доступа к QnA Maker KB в Azure.

1. Перейдите на страницу [создания Cognitive Services](#) Azure.
2. Введите значения для каждого поля, а затем нажмите кнопку " **проверить и создать** ".

The screenshot shows the Microsoft Azure 'Create' dialog for a 'QnA Maker' resource. The top navigation bar includes 'Microsoft Azure', a search bar, and user profile icons. The main form is titled 'Create' and has a 'QnA Maker' category selected. It contains several sections for configuration:

- Project details:** A note says to select a subscription for managing resources. Fields include 'Subscription' (Visual Studio Enterprise – MPN), 'Resource group' (dropdown with 'Create new' option), 'Name' (text input 'Enter a name'), and 'Pricing tier' (dropdown).
- Azure Search details - for data:** A note says the data is hosted in the Azure subscription. Fields include 'Azure Search location' (dropdown with '(US) West US') and 'Azure Search pricing tier' (dropdown).
- App Service details - for runtime:** A note says the runtime is hosted in the Azure subscription. Fields include 'App name' (text input 'App name') and 'Website location' (dropdown with '(US) West US').
- App insights details - for telemetry and chat logs:** A note says Application Insights will be provisioned. Fields include 'App insights' (button with 'Enable' and 'Disable' options) and 'App insights location' (dropdown with '(US) West US').

#### NOTE

При вводе **группы ресурсов** и **имени** Помните, что эти значения нельзя изменить позже. Также обратите внимание, что значение, присваиваемое **имени**, будет являться частью URL-**адреса конечной точки**.

3. Проверьте значения, чтобы убедиться, что они верны, а затем нажмите кнопку **создать**.

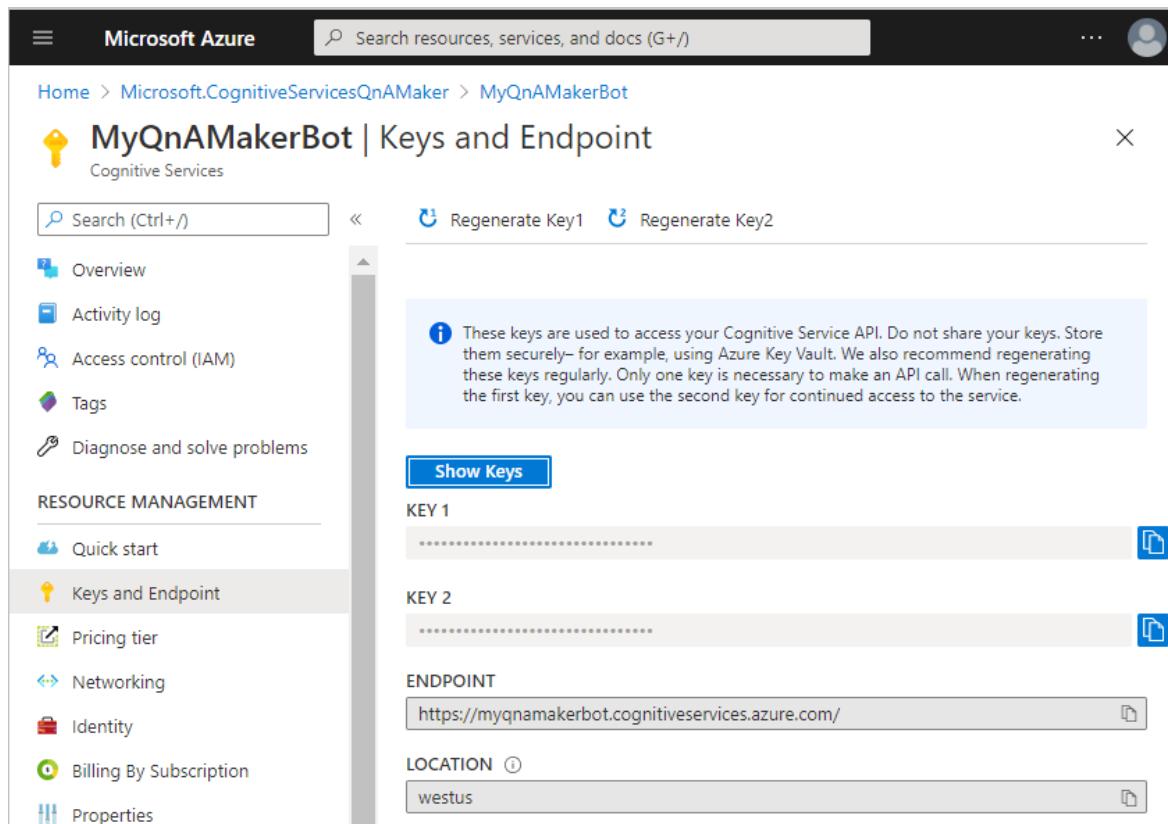
Ресурс QnA Maker включает сведения, которые будут использоваться программой Bot для доступа к базе знаний QnA Maker.

- **Ключи.** Они называются *ключами подписки* и автоматически создаются. Ключ подписки потребуется при ссылке на QnA Maker ресурс для любого действия, например при создании или обновлении QnA Maker KB, который будет подробно описан в этой статье. Ключи можно найти в колонке **ключи и конечная точка** в ресурсе QnA Maker.
- **Конечная точка.** Он создается автоматически с использованием QnA Maker имени ресурса, которое вы задаете при его создании. Он имеет следующий формат:  
`https://<qnamaker-resource-name>.cognitiveservices.azure.com/`. При ссылке на QnA Maker ресурс для любого действия, например при создании QnA Maker KB, которая будет подробно описана в этой статье. Ключ можно найти в колонке **ключи и конечная точка** в ресурсе QnA Maker.

#### TIP

Важно понимать разницу между этой QnA Maker конечной точкой создания ресурсов, на которую есть ссылка во всех командах BF CLI qnamaker, а также ключ конечной точки QnA Maker базы знаний, на который имеется ссылка в файлах конфигурации исходного кода, таких как appsettings.json в C# или `.env` JavaScript или `config.py` Python.

- **Расположение.** Это регион Azure, содержащий QnA Maker KB. Этот параметр выбирается при создании QnA Maker ресурса.



# Установка интерфейса командной строки для Bot Framework

Если вы уже установили интерфейс командной строки Bot Framework, можно сразу перейти к [созданию модели QnA Maker](#).

Для работы с Bot Framework CLI требуется [Node.js](#).

1. Убедитесь, что установлена последняя версия NPM:

```
npm i -g npm
```

2. С помощью Node.js установите последнюю версию Bot Framework CLI из командной строки.

```
npm i -g @microsoft/botframework-cli
```

Дополнительные сведения см. в статье [Bot Framework CLI Tool](#).

## Создание файла инициализации QnA Maker

Интерфейс командной строки Bot Framework предоставляет механизм для хранения всех часто используемых значений параметров в файле инициализации. После создания файла каждая `qnamaker` выполняемая команда проверит этот файл на наличие необходимых значений, если они не включены в командную строку. Если указать значение этого параметра, оно переопределит значение в файле `init`.

Этот файл инициализации создается с помощью `qnamaker:init` команды. При этом будет создан JSON-файл, содержащий данные, необходимые при выполнении многих команд интерфейса командной строки QnAMaker BF, включая *субскриптионкэй*, *kbld*, *ендпоинткэй* и *HostName*.

### NOTE

Так как вы еще не создали QnA Maker КБ, вы не сможете предоставить идентификатор базы знаний при запросе. Это приведет к тому, что в файле инициализации также будет отсутствовать значение `kbId` `hostname`. Эти значения будут добавлены автоматически при создании QnA Maker КБ, если при выполнении команды будет включен `--save` параметр.

Команда для создания файла инициализации `qnamaker`:

```
bf qnamaker:init
```

Чтобы создать QnA Maker файл инициализации CLI, выполните следующие действия.

1. В консоли введите `bf qnamaker:init`
2. Вам будет предложено ввести ключ подписки в ресурс QnA Maker Cognitive Services в Azure. Его можно найти в колонке *ключи и конечная точка*:

3. Далее вам будет предложено ввести идентификатор базы знаний (`kbId`). Так как вы еще не создали QnA Maker KB, введите **None**.
4. Значения собираются и записываются на экран для проверки. При правильном типе `yes` или просто нажмите клавишу **Ввод**.
5. Затем файл создается и сохраняется в `C:\Users\<username>\аппдата\локал@microsoft\botframework-cl\config.json`. Поскольку этот файл содержит конфиденциальные данные, он не сохраняется в том же каталоге, что и файлы проекта программы-робота, чтобы предотвратить его возврат в любое потенциально Небезопасное расположение при возврате исходного кода.

#### TIP

При вводе `bf qnamaker` команды CLI она автоматически ищет значения *субскриптионкэй*, *kbId*, *ендпоинткэй* и *HostName* в этом файле инициализации, если не включить их при вводе команды, после чего введенные значения будут переопределять значения из файла инициализации.

Дополнительные сведения об использовании этой команды см. [bf qnamaker:init](#) . в разделе BF CLI QnA Maker файле сведений.

#### IMPORTANT

Для команд, описанных в этой статье, предполагается, что у вас есть файл инициализации. Если у вас нет файла инициализации, необходимо включить эти значения при выполнении каждой команды.

## Создание модели QnA Maker

После создания отдельных файлов, QnA для программы-робота можно преобразовать их в одну *QnA Makerную модель* с помощью `qnamaker:convert` команды. Модель QnA Maker — это JSON-файл, используемый для создания QnA Maker KB.

Чтобы создать модель QnA Maker, выполните следующие действия.

```
bf qnamaker:convert -i <input-folder-name> -o <output-folder-name> --name <QnA-KB-Name> -r
```

Например, команда `bf qnamaker:convert -i dialogs -o output --name MyQnAMakerBot -r` рекурсивно выполняет поиск всех файлов. QnA в каталоге *диалоговых окон*\_ и всех подкаталогах и объединяет их в один файл с именем *converted.js* в *выходном* каталоге. Этот JSON-файл будет содержать все сведения, необходимые для создания QnA Maker КБ, включая имя *микнамакербот*, которое будет именем базы знаний QnA Maker, которая будет существовать в Azure.

Дополнительные сведения об использовании этой команды см `bf qnamaker:convert` . в разделе BF CLI QnA Maker файле сведений.

## Создание базы знаний QnA Maker

Созданный *ресурс QnA Maker* состоит из двух ключей подписки и конечной точки. Это значения, необходимые при создании QnA Maker КБ (QnA Maker KB). Можно иметь несколько QnA Maker KBs, связанных с одним QnA Makerным ресурсом, каждый QnA Maker KB будет иметь собственный идентификатор с именем `kbId` . Это значение будет возвращено как часть процесса создания. Этот идентификатор потребуется при ссылке на этот QnA Maker KB в будущем. Эта QnA Maker KB предоставляет программу-роботу все функции, предоставляемые QnA Maker.

Создание QnA Maker KB:

```
bf qnamaker:kb:create --in <QnA-Maker-model-JSON-file> --name <QnA-Maker-kb-name>
```

### NOTE

- Входной файл для этой команды — это файл, который создается путем выполнения команды, `qnamaker:convert` как описано на предыдущем шаге. Имя файла по умолчанию — *converted.js* .
- `name` Параметр является именем QnA Maker KB и является необязательным, если JSON-файл модели QnA имеет значение свойства Name, в противном случае он будет обязательным.

Дополнительные сведения об использовании этой команды см `bf qnamaker:kb:create` . в разделе BF CLI QnA Maker файле сведений.

## Тестирование базы знаний QnA Maker

Нет доступных команд BF CLI `qnamaker` для тестирования базы знаний, однако вы можете [протестировать базу знания с помощью пакетных вопросов и ожидаемых ответов](#).

Если вы планируете создавать скрипты для автоматизации этого процесса от конца к концу, это позволит включить тестирование.

## Публикация базы знаний QnA Maker

Вновь созданные QnA Maker KBs автоматически публикуются в *тестовой* конечной точке, где их можно протестировать перед выполнением в режиме реального времени. Общие сведения о тестировании базы знаний см. в [статье тестирование базы данных в QnA Maker](#).

После тестирования можно использовать, `qnamaker:kb:publish` чтобы опубликовать его в конечной точке *производства* .

Публикация базы знаний QnA Maker:

```
bf qnamaker:kb:publish
```

Чтобы опубликовать QnA Maker КБ, если файл [инициализации](#) отсутствует, выполните следующие действия.

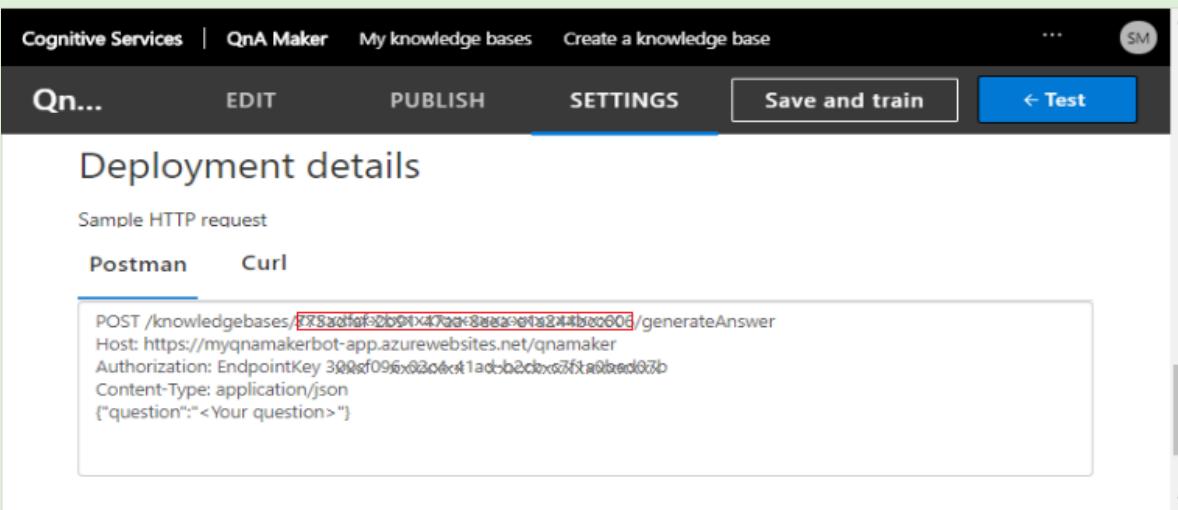
```
bf qnamaker:kb:publish --subscriptionKey <Subscription-Key> --kbId <knowledge-base-id>
```

#### TIP

Если имеется несколько QnA Maker КБ и вы хотите опубликовать не ту, которая указана в файле init, необходимо указать идентификатор базы знаний с помощью `--kbId` параметра:

```
bf qnamaker:kb:publish --kbId <knowledge-base-id>
```

Идентификатор базы знаний можно найти в [QnAMaker](#) в разделе *сведения о развертывании* страницы [Параметры](#):



## Создание базы знаний QnA Maker и ее публикация в рабочей среде с помощью команды Build

Полезно понимать, как работает процесс развертывания QnA Maker KB, и после завершения этой статьи вплоть до этого момента вы получите более полное представление о процессах, связанных с публикацией в тестовой или рабочей конечной точке, с помощью команд интерфейса командной строки Bot Framework.

Использование этих команд обеспечивает гибкость при адаптации сценариев к конкретным потребностям. Если такая гибкость не требуется, то существует еще одна команда CLI BF, которая объединяет большинство команд, обсуждаемых в этой статье, в одну команду, которую можно использовать для создания или обновления, а затем обучать и публиковать QnA Maker KB в рабочей конечной точке, а именно — `qnamaker:build` команду.

`build` Команда публикуется только в конечной точке производства. Его можно использовать отдельно от предыдущих команд, обсуждаемых в этой статье, так как они используются в основном в процессе разработки для публикации в конечной точке теста, где тестирование может быть выполнено перед публикацией в рабочей среде.

Команда QnAMaker Build объединяет все следующие действия в одну команду:

1. Создает одну модель QnA Maker для [каждого языкового стандарта](#), найденного с помощью

существующих QnAных файлов.

2. Создает новую QnA Maker КБ, если она не существует, в противном случае перезапишет существующую базу знаний.
3. Он обучает QnA Maker КБ затем публикует его в конечной точке производства.
4. При включении необязательного `dialog` параметра он выводит `.dialog` файлы определений, которые могут использоваться [распознавателем QnA Maker](#) при разработке с использованием [декларативного подхода](#). Это объясняется в разделе [файл диалогового окна](#).

## Использование команды Build

Команда QnAMaker Build с необходимыми параметрами:

```
bf qnamaker:build --in <input-file-or-folder> --subscriptionKey <Subscription-Key> --botName <bot-name>
```

### IMPORTANT

Эта команда перезапишет предыдущую модель QnA Maker, а также любое содержимое, которое может присутствовать в QnA Maker КБ, включая все содержимое, созданное непосредственно в [QnA Maker AI](#).

### Qnamaker: параметры сборки

- `in` : Каталог, включая подкаталоги, в котором будут искаться QnA-файлы.
- `out` : Каталог для сохранения выходных файлов. Сюда входят все файлы распознавателя, а также файл параметров и, при необходимости, файлы диалоговых окон. Если опустить `out` параметр, файлы не будут сохранены на диск, а на консоль будут записываться только ключи разработки и конечная точка.
- `log` : Логическое значение, определяющее, создается ли журнал во время этого процесса.
- `botName` — Имя программы-робота. Он будет использоваться для создания имени QnA Maker КБ. Это объясняется более подробно в разделе [QnA Maker Knowledge bases Created](#) ниже.
- `subscriptionKey` : Тот же ключ подписки, который находится в [файле инициализации](#).

Дополнительные сведения о дополнительных параметрах см. в разделе [BF qnamaker: Build](#) в файле `readme` интерфейса командной строки BF.

Кроме того, можно включить эти обязательные параметры в файл конфигурации и предоставить их с помощью `qnaConfig` параметра.

### Файл конфигурации qnamaker

Файл конфигурации `qnamaker` — это JSON-файл, который может содержать любой допустимый `qnamaker:build` параметр.

```
{  
  "in": "<location-of-qna-files>",  
  "out": "<location-to-save-output-files>",  
  "subscriptionKey": "<Enter-subscription-key-here>",  
  "botName": "<Enter-botName-here>"  
}
```

После создания необходимо ссылаться на него в `qnamaker:build` команде, например:

```
bf qnamaker:build --qnaConfig qnaConfig.json
```

### QnA и несколько вариантов языка

Каждый [файл QnA](#) может иметь несколько вариантов языка, по одному для каждого поддерживаемого

языка.

Шаблон для имени файла QnA при использовании вариантов языка выглядит следующим образом:

```
<file-name>.<locale>.qna
```

Пример:

```
example.en-us.qna  
example.fr-fr.qna  
example.de-de.qna  
etc.
```

В приведенном выше примере каждый из QnA файлов приведет к созданию одной QnA Maker KB для каждого из языков.

#### TIP

Если не удается определить язык на основе имени файла, будет использоваться значение, указанное в параметре CLI `--defaultCulture`. Если параметр CLI `--defaultCulture` отсутствует, по умолчанию используется язык `en-us`.

### QnA Maker созданных баз знаний

Чтобы включить сценарий, в котором несколько пользователей работают с моделями LUIS и должны иметь возможность работать с ними независимо друг от друга, в то время как все они привязаны к одному и тому же системе управления версиями, `qnamaker:build` команда создает QnA Maker базу знаний, используя сочетание значения, которое вы выдаете `botName` в качестве параметра, за которым следует имя пользователя, с которым идет вход, а затем языковой стандарт.

Имя QnA Maker KB:

```
<botName>(<user-name>).locale.qna
```

Например, если Ботнаме — *MyProject*, а имя пользователя — *юуританака*, то имена KBS будут выглядеть следующим образом:

```
MyProject(YuuriTanaka).en-us.qna  
MyProject(YuuriTanaka).fr-fr.qna  
MyProject(YuuriTanaka).de-de.qna
```

#### TIP

Включение имени пользователя в раздел базы знаний позволяет нескольким разработчикам работать независимо. Это значение создается автоматически с использованием имени пользователя, выполнившего вход, однако его можно переопределить с помощью `--suffix` параметра.

### Файл параметров, созданный с помощью команды сборки

Выходные данные `qnamaker:build` включают в себя один файл параметров, содержащий сопоставление со всеми QnA Maker базами знаний, включая идентификатор базы знаний и имя узла для каждого из них.

Файл параметров QnA Maker KB:

```
qnamaker.settings.<app-name>.<website-location>.json
```

Например, файл параметров для пользователя *юуританака* Region *westus* имеет имя:

## qnamaker.settings.YuuriTanaka.westus.json на

Пример файла параметров:

```
{  
  "qna": {  
    "RootDialog_en_us_qna": "<knowledge-base-ID-for-en-us-locale>",  
    "RootDialog_fr_fr_qna": "<knowledge-base-ID-for-fr-fr-locale>",  
    "hostname": "https://<app-name>.azurewebsites.net/qnamaker"  
  }  
}
```

### TIP

Часть имени файла *веб-сайта*, а также часть имени файла и имя узла в файле имеют значения, указанные при [создании QnA Maker ресурса в Azure](#), в разделе *сведения о службе приложений — для среды выполнения*.

#### App Service details - for runtime

When you create a QnAMaker resource, you host the runtime in your own Azure subscription. App Service is the compute engine that runs the QnA Maker queries for you.

App name *	App name
Website location *	(US) West US

## Файл диалогового окна

При использовании необязательного `--dialog` параметра файл диалогового окна будет создан для всех языковых вариантов каждого из QnA-файлов.

### IMPORTANT

`--schema` Параметр используется вместе с `--dialog` параметром. Включение `--schema` параметра гарантирует, что каждый созданный файл диалога будет иметь ссылку на файл корневой схемы проектов. Этот файл схемы содержит схемы всех компонентов, используемых программой Bot. Каждому потребителю декларативных файлов, включая [Composer] [Composer], требуется файл схемы. Если в проекте нет файла схемы, его можно создать с помощью `dialog:merge` команды. Перед выполнением команды необходимо выполнить эту команду `luis:build`. Дополнительные сведения см. в статье об [использовании декларативных ресурсов в адаптивных диалоговых окнах](#).

Диалоговое окно qnamaker: сборка и параметры схемы:

- **диалоговое окно.** Существует два допустимых значения параметра диалогового окна `multiLanguage` и `crosstrained`.
- **схема.** Это принимает относительный путь и имя файла, указывающего на файл схемы Bot.

Эти файлы будут записаны в каталог, указанный в `out` параметре. Пример:

```
./rootDialog/RootDialog.qna.dialog <-- MultiLanguageRecognizer configured to use all of the languages  
.rootDialog/RootDialog.en-us.qna.dialog <-- QnARecognizer for en-us locale  
.rootDialog/RootDialog.fr-fr.qna.dialog <-- QnARecognizer for fr-fr locale
```

Ниже приведен пример файла *мултилангуажереконизер*:

```
{  
    "$schema": "app.schema",  
    "$kind": "Microsoft.MultiLanguageRecognizer",  
    "id": "QnA_RootDialog",  
    "recognizers": {  
        "en-us": "RootDialog.en-us.qna",  
        "fr-fr": "RootDialog.fr-fr.qna",  
        "": "RootDialog.en-us.qna"  
    }  
}
```

Эти файлы будут использоваться, если вы используете декларативный подход к разработке программы Bot, и вам нужно будет добавить ссылку на этот распознаватель в файл адаптивных диалогов `.dialog`. В следующем примере `"recognizer": "RootDialog.qna"` выполняется поиск распознавателя, определенного в файле **рутдиалог. QnA. Dialog**:

```
{  
    "$schema": "app.schema",           ← The schema file created when the schema option is used  
    "$kind": "Microsoft.AdaptiveDialog", ← This is a dialog  
    "recognizer": "RootDialog.qna",     ← This is referencing the file named RootDialog.qna.dialog  
    "triggers": [  
        {$kind: "OnQnaMatch"}          ← The onQnAMatch Trigger is required to react to the QnA Match event  
    ]  
}
```

Дополнительные сведения см. в разделе [Использование декларативных ресурсов в адаптивных диалоговых окнах](#).

## Дополнительные сведения

- [Создание настраиваемых списков синонимов для базы знаний QnA Maker](#)

# Создание настраиваемых списков синонимов для базы знаний QnA Maker

27.03.2021 • 8 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Интерфейс командной строки Bot Framework позволяет автоматизировать управление базами знаний QnA Maker. Она позволяет создавать, обновлять и удалять QnA Maker базу знаний (КБ) из командной строки или скрипта. Он также позволяет создать список синонимов, которые применяются к базе знаний. В этой статье объясняется, как создать синонимы для QnA Maker КБ с помощью команды CLI `alters`.

## Предварительные требования

- Существующий QnA Maker КБ. Если у вас еще нет QnA Maker КБ, можно создать ее, выполнив действия, описанные в статье [развертывание базы знаний QnA Maker с помощью команд CLI Qnamaker Framework](#).
- Файл инициализации QnA Maker. Если у вас еще нет файла инициализации QnA Maker, его можно создать, выполнив действия, описанные в разделе [Создание файла инициализации QnA Maker](#) в базе знаний о [развертывании QnA Maker, используя команды CLI qnamaker Framework](#).

## Введение в изменения в QnA Maker

Команда "изменения" позволяет импортировать настраиваемые списки синонимов в QnA Maker КБ. Изменения представляют собой список слов, имеющих одинаковое значение. Например, синонимом для слова «подарок» может быть слово «Present».

Изменения также могут быть очень полезны для сокращений. Например, "GDPR" является широко используемым термином, но некоторые люди могут назвать его "СРЕДНИМ", которое является сокращенным. Компании часто имеют собственный уникальный список сокращений, которые ссылаются на различные функции или компоненты, предлагаемые их продуктами.

Хотя QnA Maker уже имеет собственный внутренний список стандартных синонимов на нескольких языках, многие компании по-прежнему могут воспользоваться преимуществами дополнительных синонимов.

Изменения также могут помочь улучшить качество базы знаний, одновременно сокращая количество пар вопросов и ответов, а также время, необходимое для обучения.

### TIP

Невозможно создать настраиваемые списки синонимов с помощью портала QnA Maker, однако он доступен с помощью интерфейса командной строки Bot.

## Установка интерфейса командной строки для пакета SDK для Bot Framework

Если вы уже установили интерфейс командной строки Bot, вы можете сразу перейти к [использованию команд интерфейса командной строки qnamaker, чтобы создать список синонимов для базы знаний QnA Maker](#).

Для работы с Bot Framework CLI требуется [Node.js](#).

1. Убедитесь, что установлена последняя версия NPM:

```
npm i -g npm
```

2. С помощью Node.js установите последнюю версию Bot Framework CLI из командной строки.

```
npm i -g @microsoft/botframework-cli
```

Дополнительные сведения см. в статье [Bot Framework CLI Tool](#).

## Использование команд интерфейса командной строки qnamaker для создания списка синонимов для базы знаний QnA Maker

При [создании модели QnA Maker](#) Создаются два JSON-файла: модель *QnAMaker* с именем *converted.json* и файл *изменений* с именем *alterations\_converted.json*. Хотя модель QnAMaker содержит данные из всех файлов. QnA в проекте, все вместе образуют один файл, файл изменений содержит только пустой список изменений, как показано ниже:

```
{
  "wordAlterations": []
}
```

Вам не нужно создавать модель QnA Maker или базу знаний с помощью команд интерфейса командной строки, чтобы использовать команды изменения для создания списка синонимов, но при этом создается файл изменений Word.

Файл изменений — это JSON-файл, содержащий массив *wordAlterations*, состоящий из массива *изменений*, который представляет собой список синонимов, например:

```
{
  "wordAlterations": [
    {
      "alterations": [
        "qnamaker",
        "qna maker"
      ]
    },
    {
      "alterations": [
        "botframework",
        "bot framework"
      ]
    },
    {
      "alterations": [
        "bot framework command line interface",
        "bot framework cli",
        "bf cli"
      ]
    }
  ]
}
```

После создания файла изменений его можно передать `qnamaker:alterations:replace` команде в качестве `input` свойства для замены пустого списка изменений, созданного по умолчанию при создании QnA

Maker KB. Вы будете использовать одну и ту же команду в любой момент, когда потребуется обновить существующий список.

```
bf qnamaker:alterations:replace -i <input-file-name>
```

Если у вас нет [файла инициализации](#), необходимо включить ключ подписки:

```
bf qnamaker:alterations:replace -i <input-file-name> --subscriptionKey <Subscription-Key>
```

#### IMPORTANT

Вы не можете постепенно добавлять или удалять элементы из списка изменений в Azure. При выполнении команды ALTER Replace список изменений в Azure удаляется и заменяется переданным файлом.

Дополнительные сведения об использовании этой команды см [bf qnamaker:alterations:replace](#) . в разделе BF CLI QnA Maker файле сведений.

## Скачивание списка изменений в базе знаний QnA Maker

Если необходимо увидеть, какие синонимы находятся в QnA Maker KB, можно использовать [qnamaker:alterations:list](#) команду.

```
bf qnamaker:alterations:list
```

Если у вас нет [файла инициализации](#), необходимо включить ключ подписки:

```
`bf qnamaker:alterations:list --subscriptionKey <Subscription-Key>`
```

Дополнительные сведения об использовании этой команды см [bf qnamaker:alterations:list](#) . в разделе BF CLI QnA Maker файле сведений.

## Обновление списка изменений в базе знаний QnA Maker

Хотя нет команды для непосредственного обновления существующего списка изменений в QnA Maker, можно [использовать команду изменения списка](#), чтобы скачать список изменений, внести необходимые изменения, а затем использовать этот новый список, чтобы заменить список изменений в Azure.

- Получение текущего списка изменений с помощью команды [bf qnamaker:alterations:list](#)

#### TIP

Результаты можно отправить непосредственно в файл с помощью команды трубопроводов, например [>](#) команды DOS. В следующем примере создается файл с именем `alterations_converted.json` в текущем каталоге:

```
bf qnamaker:alterations:list >alterations_converted.json
```

- Внесите необходимые обновления в файл JSON и сохраните эти изменения.

- Замените список изменений, который находится в QnA Maker KB, с помощью команды:

```
bf qnamaker:alterations:replace -i <input-file-name> . Если файл JSON списка изменений сохранен
```

как `alterations_converted.js` в текущем каталоге, команда будет иметь следующее:

```
bf qnamaker:alterations:replace -i alterations_converted.json
```

# Использование нескольких моделей LUIS и QnA

27.03.2021 • 32 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Если бот использует несколько моделей LUIS и базы знаний QnA Maker, вы можете применить средство Dispatch, чтобы определить модель LUIS или базу знаний QnA Maker, которые лучше всего соответствуют вводимым пользователем данным. Для этого средство Dispatch создает одно приложение LUIS, чтобы отправить эти данные в соответствующую модель. Дополнительные сведения об отправке, включая команды интерфейса командной строки, см. в [файле сведений об отправке](#).

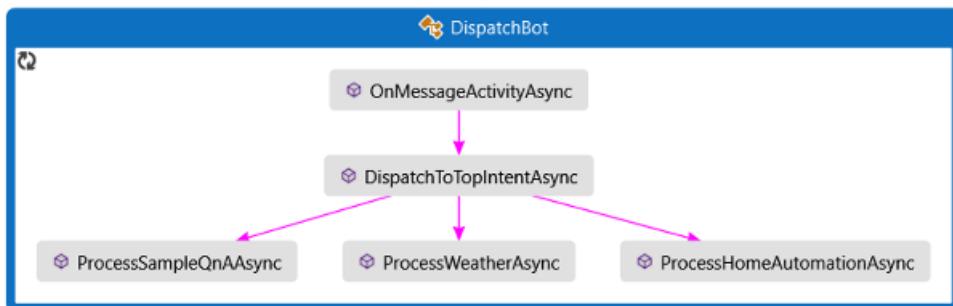
## Предварительные условия

- Учетная запись [luis.ai](#) для публикации приложений LUIS.
- Учетная запись [QnA Maker](#) для публикации базы знаний QnA.
- Копия NLP с примером диспетчеризации в [C#](#), [JavaScript](#) или [Python](#).
- [Базовые знания о ботах, LUIS и QnA Maker](#).
- [Средство отправки](#) из командной строки

## Об этом примере

Этот пример основан на предопределенном наборе LUIS и QnA Maker приложений.

- [C#](#)
- [JavaScript](#)
- [Python](#)



`OnMessageActivityAsync` вызывается для каждого полученного блока данных, введенных пользователем. Этот модуль обнаруживает намерения пользователя с наивысшими оценками и передает результат в `DispatchToTopIntentAsync`. Диспачтотопинтентасинк, в свою очередь, вызывает соответствующий обработчик приложения.

- `ProcessSampleQnAAsync` — для вопросов и ответов о боте.
- `ProcessWeatherAsync` — для запросов о погоде.
- `ProcessHomeAutomationAsync` — для команд домашнего освещения.

Обработчик вызывает службу LUIS или QnA Maker и возвращает полученный результат пользователю.

## Создание приложений LUIS и базы знаний QnA

Перед созданием модели отправки вам нужно создать и опубликовать приложения LUIS и базы знаний

QnA. В этой статье мы опубликуем следующие модели, которые присутствуют в примере *NLP with Dispatch* в папке `\CognitiveModels`:

Имя	Описание
HomeAutomation	Приложение LUIS распознает намерение обращения к службе автоматизации и данные о сущностях.
Weather	Приложение LUIS распознает намерения, связанные с погодой и данными о расположении.
QnA Maker	База данных QnA Maker предоставляет ответы на несколько простых вопросов о боте.

### Создание приложений LUIS

1. Создайте приложение LUIS из файла JSON службы "Домашняя страница" в каталоге "автоматизированные модели" примера.
  - a. Обучить и публикуйте приложение в рабочей среде.
  - b. Запишите идентификатор приложения, отображаемое имя, ключ создания и расположение.
2. Повторите эти действия для файла "Логода" JSON.

Дополнительные сведения см. в статьях **Создание приложения Luis на портале Luis** и **Получение значений для подключения к приложению Luis** в разделе [Добавление понимания естественного языка в программу Bot](#) и документацию Luis о том, как [обучить и публиковать](#) приложение в рабочей среде.

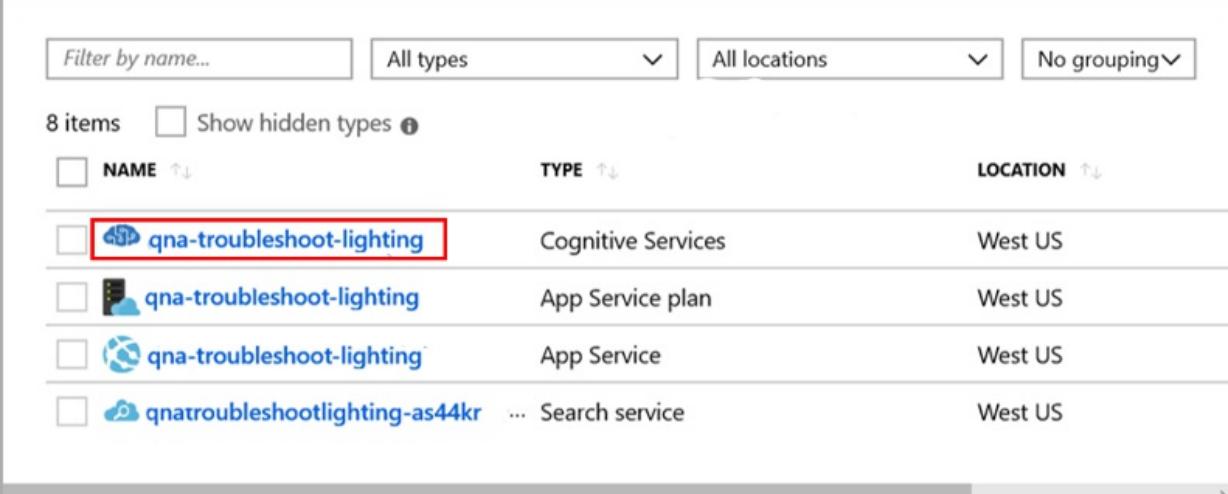
### Создание базы знаний QnA Maker

Первым шагом при настройке базы знаний службы QnA Maker является настройка службы QnA Maker в Azure. Для этого выполните пошаговые инструкции, чтобы найти статью [Настройка qnamaker Service](#).

Создав в Azure службу QnA Maker, запишите *ключ 1 Cognitive Services*, который предоставлен для вашей службы QnA Maker. Он будет использоваться как `<azure-qna-service-key>` при добавлении приложения QnA Maker в приложение диспетчеризации.

См. дополнительные сведения о [двух типах ключей](#), используемых с QnA Maker.

Чтобы получить этот ключ, сделайте следующее:



The screenshot shows the Azure portal interface with a search bar at the top. Below it, there are four dropdown menus: 'Filter by name...', 'All types', 'All locations', and 'No grouping'. A message '8 items' and a checkbox 'Show hidden types' are displayed. The main area lists four services:

NAME	TYPE	LOCATION
qna-troubleshoot-lighting	Cognitive Services	West US
qna-troubleshoot-lighting	App Service plan	West US
qna-troubleshoot-lighting	App Service	West US
qnatroubleshootlighting-as44kr	... Search service	West US

1. На портале Azure выберите службу QnA Maker из Cognitive Services.

The screenshot shows the Azure QnA Maker 'Quick start' page. On the left, there's a sidebar with links like 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', 'Diagnose and solve problems', 'RESOURCE MANAGEMENT' (which is expanded), and 'Keys'. The 'Keys' link is highlighted with a red box. The main content area says 'Congratulations! Your keys are ready.' and provides instructions for grabbing keys and making an API call. It also includes links to 'API reference', 'Realtime API usage', and 'API metrics alert'.

2. Выберите ключи, которые расположены в разделе *управления ресурсами* в меню слева.

The screenshot shows the 'Manage keys' page. It has two buttons at the top: 'Regenerate Key1' and 'Regenerate Key2'. Below that is a 'NAME' input field containing 'qna-troubleshoot-lighting'. Under 'KEY 1', there is a text input field with a redacted value and a 'Copy to clipboard' button to its right. Below it is another text input field for 'KEY 2' with a redacted value and a 'Copy to clipboard' button to its right.

3. Скопируйте значение *ключа 1* в буфер обмена и сохраните его на локальном компьютере.

Впоследствии это значение будет использоваться для значения ключа (-k) <azure-qna-service-key1> при добавлении приложения QnA Maker в приложение диспетчеризации.

4. Теперь войдите на [веб-портал QnA Maker](#).

5. На шаге 2 выберите следующие значения:

- Учетную запись Azure AD.
- Имя подписки Azure.
- Имя, с которым вы создали службу QnA Maker. (Если нужная служба Azure QnA отсутствует в этом раскрывающемся списке, попробуйте обновить страницу.)

**STEP 2****Connect your QnA service to your KB.**

After you create an Azure QnA service, [refresh this page](#) and then select your Azure service using the options below.

\* Microsoft Azure Directory ID

\* Azure subscription name

\* Azure QnA service

6. На шаге 3 укажите имя для новой базы знаний QnA Maker. В этом примере используйте имя sample-qna.

**STEP 3****Name your KB.**

The knowledge base name is for your reference and you can change it at anytime.

\* Name

7. На шаге 4 нажмите кнопку *+ Add File* (+ Добавить файл), перейдите к папке CognitiveModel с примером кода и выберите файл QnAMaker.tsv. Здесь есть дополнительный параметр, позволяющий добавить в базу знаний личность *Chit-chat* (Беседа), но в нашем примере он не используется.

**STEP 4****Populate your KB.**

Extract question-and-answer pairs from an online FAQ, product manuals, or other files. Supported formats are .tsv, .pdf, .doc, .docx, .xlsx, containing questions and answers in sequence. [Learn more about knowledge base sources](#). Skip this step to add questions and answers manually after creation. The number of sources and file size you can add depends on the QnA service SKU you choose. [Learn more about QnA Maker SKUs](#).

**URL**[+ Add URL](#)**File name**[+ Add file](#)

8. На шаге 5 выберите *Create your knowledge base* (Создать базу знаний).

9. Когда база знаний будет создана из переданного файла, выберите действие *Save and train* (Сохранить и обучить), а когда оно будет выполнено, перейдите на вкладку *PUBLISH* (Публикация) и опубликуйте приложение.
10. После публикации приложения QnA Maker откройте вкладку *SETTINGS* (Параметры) и прокрутите эту страницу вниз до раздела *Deployment Details* (Сведения о развертывании). Запишите следующие значения из примера HTTP-запроса *Postman*.

```
POST /knowledge_bases/<knowledge-base-id>/generateAnswer
Host: <your-hostname> // NOTE - this is a URL.
Authorization: EndpointKey <qna-maker-resource-key>
```

Полная строка URL-адреса для имени узла будет выглядеть так:

[https://<имя\\_узла>.azure.net/qnamaker](https://<имя_узла>.azure.net/qnamaker). Эти значения будут потом использоваться в файле

`appsettings.json` ИЛИ `.env`.

## Приложению для отправки требуется доступ на чтение к имеющимся приложениям

Средству отправки требуется доступ для чтения имеющихся приложений LUIS и QnA Maker для создания родительского приложения LUIS, которое отправляется приложениям LUIS и QnA Maker. Вместе с этим доступом предоставляются идентификаторы приложений и ключи разработки.

### Ключи для создания служб

**Ключ разработки** используется только для создания и редактирования моделей. Вам нужен идентификатор и ключ для каждого приложения LUIS и приложения QnA Maker.

- для LUIS
  - **Идентификатор приложения** находится на [портале Luis](#) для каждого приложения, управляет параметрами > > параметры приложения.
  - **Ключ разработки** находится на портале Luis, правом верхнем углу, выборе собственного пользователя, а затем параметры.
- для QnA Maker
  - **Идентификатор приложения** находится на [QnA Maker портале](#) на странице параметры после публикации приложения. Это идентификатор, расположенный в первой части команды POST после элемента knowledgebase. Например, `POST /knowledgebases/<APP-ID>/generateAnswer`.
  - **Ключ разработки** находится в портал Azure для ресурса QnA Maker в разделе **ключи**. Вам потребуется только один ключ.

Ключ разработки не используется для получения оценки прогнозирования или достоверности из опубликованного приложения. Для этого действия нужны ключи конечных точек. Мы найдем и применим **ключи конечной точки** далее в этом руководстве.

См. дополнительные сведения о [двух типах ключей](#), используемых с QnA Maker.

## Создание модели отправки

Интерфейс CLI для средства Dispatch создает модель для отправки в соответствующее приложение LUIS или QnA Maker.

1. Откройте командную строку или окно терминала и измените каталоги на каталог `CognitiveModels`.
2. Убедитесь, что у вас установлена текущая версия npm и средства Dispatch.

```
npm i -g npm
npm i -g botdispatch
```

3. Используйте `dispatch init` для инициализации файла `.dispatch` для модели отправки. Присвойте файлу понятное имя.

```
dispatch init -n <filename-to-create> --luisAuthoringKey "<your-luis-authoring-key>" --  
luisAuthoringRegion <your-region>
```

4. Используйте `dispatch add` для добавления приложений LUIS и баз знаний QnA Maker в файл `.dispatch`.

```
dispatch add -t luis -i "<app-id-for-weather-app>" -n "<name-of-weather-app>" -v <app-version-number>  
-k "<your-luis-authoring-key>" --intentName l_Weather  
dispatch add -t luis -i "<app-id-for-home-automation-app>" -n "<name-of-home-automation-app>" -v  
<app-version-number> -k "<your-luis-authoring-key>" --intentName l_HomeAutomation  
dispatch add -t qna -i "<knowledge-base-id>" -n "<knowledge-base-name>" -k "<azure-qna-service-key1>"  
--intentName q_sample-qna
```

5. Используйте `dispatch create` для создания модели отправки из файла `.dispatch`.

```
dispatch create
```

6. Опубликуйте созданное приложение LUIS для отправки.

## Использование приложения LUIS для отправки

Созданное приложение LUIS определяет намерения для каждого дочернего приложения и каждой базы знаний, а также намерение *None* при несоответствии речевого фрагмента.

- `l_HomeAutomation`
- `l_Weather`
- `None`
- `q_sample-qna`

Эти службы нужно опубликовать с соответствующими именами, чтобы обеспечить правильную работу бота. Бот получает доступ к опубликованным службам на основе этих сведений.

### Ключи конечной точки службы

Боту требуются конечные точки прогнозирования запросов для трех приложений LUIS (отправка, погода и домашняя автоматика) и единой базы знаний QnA Maker. Найдите ключи конечных точек на порталах LUIS и QnA Maker:

- Чтобы найти ключи, связанные с каждым приложением, перейдите на портал LUIS и для каждого приложения LUIS в разделе "Управление" выберите **Keys and Endpoint settings** (Параметры ключей и конечной точки). Если вы в точности выполняете инструкции, приведенные в этом учебнике, ключом конечной точки является ключ `<your-luis-authoring-key>`. Ключ разработки допускает 1000 попаданий конечной точки, а затем срок его действия истекает.
- На портале QnA Maker для базы знаний перейдите в раздел "Управление настройками" и используйте значение ключа, отображаемое в параметрах Postman для заголовка **Авторизация** без текста `EndpointKey`.

Эти значения используются в файле конфигурации образца: `appsettings.js` (C#), `.env` (JavaScript) или `config.py` (Python).

- C#
- JavaScript
- Python

### Установка пакетов

Перед первым запуском приложения убедитесь, что установлены следующие пакеты NuGet:

- Microsoft.Bot.Builder
- Microsoft.Bot.Builder.AI.Luis
- Microsoft.Bot.Builder.AI.QnA

### Обновление файла appsettings.json вручную

После создания всех приложений службы вам нужно добавить сведения о них в файл appsettings.json.

Исходный пример кода [C#](#) содержит пустой файл appsettings.json:

#### appsettings.json

```
"MicrosoftAppId": "",  
"MicrosoftAppPassword": "",  
  
"QnAKnowledgebaseId": "",  
"QnAEndpointKey": "",  
"QnAEndpointHostName": "",  
  
"LuisAppId": "",  
"LuisAPIKey": "",  
"LuisAPIHostName": "",
```

Для каждой сущности ниже добавьте значения, которые вы записали ранее при выполнении этих инструкций:

#### appsettings.json

```
"MicrosoftAppId": "",  
"MicrosoftAppPassword": "",  
  
"QnAKnowledgebaseId": "<knowledge-base-id>",  
"QnAEndpointKey": "<qna-maker-resource-key>",  
"QnAEndpointHostName": "<your-hostname>",  
  
"LuisAppId": "<app-id-for-dispatch-app>",  
"LuisAPIKey": "<your-luis-endpoint-key>",  
"LuisAPIHostName": "<your-dispatch-app-region>";
```

Когда все изменения будут внесены, сохраните файл.

### Подключение к службам из бота

Чтобы подключиться к службам диспетчеризации, LUIS и QnA Maker, бот получает сведения из файла параметров.

- [C#](#)
- [JavaScript](#)
- [Python](#)

В файле BotServices.cs сведения из файла конфигурации *appsettings.json* используются для подключения бота отправки к службам `Dispatch` и `SampleQnA`. Конструкторы используют предоставленные значения для подключения к этим службам.

#### BotServices.cs

```

public class BotServices : IBotServices
{
    public BotServices(IConfiguration configuration)
    {
        // Read the setting for cognitive services (LUIS, QnA) from the appsettings.json
        // If includeApiResults is set to true, the full response from the LUIS api (LuisResult)
        // will be made available in the properties collection of the RecognizerResult

        var luisApplication = new LuisApplication(
            configuration["LuisAppId"],
            configuration["LuisAPIKey"],
            $"https://{{configuration["LuisAPIHostName"]}}.api.cognitive.microsoft.com");

        // Set the recognizer options depending on which endpoint version you want to use.
        // More details can be found in https://docs.microsoft.com/en-gb/azure/cognitive-services/luis/luis-
migration-api-v3
        var recognizerOptions = new LuisRecognizerOptionsV2(luisApplication)
        {
            IncludeAPIResults = true,
            PredictionOptions = new LuisPredictionOptions()
            {
                IncludeAllIntents = true,
                IncludeInstanceData = true
            }
        };

        Dispatch = new LuisRecognizer(recognizerOptions);

        SampleQnA = new QnAMaker(new QnAMakerEndpoint
        {
            KnowledgeBaseId = configuration["QnAKnowledgebaseId"],
            EndpointKey = configuration["QnAEndpointKey"],
            Host = configuration["QnAEndpointHostName"]
        });
    }

    public LuisRecognizer Dispatch { get; private set; }
    public QnAMaker SampleQnA { get; private set; }
}

```

#### NOTE

По умолчанию параметру `includeApiResults` присваивается значение `false`, означающее, что распознаватель будет возвращать только основные сведения о сущностях и намерениях. Если требуется полный ответ от LUIS (например, `ConnectedServiceResult`, см. далее в этом руководстве), установите для этого параметра значение `true`. После этого будет добавлен полный ответ от службы LUIS в коллекцию свойств для `RecognizerResult`.

#### Вызов служб из бота

Логика бота проверяет каждый блок вводимых пользователем данных с использованием объединенной модели отправки, находит первые возвращаемые намерения и использует эти сведения, чтобы вызвать соответствующую службу для входных данных.

- [C#](#)
- [JavaScript](#)
- [Python](#)

В файле `DispatchBot.cs` при вызове метода `OnMessageActivityAsync` мы проверяем входящее сообщение пользователя с использованием модели отправки. Затем мы передаем `topIntent` и `recognizerResult` модели отправки в соответствующий метод, чтобы вызвать службу и получить результат.

`bots\DispatchBot.cs`

```

protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    // First, we use the dispatch model to determine which cognitive service (LUIS or QnA) to use.
    var recognizerResult = await _botServices.Dispatch.RecognizeAsync(turnContext, cancellationToken);

    // Top intent tell us which cognitive service to use.
    var topIntent = recognizerResult.GetTopScoringIntent();

    // Next, we call the dispatcher with the top intent.
    await DispatchToTopIntentAsync(turnContext, topIntent.intent, recognizerResult, cancellationToken);
}

```

## Использование результатов распознавания

- [C#](#)
- [JavaScript](#)
- [Python](#)

Когда модель возвращает результат, он позволяет понять, какие службы лучше всего подходят для обработки этого высказывания. Код нашего бота направляет запрос в соответствующую службу и обрабатывает полученный от нее ответ. Код использует *намерение*, которое возвращает средство Dispatch, чтобы выполнить перенаправление в соответствующую модель LUIS или службу QnA.

`bots\DispatchBot.cs`

```

private async Task DispatchToTopIntentAsync(ITurnContext<IMessageActivity> turnContext, string intent,
RecognizerResult recognizerResult, CancellationToken cancellationToken)
{
    switch (intent)
    {
        case "l_HomeAutomation":
            await ProcessHomeAutomationAsync(turnContext, recognizerResult.Properties["luisResult"] as LuisResult, cancellationToken);
            break;
        case "l_Weather":
            await ProcessWeatherAsync(turnContext, recognizerResult.Properties["luisResult"] as LuisResult, cancellationToken);
            break;
        case "q_sample-qna":
            await ProcessSampleQnAAsync(turnContext, cancellationToken);
            break;
        default:
            _logger.LogInformation($"Dispatch unrecognized intent: {intent}.");
            await turnContext.SendActivityAsync(MessageFactory.Text($"Dispatch unrecognized intent: {intent}."),
                cancellationToken);
            break;
    }
}

```

Вызываемые методы `ProcessHomeAutomationAsync` или `ProcessWeatherAsync` передают результаты из модели отправки с использованием `luisResult.ConnectedServiceResult`. Затем указанный метод предоставляет отзыв пользователя, отображая первое намерение модели отправки, а также ранжированный список всех намерений и сущностей, которые были обнаружены.

Вызываемый метод `q_sample-qna` использует введенные пользователем данные, содержащиеся в `turnContext`, для создания ответа из базы знаний и отображения результата пользователю.

#### NOTE

Если используется рабочее приложение, выбранные методы LUIS подключаются к соответствующей службе, передают введенные пользователем данные и обрабатывают возвращаемые LUIS данные о намерениях и сущностях.

## Тестирование бота

1. В среде разработки откройте файл с пустым кодом. Запишите адрес `/local/host` из адресной строки окна браузера, открытого приложением: `https://localhost:<Port_Number>`.
2. Откройте эмулятор Bot Framework, а затем выберите `Create a new bot configuration`. `.bot` Файл позволяет использовать *инспектор* в эмуляторе для просмотра JSON, возвращаемого из Luis и QnA Maker.
3. В диалоговом окне **New bot configuration** (Новая конфигурация бота) введите имя своего бота и URL-адрес конечной точки, например `http://localhost:3978/api/messages`. Сохраните файл в корневой папке своего проекта примера кода бота.
4. Откройте файл бота и добавьте разделы для приложений LUIS и QnA Maker. Используйте [этот пример файла](#) в качестве шаблона для настроек. Сохраните изменения.
5. Выберите имя бота в списке **My Bots** (Мои боты), чтобы получить доступ к запущенному боту. Ниже для справки приведены некоторые вопросы и команды, поддерживаемые в используемых для бота службах.
  - QnA Maker
    - `hi` , `good morning`
    - `what are you` , `what do you do`
  - LUIS (автоматизация дома)
    - `turn on bedroom light`
    - `turn off bedroom light`
    - `make some coffee`
  - LUIS (погода)
    - `whats the weather in redmond washington`
    - `what's the forecast for london`
    - `show me the forecast for nebraska`

## Отправка пользовательских речевых фрагментов в QnA Maker

1. В эмуляторе Bot введите текст `hi` и отправьте utterance. Бот отправляет этот запрос в приложение LUIS для отправки и возвращает ответ, указывающий дочернее приложение, которое должно получить этот речевой фрагмент для дальнейшей обработки.
2. Выбрав `Luis Trace` строку в журнале, можно увидеть ответ Luis в эмуляторе Bot. Результат LUIS, полученный от приложения LUIS для отправки отображается в инспекторе.

```
{
  "luisResponse": {
    "entities": [],
    "intents": [
      {
        "intent": "q_sample-qna",
        "score": 0.9489713
      },
      {
        "intent": "l_HomeAutomation",
        "score": 0.0612499453
      },
      {
        "intent": "None",
        "score": 0.008567564
      },
      {
        "intent": "l_Weather",
        "score": 0.0025761195
      }
    ],
    "query": "Hi",
    "topScoringIntent": {
      "intent": "q_sample-qna",
      "score": 0.9489713
    }
  }
}
```

Так как речевой фрагмент `hi` является частью намерения `q_sample-qna` приложения LUIS для отправки и выбрано в качестве `topScoringIntent`, бот выполнит второй запрос с тем же речевым фрагментом, на этот раз к приложению QnA Maker.

3. Выберите `QnAMaker Trace` строку в журнале эмулятора Bot. Результат QnA Maker отображается в инспекторе.

```
{
  "questions": [
    "hi",
    "greetings",
    "good morning",
    "good evening"
  ],
  "answer": "Hello!",
  "score": 1,
  "id": 96,
  "source": "QnAMaker.tsv",
  "metadata": [],
  "context": {
    "isContextOnly": false,
    "prompts": []
  }
}
```

## Удаление неверного основного намерения из средства Dispatch

Когда бот будет запущен, вы можете повысить его речевые фрагменты, удаляя похожие или перекрывающиеся высказывания в приложениях для отправки.

Чтобы протестировать и оценить модель подготовки к отправке, можно использовать средство командной строки `Dispatch`.

## Обновление или создание модели LUIS

этот пример основан на предварительно настроенной модели LUIS. Дополнительные сведения, которые помогут вам обновить эту модель или создать новую модель LUIS, можно найти в статье [итеративное проектирование приложений для Luis](#).

После обновления базовых моделей (QnA или LUIS) запустите `dispatch refresh`, чтобы обновить приложение LUIS для отправки. По сути `dispatch refresh` отличается от `dispatch create` только тем, что не создает новый идентификатор приложения LUIS.

Обратите внимание, что добавленные непосредственно в LUIS речевые фрагменты не будут сохранены при выполнении `dispatch refresh`. Чтобы сохранить эти дополнительные речевые фрагменты в приложении отправки, добавьте их в текстовый файл (по одному фрагменту в строке), а затем добавьте этот файл в приложение с помощью следующей команды:

```
dispatch add -t file -f <file path> --intentName <target intent name, ie l_General>
```

Когда вы добавите файл с дополнительными речевыми фрагментами в приложение отправки, фрагменты будут сохраняться нетронутыми после каждого обновления.

## Удаление ресурсов

С помощью этого примера создается ряд приложений и ресурсов, которые можно удалить с помощью описанных ниже действий. Следите при этом за тем, чтобы не удалить ресурсы, от которых зависят [другие приложения или службы](#).

Чтобы удалить ресурсы LUIS, сделайте следующее:

1. Войдите на портал [luis.ai](#).
2. Перейдите к странице *My Apps* (Мои приложения).
3. Выберите приложения, созданные этим примером.
  - Home Automation
  - Weather
  - NLP-With-Dispatch-BotDispatch
4. Щелкните *Delete* (Удалить), а затем *OK* для подтверждения.

Чтобы удалить ресурсы QnA Maker, сделайте следующее:

1. Войдите на портал [qnamaker.ai](#).
2. Перейдите к странице *Мои базы знаний*.
3. Нажмите кнопку "Удалить" для базы знаний `Sample QnA`, затем щелкните *Удалить* для подтверждения.

## Рекомендации

Чтобы улучшить описанные здесь службы, изучите рекомендации по [LUIS](#) и [QnA Maker](#).

# Создание проекта бота с использованием адаптивных диалогов

27.10.2020 • 10 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Если вы хотите создать Адаптивное диалоговое окно с нуля или использовать диалоговое окно, созданное на платформе Bot Framework, вам потребуется определенное формирование шаблонов на месте, которое не требуется для других типов проектов Bot.

В этой статье показано, как создать проект BOT C#, начиная с **пустого шаблона Bot**. Позднее можно добавить адаптивные диалоговые окна, в том числе декларативные диалоговые окна, созданные Composer.

## Предварительные требования

- Visual Studio 2019 или более поздней версии или Visual Studio Code.
- Концептуальное представление о [адаптивных диалоговых окнах](#).
- В примере кода эта статья ссылается на пример адаптивной [многофакторной подсказки в C#](#).

## Создание проекта

Сведения о том, как создать проект BOT, см. в [кратком руководстве по C#](#).

1. Обновите копию шаблонов Bot Framework (VSIX или .NET Core) до последней версии.
2. Создайте новый проект на основе **пустого шаблона Bot** .net Core 3,1.

### NOTE

В этой статье не выделяются инструкции using, которые необходимо добавить, чтобы получить проект для компиляции.

## Добавление пакетов NuGet

1. Откройте проект в вашей предпочтительной среде IDE.
2. Добавьте в качестве зависимости объект Microsoft. Bot. Builder. Dialogs. **адаптивный** 4.10.0 или более поздний пакет NuGet.
3. Зарегистрируйте дополнительные пакеты, которые потребуются для программы Bot:
  - Для адаптивного модульного тестирования в диалоговом окне Добавьте пакет Microsoft. Bot. Builder. Dialogs. **адаптивный. testing** .
  - Для понимания языка LUIS добавьте пакет Microsoft. Bot. Builder. AI. Luis .
  - Для QnA Makerного понимания языка добавьте пакет Microsoft. Bot. Builder. AI. QnA .

Если ваш робот использует пользовательские компоненты, для него могут потребоваться дополнительные пакеты. Автор каждого пользовательского компонента должен четко указать, какие пакеты требуются.

## Регистрация компонентов

Для поддержки настройки Адаптивная библиотека диалогов использует регистрацию компонентов для обнаружения всех компонентов, декларативных типов, областей памяти, арбитров имен путей и преобразователей типов.

При необходимости Зарегистрируйте адаптивные и декларативные компоненты в `ConfigureServices` методе для вашего проекта.

ОБЯЗАТЕЛЬНО	ОПИСАНИЕ	РЕГИСТРАЦИЯ КОМПОНЕНТОВ
Обязательно	Компоненты, общие для всех адаптивных диалоговых окон.	<code>ComponentRegistration.Add(new AdaptiveComponentRegistration());</code>
Обязательно	Общие области памяти и арбитры конфликтов по путям.	<code>ComponentRegistration.Add(new DialogsComponentRegistration());</code>
Необязательно	Компоненты, используемые для модульного тестирования адаптивных диалоговых окон.	<code>ComponentRegistration.Add(new AdaptiveTestingComponentRegistration());</code>
Необязательно	Компоненты, используемые для использования декларативных диалоговых окон.	<code>ComponentRegistration.Add(new DeclarativeComponentRegistration());</code>
Необязательно	Компоненты, используемые для функций формирования языка.	<code>ComponentRegistration.Add(new LanguageGenerationComponentRegistration());</code>
Необязательно	Компоненты, используемые для функций LUIS (понимание языка).	<code>ComponentRegistration.Add(new LuisComponentRegistration());</code>
Необязательно	Компоненты, используемые для QnA Maker (сведения об особенностях языка).	<code>ComponentRegistration.Add(new QnAMakerComponentRegistration());</code>
Необязательно	Компоненты, относящиеся к каналу команд.	<code>ComponentRegistration.Add(new TeamsComponentRegistration());</code>

Если проект не регистрирует компонент, который требуется другой части программы Bot, его можно добавить позже, но может возникнуть ошибка времени инициализации или времени выполнения.

#### NOTE

Если программа-робот использует пользовательские компоненты, необходимо также зарегистрировать эти компоненты. Автор каждого пользовательского компонента должен четко указать, какой метод следует использовать для регистрации своих компонентов.

#### startup.cs

Например, пример адаптивных многострочных запросов регистрирует эти компоненты в `ConfigureServices`.

```
// Register dialog. This sets up memory paths for adaptive.  
ComponentRegistration.Add(new DialogsComponentRegistration());  
  
// Register adaptive component  
ComponentRegistration.Add(new AdaptiveComponentRegistration());  
  
// Register to use language generation.  
ComponentRegistration.Add(new LanguageGenerationComponentRegistration());
```

## Добавить состояние

Как и в случае с другими программы-роботы, вам все равно нужно создать объекты уровня хранилища и объектов управления состоянием.

Вы будете использовать диспетчер диалоговых окон для запуска адаптивных диалоговых окон, и ему понадобятся ссылки на уровень хранилища, а также на объекты состояния пользователя и диалогового окна. (Они необходимы для настройки областей памяти.) Хотя их можно добавить непосредственно в диспетчер диалоговых окон, диспетчер диалоговых окон обнаружит их, если они правильно зарегистрированы в контексте включения.

### startup.cs

Адаптер будет обновлен для использования `ICredentialProvider`, поэтому необходимо зарегистрировать его в `ConfigureServices`.

```
// Create the credential provider to be used with the Bot Framework Adapter.  
services.AddSingleton<ICredentialProvider, ConfigurationCredentialProvider>();
```

Также зарегистрируйте объекты уровня хранилища и управления состоянием, которые будет использовать Bot в `ConfigureServices`.

```
// Create the storage we'll be using for User and Conversation state. (Memory is great for testing  
purposes.)  
services.AddSingleton<IStorage, MemoryStorage>();  
  
// Create the User state. (Used in this bot's Dialog implementation.)  
services.AddSingleton<UserState>();  
  
// Create the Conversation state. (Used by the Dialog system itself.)  
services.AddSingleton<ConversationState>();
```

### AdapterWithErrorHandler.cs

В C# можно использовать `UseStorage` методы адаптера и `UseBotState` для добавления ссылок на объекты хранения и управления состоянием в каждый из них.

Измените сигнатуру для существующего конструктора адаптера.

```
public AdapterWithErrorHandler(ICredentialProvider credentialProvider, ILogger<BotFrameworkHttpAdapter>  
logger, IStorage storage,  
UserState userState, ConversationState conversationState, IConfiguration configuration)  
: base(credentialProvider)
```

Перед установкой свойства `OnTurnError` адаптера добавьте вызовы методов `UseStorage` и `UseBotState`.

```
// These methods add middleware to the adapter. The middleware adds the storage and state objects to the
// turn context each turn so that the dialog manager can retrieve them.
this.UseStorage(storage);
this.UseBotState(userState);
this.UseBotState(conversationState);
```

## Добавление адаптивного диалогового окна

Создайте папку **диалоговых окон**, а затем создайте корневое диалоговое окно, которое будет использовать программа Bot.

**RootDialog.cs** **диалоговых окон \**

Создайте корневой класс диалогового окна. В примере определяется **RootDialog** используемый объект.

```
public class RootDialog : AdaptiveDialog
{
    public RootDialog() : base(nameof(RootDialog))
    {
    }
}
```

На этом этапе диалоговое окно еще не реагирует на пользователя. Вы можете добавить триггер в диалоговое окно (в конструкторе), чтобы дать ему ответ. Ниже приведен пример триггера.

```
Triggers = new List<OnCondition>
{
    new OnUnknownIntent
    {
        Actions =
        {
            new SendActivity("Hi, we are up and running!!"),
        }
    },
};
```

**startup.cs**

Пример регистрирует корневое диалоговое окно в файле запуска в **ConfigureServices**.

```
// The adaptive dialog that will be run by the bot.
services.AddSingleton<RootDialog>();
```

## Добавление диспетчера диалоговых окон

Как и в случае с другими программы-роботы на основе диалоговых окон, разработайте класс Bot для получения диалогового окна в качестве параметра конструктора. Однако создайте и используйте диспетчер диалоговых окон для запуска корневого диалогового окна.

В примере адаптивных запросов подвергнуты-turn корневым диалоговым окном является Адаптивное диалоговое окно. Диспетчер диалоговых окон можно использовать для запуска любого диалогового окна типа, но он необходим, если корневое диалоговое окно или любые его дочерние диалоговые окна являются адаптивными.

**DialogBot.cs**

Переименуйте класс **эмтибот** по умолчанию и файл в **диалогбот**.

В примере определяется программа-робот и добавляется код для создания и использования диспетчера диалоговых окон для выполнения каждого включения корневого диалогового окна. Диспетчер диалоговых окон будет пересыпать все действия в диалоговое окно. В примере с несколькими пересказками этот файл — **программы-роботы \ DialogBot.cs**.

Замените существующее определение класса этим кодом. (Если оператор не включен

```
using Microsoft.Extensions.Logging; , ILogger ссылка является неоднозначным.)
```

```
using Microsoft.Extensions.Logging;

public class DialogBot<T> : ActivityHandler
    where T : Dialog
{
    private readonly DialogManager DialogManager;
    protected readonly ILogger Logger;

    public DialogBot(T rootDialog, ILogger<DialogBot<T>> logger)
    {
        Logger = logger;

        DialogManager = new DialogManager(rootDialog);
    }

    public override async Task OnTurnAsync(ITurnContext turnContext, CancellationToken cancellationToken
= default)
    {
        Logger.LogInformation("Running dialog with Activity.");
        await DialogManager.OnTurnAsync(turnContext, cancellationToken:
cancellationToken).ConfigureAwait(false);
    }
}
```

## startup.cs

В этом примере объект Bot регистрируется в файле запуска в `ConfigureServices`. Замените вызов, чтобы зарегистрировать класс Bot (`services.AddTransient<IBot, EmptyBot>();`) с помощью этого кода.

```
// Create the bot. the ASP Controller is expecting an IBot.
services.AddSingleton<IBot, DialogBot<RootDialog>>();
```

## Компиляция и тестирование

На этом этапе необходимо скомпилировать и запустить Bot. Его можно протестировать в эмуляторе. Если в корневом диалоговом окне не определены триггеры или действия, то Bot не будет отвечать на ввод, но вы должны получить ответ о состоянии HTTP 200 обратно от Bot.

## Дополнительные сведения

Адаптивные диалоговые окна являются мощным средством и имеют множество способов для их настройки и расширения. Дополнительные сведения см. в приведенных ниже статьях.

- [Адаптивные выражения](#)
- [Создание и генераторы языков](#)
- [События и триггеры](#)
- [Действия и входные данные](#)
- [Распознаватели](#)
  - [Что такое LUIS?](#)

- Что такое QnA Maker
- Области памяти и пути
- Декларативные диалоговые окна и Преобразователи типов
  - Создание файла схемы

## Дальнейшие действия

[Создание бота с использованием адаптивных диалогов](#)

# Создание бота с использованием адаптивных диалогов

27.10.2020 • 9 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье показано, как использовать компоненты **Адаптивный диалог** и **Создание текста**, чтобы реализовать те же возможности, что и в каскадной модели.

## Предварительные требования

- Понимание [основных принципов работы ботов, управления состоянием и библиотеки диалогов](#).
- Копия образца [многострочной подсказки](#) в [C#](#).

### Предварительные действия для добавления адаптивного диалога в бота

Прежде чем добавлять в бот адаптивный диалог, необходимо выполнить указанные ниже шаги.

1. Обновите все пакеты NuGet для Bot Builder до версии 4.9.x.
2. Добавьте в проект бота пакет `Microsoft.Bot.Builder.Dialogs.Adaptive`.
3. Обновите адаптер бота, добавив объекты хранилища, пользователя и состояния беседы в контекст каждого шага.
4. Настройте в коде бота запуск или продолжение корневого диалога на каждом шаге с использованием диспетчера диалогов.

## Сведения о примере

В этом примере используются адаптивный диалог, несколько запросов и компонентный диалог для реализации простого взаимодействия, в рамках которого пользователю предлагается несколько вопросов. Вопросы создаются с помощью шаблонов создания текста.

- Для C# они определяются в [RootDialog.cs](#).

Код диалога циклически перебирает следующие действия:

ШАГИ	ШАБЛОН СОЗДАНИЯ ТЕКСТА
Запрос к пользователю о режиме транспортировки	<code>ModeOfTransportPrompt</code>
Запрос имени пользователя	<code>AskForName</code>
Запрос к пользователю, готов ли он указать свой возраст	<code>AgeConfirmPrompt</code>
Если получен положительный ответ, запрос возраста пользователя	Запрос <code>AskForAge</code> с проверкой, которая принимает возраст в диапазоне от 0 до 150
Запрос на подтверждение собранной информации	Запрос <code>ConfirmPrompt</code>

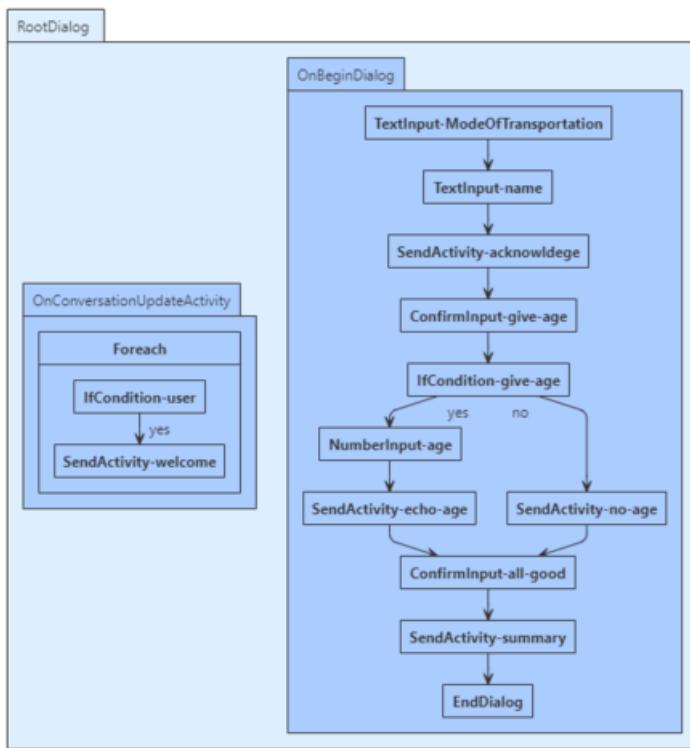
И наконец, если получен положительный ответ, отображается вся собранная информация. В противном случае пользователь получает сообщение о том, что данные не будут сохранены.

# Создание главного диалога

Чтобы использовать диалоги, установите пакет NuGet **Microsoft.Bot.Builder.Dialogs.Adaptive**.

Программа Bot взаимодействует с пользователем с помощью корневого адаптивного диалогового окна.

Затем бот применяет `DialogManager.OnTurnAsync` для запуска диалога.



## Dialogs\RootDialog.cs

Код начинается с создания экземпляра корневого адаптивного диалогового окна. В это же время в диалог добавляются следующие `WelcomeUserSteps` и `OnBeginDialogSteps`. Обратите внимание на определение `paths`, в котором есть ссылка на файл `Dialogs\RootDialog.lg` с шаблонами Создания текста, которые применяются в адаптивном диалоге.

```
public RootDialog() : base(nameof(RootDialog))
{
    string[] paths = { ".", "Dialogs", $"RootDialog.lg" };
    string fullPath = Path.Combine(paths);

    // These steps are executed when this Adaptive Dialog begins
    Triggers = new List<OnCondition>()
    {
        // Add a rule to welcome user
        new OnConversationUpdateActivity()
        {
            Actions = WelcomeUserSteps()
        },

        // Respond to user on message activity
        new OnUnknownIntent()
        {
            Actions = GatheUserInformation()
        },
    };
    Generator = new TemplateEngineLanguageGenerator(Templates.ParseFile(fullPath));
}
```

Обратите также внимание на следующее.

- Генератор шаблонов Создания текста добавляется в адаптивный диалог, чтобы обеспечить использование **шаблонов Создания текста**.
- Добавляются два триггера, действия для которых предоставлены двумя вспомогательными методами.

`WelcomeUserSteps` Метод предоставляет действия, выполняемые при срабатывании триггера. Действия `Foreach` последовательно перебирают список `membersAdded`, приветствуя пользователя, добавленного в беседу.

#### NOTE

В контексте адаптивных диалогов и триггеров допустимыми действиями считаются любые диалоги, а все типы действий (`Foreach`, `IfCondition`, `SendActivity`) являются диалогами.

Некоторые каналы отправляют по два события обновления беседы: одно для добавления в беседу бота, и второе для пользователя. В нашем коде отфильтровываются все случаи, когда бот сам является получателем сообщения. См. сведения в статье [Разделенные на категории действия по каналам](#).

```
private static List<Dialog> WelcomeUserSteps()
{
    return new List<Dialog>()
    {
        // Iterate through membersAdded list and greet user added to the conversation.
        new Foreach()
        {
            ItemsProperty = "turn.activity.membersAdded",
            Actions = new List<Dialog>()
            {
                // Note: Some channels send two conversation update events - one for the Bot added to the
                // conversation and another for user.
                // Filter cases where the bot itself is the recipient of the message.
                new IfCondition()
                {
                    Condition = "$foreach.value.name != turn.activity.recipient.name",
                    Actions = new List<Dialog>()
                    {
                        new SendActivity("Hello, I'm the multi-turn prompt bot. Please send a message to get
started!")
                    }
                }
            }
        };
    }
}
```

В `GatherUserInformation` реализуются **шаги**, которые использует диалог. Он также определяет запросы, использующие шаблоны Создания текста из файла `RootDialog.1g`. Следующий код демонстрирует создание запроса `Name`.

```
private static List<Dialog> GatheUserInformation()
{
    return new List<Dialog>()
    {
        // Ask for user's age and set it in user.userProfile scope.
        new TextInput()
        {
            Prompt = new ActivityTemplate("${ModeOfTransportPrompt()}"),
            // Set the output of the text input to this property in memory.
            Property = "user.userProfile.Transport"
        },
        new TextInput()
    }
}
```

```

        },
        Prompt = new ActivityTemplate("${AskForName()}"),
        Property = "user.userProfile.Name"
    },
    // SendActivity supports full language generation resolution.
    // See here to learn more about language generation
    // https://aka.ms/language-generation
    new SendActivity("${AckName()}"),
    new ConfirmInput()
{
    Prompt = new ActivityTemplate("${AgeConfirmPrompt()}"),
    Property = "turn.ageConfirmation"
},
new IfCondition()
{
    // All conditions are expressed using adaptive expressions.
    // See https://aka.ms/adaptive-expressions to learn more
    Condition = "turn.ageConfirmation == true",
    Actions = new List<Dialog>()
    {
        new NumberInput()
        {
            Prompt = new ActivityTemplate("${AskForAge()}"),
            Property = "user.userProfile.Age",
            // Add validations
            Validations = new List<BoolExpression>()
            {
                // Age must be greater than or equal 1
                "int(this.value) >= 1",
                // Age must be less than 150
                "int(this.value) < 150"
            },
            InvalidPrompt = new ActivityTemplate("${AskForAge.invalid()}"),
            UnrecognizedPrompt = new ActivityTemplate("${AskForAge.unRecognized()}")
        },
        new SendActivity("${UserAgeReadBack()}")
    },
    ElseActions = new List<Dialog>()
    {
        new SendActivity("${NoName()}")
    }
},
new ConfirmInput()
{
    Prompt = new ActivityTemplate("${ConfirmPrompt()}"),
    Property = "turn.finalConfirmation"
},
// Use LG template to come back with the final read out.
// This LG template is a great example of what logic can be wrapped up in LG sub-system.
new SendActivity("${FinalUserProfileReadOut()}"), // examines turn.finalConfirmation
new DeleteProperty() {
    Property = "user.userProfile"
},
new EndDialog(),
};

}

```

Действие `IfCondition` использует адаптивное выражение, чтобы запросить у пользователя возраст или отправить сообщение с подтверждением в зависимости от ответа пользователя на предыдущий вопрос. Оно также использует шаблоны создания текста для форматирования запросов и сообщений.

```

new IfCondition()
{
    // All conditions are expressed using adaptive expressions.
    // See https://aka.ms/adaptive-expressions to learn more
    Condition = "turn.ageConfirmation == true",
    Actions = new List<Dialog>()
    {
        new NumberInput()
        {
            Prompt = new ActivityTemplate("${AskForAge()}"),
            Property = "user.userProfile.Age",
            // Add validations
            Validations = new List<BoolExpression>()
            {
                // Age must be greater than or equal 1
                "int(this.value) >= 1",
                // Age must be less than 150
                "int(this.value) < 150"
            },
            InvalidPrompt = new ActivityTemplate("${AskForAge.invalid()}"),
            UnrecognizedPrompt = new ActivityTemplate("${AskForAge.unRecognized()}")
        },
        new SendActivity("${UserAgeReadBack()}")
    },
    ElseActions = new List<Dialog>()
    {
        new SendActivity("${NoName()}")
    }
},

```

## Регистрация адаптивного диалога

Чтобы разрешить использование адаптивного диалога, код запуска должен зарегистрировать диалог вместе с другими службами, как показано ниже в выделенных строках.

При создании программы-робота создается корневой адаптивный диалог.

## Запуск диалога

### Bots/Dialogs.cs

`DialogManager.OnTurnAsync` выполняет адаптивный диалог с действиями. Предложенная здесь реализация может выполнять `Dialog` любого типа. Система диалогов использует `ConversationState`. Она не использует `UserState`, но это тоже поддерживается в реализации диалога. Метод `DialogManager.OnTurnAsync` обрабатывает сохранение состояния.

```

public override async Task OnTurnAsync(ITurnContext turnContext, CancellationToken cancellationToken =
default)
{
    Logger.LogInformation("Running dialog with Activity.");
    await DialogManager.OnTurnAsync(turnContext, cancellationToken:
cancellationToken).ConfigureAwait(false);
}

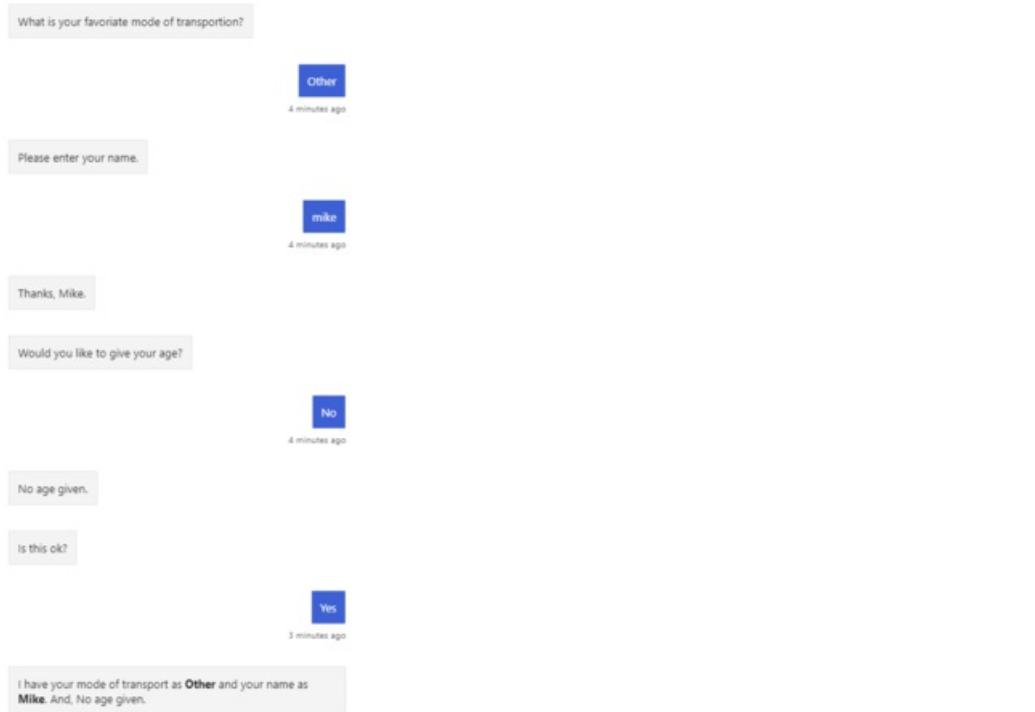
```

#### NOTE

Хранилище в памяти используется только для тестирования и не предназначено для рабочей среды. Для ботов в рабочей среде обязательно используйте постоянное хранилище любого типа.

## Тестирование бота

1. Установите [Bot Framework Emulator](#), если вы этого еще не сделали.
2. Выполните этот пример на локальном компьютере.
3. Запустите эмулятор, подключитесь к боту и отправьте несколько сообщений, как показано ниже.



## Дальнейшие действия

[Создание бота с использованием комбинированных типов диалогов](#)

# Создание бота с использованием адаптивных, компонентных, каскадных и настраиваемых диалогов

27.03.2021 • 14 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Все диалоги являются производными от базового класса `dialog`. Если вы используете [диспетчер диалогов](#) для запуска корневого диалога, все классы диалогов могут работать вместе. В этой статье показано, как объединить в одном боте компонентные, каскадные, настраиваемые и адаптивные диалоги.

Также здесь приводится код, который позволяет всем диалогам работать вместе. Подробные статьи о каждом типе диалогов см. в [дополнительных материалах](#).

## Предварительные требования

- Понимание [принципов работы ботов, принципов управления состоянием, принципов работы библиотек диалогов](#) и [принципов работы адаптивных диалогов](#).
- Копия **каскадного или пользовательского диалогового окна с адаптивным** примером в [C#](#)

### Предварительные действия для добавления адаптивного диалога в бота

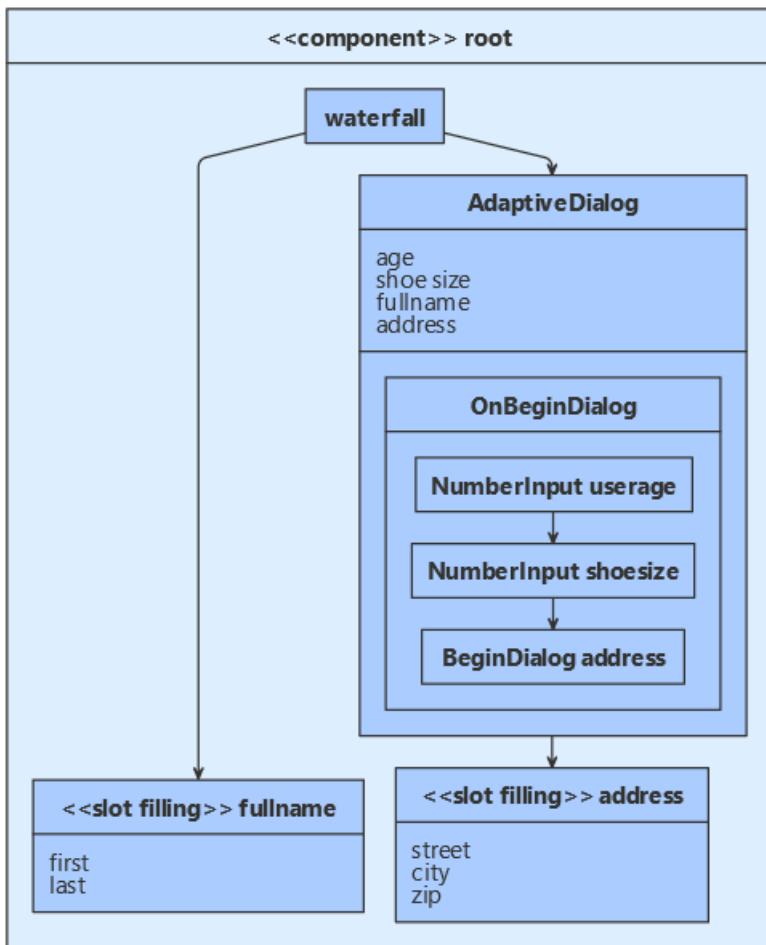
Прежде чем добавлять в бот адаптивный диалог, необходимо выполнить указанные ниже шаги. Эти шаги подробнее описаны в разделе [Создание бота с использованием адаптивных диалогов](#).

1. Обновите все пакеты NuGet для Bot Builder до версии 4.9.x.
2. Добавьте в проект бота пакет `Microsoft.Bot.Builder.Dialogs.Adaptive`.
3. Обновите адаптер бота, добавив объекты хранилища, пользователя и состояния беседы в контекст каждого шага.
4. Настройте в коде бота запуск или продолжение корневого диалога на каждом шаге с использованием диспетчера диалогов.

## Сведения о примере

Этот пример демонстрирует способ объединения разных типов диалогов в одном боте. Это не рекомендации по разработке потока беседы. Этот пример выполняет следующее:

- Определяет класс настраиваемого диалога `заполнения слотов`.
- Создает корневой компонентный диалог:
  - Каскадный диалог управляет потоком беседы верхнего уровня.
  - Адаптивный диалог и два настраиваемых диалога заполнения слотов управляет остальной частью потока беседы.



Настраиваемые диалоги заполнения слотов принимают список свойств (слотов для заполнения). Затем каждый такой диалог предлагает запросы по отсутствующим значениям, пока не будут заполнены все слоты. В этом примере свойство *привязывается* к адаптивному диалогу, чтобы он также мог заполнять слоты.

В этой статье показано, как объединить несколько типов диалогов. См. сведения в статье о [настройке бота для использования адаптивных диалогов](#). См. сведения в статье об [использовании адаптивных диалогов для сбора вводимых пользователем данных](#).

## Настраиваемые диалоги заполнения слотов

Настраиваемым считается любой диалог, являющийся производным от любого класса диалогов в пакете SDK и переопределяющий один или несколько из базовых методов диалога: *begin dialog*, *continue dialog*, *resume dialog* или *end dialog*.

При создании диалога заполнения слотов вы предоставляете список определений для *слотов*, которые будет заполнять этот диалог. Этот диалог переопределяет методы начала, продолжения и возобновления диалога, чтобы итеративно запрашивать у пользователя данные для заполнения каждого слота. Когда все слоты будут заполнены, диалог завершится и вернет собранную информацию.

Каждое определение слота содержит имя запроса диалога, с помощью которого собирается информация.

Корневой диалог создает два диалога заполнения слотов: один из них получает полное имя пользователя, а второй — адрес. Также он создает *текстовый запрос*, с помощью которого эти диалоги будут заполнять свои слоты.

Класс `SlotDetails` описывает собираемую информацию и запрос, используемый для ее сбора.

```
/// <summary>
/// A list of SlotDetails defines the behavior of our SlotFillingDialog.
/// This class represents a description of a single 'slot'. It contains the name of the property we want to
/// gather
/// and the id of the corresponding dialog that should be used to gather that property. The id is that used
/// when the
/// dialog was added to the current DialogSet. Typically this id is that of a prompt but it could also be
/// the id of
/// another slot dialog.
/// </summary>
public class SlotDetails
{
    public SlotDetails(string name, string dialogId, string prompt = null, string retryPrompt = null)
        : this(name, dialogId, new PromptOptions
    {
        Prompt = MessageFactory.Text(prompt),
        RetryPrompt = MessageFactory.Text(retryPrompt),
    })
    {
    }

    public SlotDetails(string name, string dialogId, PromptOptions options)
    {
        Name = name;
        DialogId = dialogId;
        Options = options;
    }

    public string Name { get; set; }

    public string DialogId { get; set; }

    public PromptOptions Options { get; set; }
}
```

## Dialogs\SlotFillingDialog.cs

Класс `SlotFillingDialog` является производным от базового класса `Dialog`.

Он отслеживает уже собранные значения, последний запрошенный слот и сведения об еще не заполненных слотах.

```
// Custom dialogs might define their own custom state.
// Similarly to the Waterfall dialog we will have a set of values in the ConversationState. However, rather
// than persisting
// an index we will persist the last property we prompted for. This way when we resume this code following a
// prompt we will
// have remembered what property we were filling.
private const string SlotName = "slot";
private const string PersistedValues = "values";

// The list of slots defines the properties to collect and the dialogs to use to collect them.
private readonly List<SlotDetails> _slots;

public SlotFillingDialog(string dialogId, List<SlotDetails> slots)
    : base(dialogId)
{
    _slots = slots ?? throw new ArgumentNullException(nameof(slots));
}
```

Основная логика для сбора недостающей информации находится во вспомогательном методе `RunPromptAsync`. Когда сбор информации закончится, он завершит диалог и вернет информацию.

```

/// <summary>
/// This helper function contains the core logic of this dialog. The main idea is to compare the state we
have gathered with the
/// list of slots we have been asked to fill. When we find an empty slot we execute the corresponding
prompt.
/// </summary>
/// <param name="dialogContext">A handle on the runtime.</param>
/// <param name="cancellationToken">The cancellation token.</param>
/// <returns>A DialogTurnResult indicating the state of this dialog to the caller.</returns>
private Task<DialogTurnResult> RunPromptAsync(DialogContext dialogContext, CancellationToken
cancellationToken)
{
    var state = GetPersistedValues(dialogContext.ActiveDialog);

    // Run through the list of slots until we find one that hasn't been filled yet.
    var unfilledSlot = _slots.FirstOrDefault(item => !state.ContainsKey(item.Name));

    // If we have an unfilled slot we will try to fill it
    if (unfilledSlot != null)
    {
        // The name of the slot we will be prompting to fill.
        dialogContext.ActiveDialog.State[SlotName] = unfilledSlot.Name;

        // If the slot contains prompt text create the PromptOptions.

        // Run the child dialog
        return dialogContext.BeginDialogAsync(unfilledSlot.DialogId, unfilledSlot.Options,
cancellationToken);
    }
    else
    {
        // No more slots to fill so end the dialog.
        return dialogContext.EndDialogAsync(state);
    }
}

```

Дополнительные сведения о реализации пользовательских диалогов см. в обсуждении диалога *отмены и справки* в статье об [обработке пользовательских прерываний](#).

## Корневой компонентный диалог

Корневой диалог выполняет следующее:

- Определяет все слоты для заполнения: для себя, двух диалогов заполнения и адаптивного диалога.
- Создает метод доступа к свойству состояния пользователя, где будет храниться собранная информация.
- Создает адаптивный диалог, два диалога для заполнения слотов, каскадный диалог и запросы для использования в каскадном диалоге и диалогах заполнения слотов.
- Указывает каскадный диалог в качестве исходного для выполнения при первом запуске компонента.

Каскадный диалог будет агрегировать всю собранную информацию и хранить ее в свойстве состояния пользователя.

Каскадный и адаптивный диалоги будут описаны в следующих разделах.

### Dialogs\RootDialog.cs

Класс `RootDialog` является `ComponentDialog`. Он определяет свойство состояния пользователя, где будет храниться собранная информация.

```
public class RootDialog : ComponentDialog
{
    private IStatePropertyAccessor< JObject> _userStateAccessor;

    public RootDialog(UserState userState)
        : base("root")
    {
        _userStateAccessor = userState.CreateProperty< JObject>("result");
```

Его конструктор создает все необходимые диалоги, включая адаптивный диалог

```
adaptiveSlotFillingDialog .
```

Затем он добавляет эти диалоги в набор диалогов.

```
// Add the various dialogs that will be used to the DialogSet.
AddDialog(new SlotFillingDialog("address", address_slots));
AddDialog(new SlotFillingDialog("fullname", fullname_slots));
AddDialog(new TextPrompt("text"));
AddDialog(new NumberPrompt< int >("number", defaultLocale: Culture.English));
AddDialog(new NumberPrompt< float >("shoesize", ShoeSizeAsync, defaultLocale: Culture.English));

// We will instead have adaptive dialog do the slot filling by invoking the custom dialog
// AddDialog(new SlotFillingDialog("slot-dialog", slots));

// Add adaptive dialog
AddDialog(adaptiveSlotFillingDialog);

// Defines a simple two step Waterfall to test the slot dialog.
AddDialog(new WaterfallDialog("waterfall", new WaterfallStep[] { StartDialogAsync, DoAdaptiveDialog,
ProcessResultsAsync }));

// The initial child Dialog to run.
InitialDialogId = "waterfall";
}
```

## Каскадный диалог

Каскадный диалог содержит следующие три шага:

1. Запуск диалога заполнения слотов fullname, который получит и вернет полное имя пользователя.
2. Получение имени пользователя и запуск адаптивного диалога, который получит остальную информацию о пользователе.
3. Запись сведений о пользователе в метод доступа свойства состояния пользователя и вывод пользователю сводки по собранной информации.

Dialogs\RootDialog.cs

```

private async Task<DialogTurnResult> StartDialogAsync(WaterfallStepContext stepContext, CancellationToken cancellationToken)
{
    // Start the child dialog. This will just get the user's first and last name.
    return await stepContext.BeginDialogAsync("fullname", null, cancellationToken);
}

private async Task<DialogTurnResult> DoAdaptiveDialog(WaterfallStepContext stepContext, CancellationToken cancellationToken)
{
    object adaptiveOptions = null;
    if (stepContext.Result is IDictionary<string, object> result && result.Count > 0)
    {
        adaptiveOptions = new { fullname = result };
    }
    // begin the adaptive dialog. This in-turn will get user's age, shoe-size using adaptive inputs and
    subsequently
    // call the custom slot filling dialog to fill user address.
    return await stepContext.BeginDialogAsync(nameof(AdaptiveDialog), adaptiveOptions, cancellationToken);
}

private async Task<DialogTurnResult> ProcessResultsAsync(WaterfallStepContext stepContext, CancellationToken cancellationToken)
{
    // To demonstrate that the slot dialog collected all the properties we will echo them back to the user.
    if (stepContext.Result is IDictionary<string, object> result && result.Count > 0)
    {
        // Now the waterfall is complete, save the data we have gathered into UserState.
        // This includes data returned by the adaptive dialog.
        var obj = await _userStateAccessor.GetAsync(stepContext.Context, () => new JObject());
        obj["data"] = new JObject
        {
            { "fullname", $"{result["fullname"]}" },
            { "shoesize", $"{result["shoesize"]}" },
            { "address", $"{result["address"]}" },
        };

        await stepContext.Context.SendActivityAsync(MessageFactory.Text(obj["data"])
["fullname"].Value<string>(), cancellationToken);
        await stepContext.Context.SendActivityAsync(MessageFactory.Text(obj["data"])
["shoesize"].Value<string>(), cancellationToken);
        await stepContext.Context.SendActivityAsync(MessageFactory.Text(obj["data"]["address"].Value<string>()
()), cancellationToken);
    }

    // Remember to call EndAsync to indicate to the runtime that this is the end of our waterfall.
    return await stepContext.EndDialogAsync();
}

```

## Адаптивный диалог

Адаптивный диалог определяет один триггер, который выполняется при запуске диалога. Этот триггер выполняет следующее:

1. Применяет диалог ввода, чтобы узнать возраст пользователя.
2. Применяет диалог ввода, чтобы узнать размер обуви пользователя.
3. Запускает диалог заполнения словов address, чтобы узнать адрес пользователя.
4. Устанавливает результирующее значение триггера и завершает работу.

Так как других действий в очереди нет, адаптивный диалог также завершает работу и возвращает полученное результирующее значение.

В адаптивном диалоге применяется генератор кода языка для форматирования текста и включения

значений из состояния бота и диалога. (См. сведения об [использовании генераторов в адаптивных диалогах](#).)

## Dialogs\RootDialog.cs

```
// define adaptive dialog
var adaptiveSlotFillingDialog = new AdaptiveDialog();
adaptiveSlotFillingDialog.Id = nameof(AdaptiveDialog);

// Set a language generator
// You can see other adaptive dialog samples to learn how to externalize generation resources into .lg
files.
adaptiveSlotFillingDialog.Generator = new TemplateEngineLanguageGenerator();

// add set of actions to perform when the adaptive dialog begins.
adaptiveSlotFillingDialog.Triggers.Add(new OnBeginDialog()
{
    Actions = new List<Dialog>()
    {
        // any options passed into adaptive dialog is automatically available under dialog.xxx
        // get user age
        new NumberInput()
        {
            Property = "dialog.usage",

            // use information passed in to the adaptive dialog.
            // See https://aka.ms/language-generation to learn more.
            Prompt = new ActivityTemplate("Hello ${dialog.fullname.first}, what is your age?"),
            Validations = new List<BoolExpression>()
            {
                // Conditions are written using Adaptive Expressions.
                // See https://aka.ms/adaptive-expressions to learn more.
                "int(this.value) >= 1",
                "int(this.value) <= 150"
            },
            InvalidPrompt = new ActivityTemplate("Sorry, ${this.value} does not work. Looking for age to be
between 1-150. What is your age?"),
            UnrecognizedPrompt = new ActivityTemplate("Sorry, I did not understand ${this.value}. What is
your age?"),
            MaxTurnCount = 3,
            DefaultValue = "=30",
            DefaultValueResponse = new ActivityTemplate("Sorry, this is not working. For now, I'm setting
your age to ${this.defaultValue}"),
            AllowInterruptions = false
        },
        new NumberInput()
        {
            Property = "dialog.shoesize",
            Prompt = new ActivityTemplate("Please enter your shoe size."),
            InvalidPrompt = new ActivityTemplate("Sorry ${this.value} does not work. You must enter a size
between 0 and 16. Half sizes are acceptable."),
            Validations = new List<BoolExpression>()
            {
                // size can only between 0-16
                "int(this.value) >= 0 && int(this.value) <= 16",
                // can only full or half size
                "isMatch(string(this.value), '^[0-9]+(\.\5)*$')"
            },
            AllowInterruptions = false
        },
        // get address - adaptive is calling the custom slot filling dialog here.
        new BeginDialog()
        {
            Dialog = "address",
            ResultProperty = "dialog.address"
        },
        // return everything under dialog scope.
        ...
    }
})
```

```
new Endialog()
{
    Value = "=dialog"
}
}) ;
```

## Тестирование бота

1. Установите [Bot Framework Emulator](#), если вы этого еще не сделали.
2. Выполните этот пример на локальном компьютере.
3. Запустите эмулятор, подключитесь к роботу и ответьте на запросы: имя и фамилия, размер обуви, улица, город и почтовый индекс.
4. После этого бот отобразит собранную информацию.
5. Отправьте боту любое сообщение, чтобы повторить процесс.

## Дополнительные сведения

Дополнительные сведения об использовании каждого типа диалогов см. в следующих статьях.

ТИП ДИАЛОГА	СТАТЬЯ
Адаптивные диалоги и диалоги ввода данных	<a href="#">Создание бота с использованием адаптивных диалогов</a>
Компонентные диалоги	<a href="#">Управление сложностью диалогов</a>
Настраиваемые диалоги	<a href="#">Обработка прерываний диалога пользователем</a>
Каскадные диалоги и диалоги запросов	<a href="#">Реализация процесса общения</a>

# Управление пользовательскими прерываниями в адаптивных диалогах

27.03.2021 • 27 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Обработка прерываний является важным аспектом для создания надежного бота. Пользователи не всегда будут следовать определенному потоку диалога. Часто им нужно задать вопрос в середине или отменить этот процесс. В этой статье описываются некоторые распространенные способы обработки прерываний пользователей в роботе.

## Предварительные условия

Обработка прерываний — это расширенный раздел для разработки на Bot. Чтобы получить максимальную выработку из этой статьи, вам потребуются хорошие знания об основных концепциях пакета SDK для Bot Framework и адаптивных диалоговых окнах, а также о том, как разработать робот, который объединяет адаптивные диалоговые окна. Прежде чем продолжить, ознакомьтесь со статьями, перечисленными в первых двух пунктах.

- Основные [сведения об основах Bot, управлении состояниеми библиотеках диалоговых окон](#) Bot Framework SDK.
- Понимание адаптивных концепций диалоговых окон, рассмотренных в статьях [Введение в адаптивные диалоговые окна](#), [запрос ввода данных пользователем в адаптивных диалоговых окнах](#) и [обработка прерываний в адаптивных диалоговых окнах](#).
- Копия `InterruptionsBot` примера на [языке C#](#).
- Полезно знать, как [создать робот с помощью адаптивных диалоговых окон](#), но в этой статье `InterruptionsBot` подробно рассматривается пример.

### NOTE

`InterruptionsBot` Образец в настоящее время доступен только в [C#](#).

## Настройка LUIS для работы в Bot

В этом разделе подробно описано, как создать ресурсы LUIS в портал Azure, а затем настроить и установить LUIS для работы с этим примером. Если вы уже сделали это, можно перейти к [примеру интерреквионбот](#).

Что рассматривается в этом разделе:

1. [Создание ресурсов LUIS на портале Azure](#)
2. [Получите свой ключ разработки](#)
3. [Подключение программы Bot к ресурсу LUIS в Azure с помощью интерфейса командной строки Bot Framework](#)
4. [Обновление файла конфигурации](#)

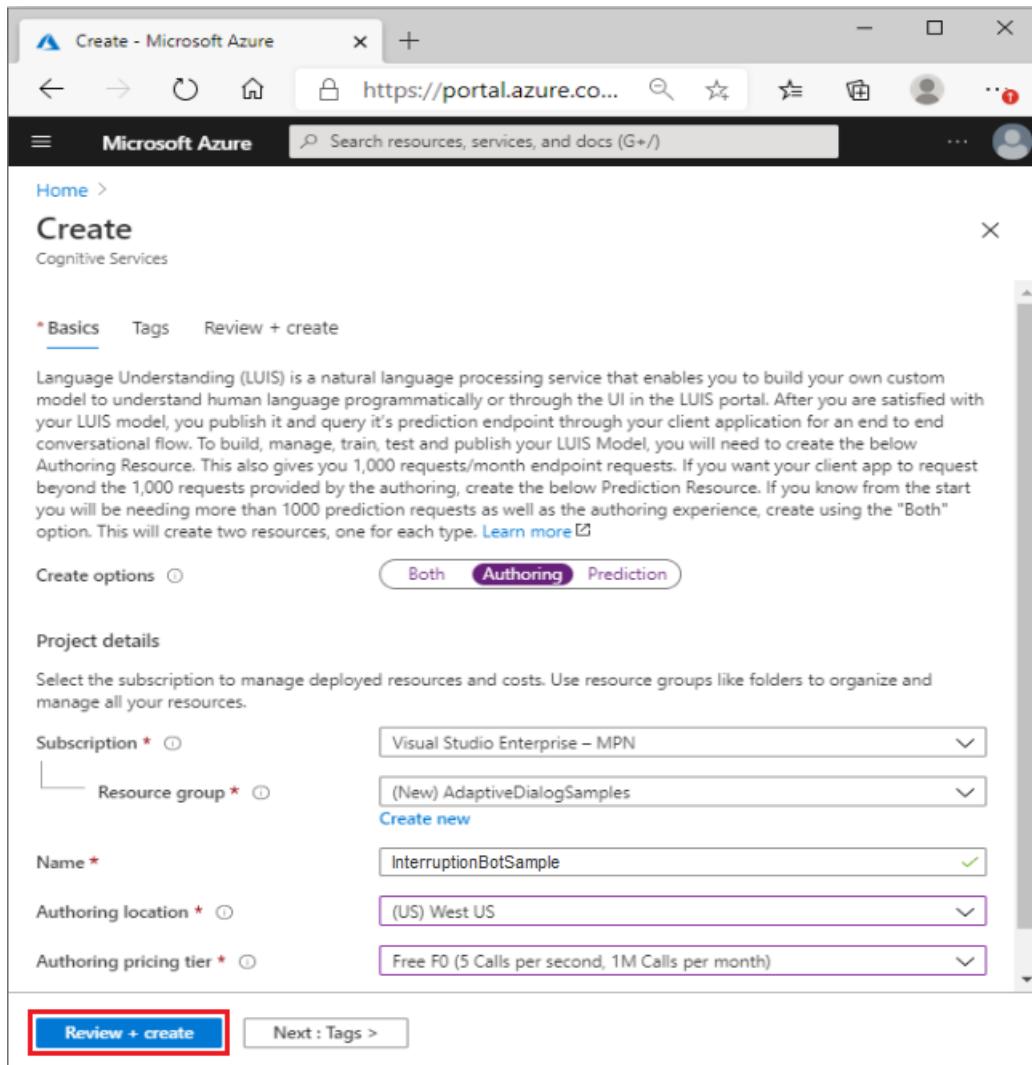
### Создание ресурсов LUIS на портале Azure

1. Перейдите на страницу [создания Cognitive Services Azure](#).

2. В разделе **Параметры создания** выберите **Создание** и настройка.



3. Введите значения для каждого поля, а затем нажмите кнопку "проверить и создать".



The screenshot shows the Microsoft Azure portal's 'Create' interface for Cognitive Services. The 'Authoring' tab is active in the top navigation bar. On the left, the 'Basics' tab is selected in the sidebar. The main form contains fields for 'Subscription' (Visual Studio Enterprise – MPN), 'Resource group' ((New) AdaptiveDialogSamples), 'Name' (InterruptionBotSample), 'Authoring location' ((US) West US), and 'Authoring pricing tier' (Free F0 (5 Calls per second, 1M Calls per month)). At the bottom left, the 'Review + create' button is highlighted with a red box.

#### NOTE

При вводе **группы ресурсов** и **имени** Помните, что эти значения нельзя изменить позже. Также обратите внимание, что значение, присваиваемое **имени**, будет являться частью URL-адреса **конечной точки**.

4. Проверьте значения, чтобы убедиться, что они верны, а затем нажмите кнопку **создать**.

#### Получите свой ключ разработки

Теперь, когда вы создали ресурс LUIS в портал Azure, вы можете получить свой ключ разработки.

- Когда Azure завершит создание ресурсов LUIS в портал Azure, вы увидите уведомление о **завершении развертывания**, щелкните **Переход к ресурсу**.

The screenshot shows the Microsoft Azure Cognitive Services LUIS AllInOne Overview page. It displays a success message: "Your deployment is complete". Below it, deployment details are listed: Deployment name: Microsoft.CognitiveServicesLUISAllInOne, Subscription: Visual Studio Enterprise – MPN, Resource group: AdaptiveDialog. There are links for "Deployment details (Download)" and "Next steps". A prominent blue button labeled "Go to resource" is highlighted with a red box. On the right side, there are links to Security Center, Free Microsoft tutorials, and Start learning today.

2. На левой панели выберите **ключи и конечная точка**.
3. Скопируйте значение **ключа 1**, это ваш *ключ разработки*. Необходимо ввести это значение в качестве значения для:
  - `LuisAPIKey` : Параметр в файле конфигурации.
  - `--authoringKey` : Свойство `bf luis:build` команды CLI, описанное в следующем разделе.

The screenshot shows the Microsoft Azure Cognitive Services InterruptionBotSample Keys and Endpoint page. The left sidebar has a red box around the "Keys and Endpoint" item under RESOURCE MANAGEMENT. The main area shows two keys: KEY 1 and KEY 2, both with copy icons. Below them is an ENDPOINT field containing the URL `https://interruptionsbot-authoring.cognitiveservices.azure.com/`, which is also highlighted with a red box. The LOCATION dropdown is set to "westus".

4. Скопируйте и сохраните **конечную точку**. Это значение будет присвоено `LuisAPIHostName` в файле конфигурации.

С помощью новых ресурсов LUIS в портал Azure вы можете подключить к нему робот.

### Подключение программы Bot к ресурсу LUIS в Azure с помощью интерфейса командной строки Bot Framework

В этом разделе объясняется, как использовать интерфейс командной строки Bot Framework для подключения программы Bot к ресурсам LUIS в Azure. Это автоматизирует различные задачи, необходимые для создания, обновления, обучения и публикации LUIS приложений для каждого Lu-файла для программы-робота. Чтобы использовать это, сначала требуется Node.js и интерфейс командной строки Bot Framework.

1. Если Node.js установлен, убедитесь, что у вас установлена версия 10.14 или более поздняя, выполнив следующую команду из командной строки: `npm node.js -version`. Последнюю версию

можно получить, выполнив следующую команду в командной строке: `npm i -g npm`.

Если он не установлен, его можно установить на [странице загрузки Node.js](#).

- С помощью Node.js установите последнюю версию интерфейса командной строки для Bot Framework в командной строке.

```
npm i -g @microsoft/botframework-cli
```

- В командной строке перейдите в корневой каталог образца Interruptionbot, как правило, в `..\samples\csharp_dotnetcore\adaptive-dialog\05.interruptions-bot`.

Теперь вы можете подключить робот к ресурсам LUIS в Azure с помощью интерфейса командной строки Bot Framework.

- В командной строке, находясь в корневом каталоге исходного кода проекта, выполните следующую команду:

```
bf luis:build --in Dialogs --out generated --log --botName InterruptionBotSample --authoringKey <your-authoring-key>
```

Ниже приведены **параметры сборки BF Luis**:

- `in` — Это каталог вместе с вложенными каталогами, в которых будут искаться файлы. lu.
- `out` — Это каталог, в который сохраняются файлы, созданные этим процессом.
- `log` : Логическое значение, определяющее, создается ли журнал во время этого процесса.
- `botName` : Используйте то же значение, которое вы использовали для **имени** на шаге 3 раздела [CREATE Luis Resources in портал Azure](#) выше.
- `authoringKey` : Ваш [ключ разработки](#).

Выполнение `bf luis:build` команды выполняет несколько действий. Она передает всю информацию, содержащуюся во всех файлах. lu, в ресурс LUIS в Azure, а затем выполняет необходимые учебные курсы и публикацию LUIS. Теперь все, что нужно сделать, — это обновить файл конфигурации, указав сведения, созданные в результате действий, выполненных ранее.

#### Созданные файлы

Во время процесса создается два типа файлов `bf luis:build`:

- `.json` Файл, содержащий сведения, необходимые для программы Bot, которые необходимо добавить в [файл конфигурации](#). Этот файл называется `luis.settings.<youralias>.<region>.json`.
- `.dialog` файлы, которые полезны при использовании декларативной формы адаптивных диалоговых окон.

Для этого пошагового руководства не требуются два созданных диалоговых файла.

#### Обновление файла конфигурации

Добавьте следующие значения в `appsettings.json`:

```
"luis": {  
    "GetUserProfileDialog_en_us_lu": "",  
    "RootDialog_en_us_lu": "",  
    "LuisAPIKey": "",  
    "LuisAPIHostName": ""  
}
```

#### NOTE

Созданные имена файлов могут различаться в зависимости от локальных параметров компьютера. В примере, используемом здесь, параметр locale имеет значение `en_us`.

- `GetUserProfileDialog_en_us_lu`: получить это значение из файла с именем `Luis.Settings.<youralias>.JSON`, который был создан как часть выполнения `bf luis:build` команды.
- `RootDialog_en_us_lu`: получить это значение из файла с именем `Luis.Settings.<youralias>.JSON`, который был создан как часть выполнения `bf luis:build` команды.
- **Лусапикэй**: это ваш [ключ разработки](#).
- **Лусапихостнаме**: это значение URL-адреса **конечной точки**, полученное из раздела **ключи и конечная точка** в Azure, которое будет выглядеть примерно так:  
`https://InterruptionBotSample.cognitiveservices.azure.com`.

Теперь вы можете использовать LUIS в Bot.

## Пример Интерруптионбот

Пример, используемый в этой статье, демонстрирует создание программы-робота, использующей LUIS и адаптивные диалоговые окна для достижения более сложных концепций LU, включая обработку прерываний. В любой момент во время диалога с программой-роботом пользователь может выдавать команды *справки* или *отмены*, чтобы прервать текущий поток диалога. В примере Интерруптионбот два адаптивных диалоговых окна, и каждое диалоговое окно включает обработку прерываний, относящуюся к их потоку диалога.

### RootDialog

Рутдиалог — это корневой адаптивный диалог для этого робота. Это родительский элемент только для другого адаптивного диалогового окна в этой роботе: [усерпрофиледиалог](#).

#### Распознаватель Рутдиалог

Первое, что происходит при `rootDialog` создании, — это определение распознавателя. В этом примере вы будете использовать Адаптивный распознаватель LUIS. Инструкции по началу работы с программой-роботом с помощью распознавателя LUIS см. в разделе [Настройка Luis для работы в Bot](#).

Каждое Адаптивное диалоговое окно имеет собственный распознаватель, и все адаптивные диалоги, использующие распознаватель LUIS, могут иметь один или несколько файлов. lu. Этот файл обычно имеет то же имя, что и имя файла, содержащего диалоговое окно, с расширением. lu, например если файл, в котором размещается диалоговое окно, называется *рутдиалог*, то файл. lu будет *RootDialog.lu*. Lu-файл используется исключительно этим диалоговым окном. Дополнительные сведения о файлах. lu см. в разделе [Формат файла article.lu](#).

В файле. lu определяются способы, [фразы продолжительностью и сущности, которые](#) будут использоваться в этом диалоговом окне. Если Адаптивное диалоговое окно не определяет триггер для выполнения определенной цели, но одно из его родительских диалоговых окон, то механизм консультации позволяет родительскому диалоговому окну выполнять utterance. После завершения процесса пользователь возвращается в тот момент, когда он был до прерывания.

Распознаватель задается путем вызова `CreateLuisRecognizer()` метода и передачи параметров конфигурации Bot, содержащих значения, хранящиеся в `appsettings.json`, которые содержат необходимые параметры Luis.

```
// Create instance of adaptive dialog.
var rootDialog = new AdaptiveDialog(nameof(AdaptiveDialog))
{
    Recognizer = CreateLuisRecognizer(this.configuration),
}
```

`CreateLuisRecognizer()` Сначала метод проверяет наличие необходимых значений в файле конфигурации (`appsettings.json`) и, если это так, создает новый `LuisAdaptiveRecognizer` именованный объект `Recognizer`. Теперь ваш распознаватель LUIS готов.

```
private static Recognizer CreateLuisRecognizer(IConfiguration configuration)
{
    if (string.IsNullOrEmpty(configuration["luis:RootDialog_en_us_lu"]) ||
        string.IsNullOrEmpty(configuration["luis:LuisAPIKey"]) ||
        string.IsNullOrEmpty(configuration["luis:LuisAPIHostName"]))
    {
        throw new Exception("NOTE: LUIS is not configured for RootDialog. To enable all capabilities, add 'LuisAppId-RootDialog', 'LuisAPIKey' and 'LuisAPIHostName' to the appsettings.json file.");
    }

    return new LuisAdaptiveRecognizer()
    {
        ApplicationId = configuration["luis:RootDialog_en_us_lu"],
        EndpointKey = configuration["luis:LuisAPIKey"],
        Endpoint = configuration["luis:LuisAPIHostName"]
    };
}
```

#### Генератор Рутдиалог

Для генератора требуется допустимый файл **языкового формирования** (.LG), определяющий шаблоны создания языка, которые будут использоваться этим диалоговым окном. Этому файлу присваивается то же имя, что и имя файла, содержащего диалоговое окно с расширением LG, например если файл, в котором размещается диалоговое окно, называется *рутдиалог*, то LG-файл будет *рутдиалог.LG*. Файл. LG используется исключительно в этом диалоговом окне.

```
_templates = Templates.ParseFile(Path.Combine(".", "Dialogs", "RootDialog", "RootDialog.lg"));

// Create instance of adaptive dialog.
var rootDialog = new AdaptiveDialog(nameof(AdaptiveDialog))
{
    Recognizer = CreateLuisRecognizer(this.configuration),
    Generator = new TemplateEngineLanguageGenerator(_templates),
```

#### Триггеры Рутдиалог

При запуске этого примера программы-робота первое, что корневое диалоговое окно выполняется после настройки его распознавателя и генератора, заключается в обработке `conversationUpdate` события действия с помощью триггера `онконверсационупдатеактивити`. Это будет срабатывать каждый раз, когда новый пользователь присоединяется к беседе и вызывает действия, возвращаемые `WelcomeUserActions` методом. Первый — это `SendActivity` действие с использованием  `${WelcomeUser() }` шаблона, определенного в файле **рутдиалог.LG**. При этом отправляется сообщение Hello и три предлагаемых действия: "профиль", "Справка" и "Отмена". После этого пользователь покажет сообщение Hello, где отображаются три параметра, которые можно выбрать напрямую. Пользователь может выбрать одно из действий или ввести любое значение в поле ввода. Краткое описание шаблона см. в следующем подсказке.

## TIP

В приведенном выше описании  `${WelcomeUser() }` передается в качестве аргумента в `SendActivity` действие. `WelcomeUser` определен в корневом диалоговом окне File Dialogs . `LG` , а — это имя [шаблона структурированного ответа](#) .

Ссылки на шаблоны структурированных ответов обозначаются следующим образом:

- Добавление открывающей и закрывающей круглой скобки () сразу после имени шаблона: `WelcomeUser()`
- Окружающие его с фигурными скобками {} : `{WelcomeUser()}`
- Перед префиксом знака доллара \$ :  `${WelcomeUser() }`

Далее представлены параметры, отображаемые пользователю, и объясняется, что происходит при выборе.

### Профиль

Если этот флагок установлен:

1. При этом создается utterance "Profile" от пользователя.
2. Распознаватель корневого диалогового окна распознает намерение "UserProfile".
3. Срабатывает триггер корневого диалогового окна `OnIntent` для этой цели, который запускает  `userProfileDialog`  диалоговое окно, другое Адаптивное диалоговое окно.

Диалоговое окно профиля пользователя имеет собственный распознаватель и генератор и выполняет собственные триггеры и действия в ответ на введенные пользователем данные, как описано далее в разделе [жетусерпрофилдиалог](#) .

Ниже приведен код `OnIntent` триггера в **рутдиалог. CS**, который обрабатывает намерение *UserProfile*:

```
new OnIntent()
{
    Intent = "GetUserProfile",
    Actions = new List<Dialog>()
    {
        new BeginDialog()
        {
            Dialog = nameof(GetUserProfileDialog)
        }
    }
},
```

### Справка

Если этот флагок установлен:

1. Это приведет к созданию utterance "Help" от пользователя.
2. Распознаватель корневого диалогового окна распознает цель "Справка".
3. Активируется триггер корневого диалогового окна `OnIntent` для этой цели, что приводит к выполнению его действий.
4. `SendActivity` Действие называется передачей в  `${RootHelp() }` качестве параметра. В результате пользователь получает сообщение, аналогичное тому, которое было получено при запуске диалога. Подробное описание работы см. в предыдущем [подсказке](#).

Ниже приведен `OnIntent` код триггера, который обрабатывает цель *справки* в рутдиалог. cs:

```
new OnIntent()
{
    Intent = "Help",
    Actions = new List<Dialog>()
    {
        new SendActivity("${RootHelp()}")
    }
},
```

Ниже приведен [шаблон структурированного ответа](#) в рутдиалог. LG , связанный с намерением «Help»:

```
# RootHelp
[Activity
    Text = I'm a sample interruptions bot
    SuggestedActions = Profile | Help | Cancel
]
```

Отменить

Если этот флагок установлен:

1. Bot запускает `intent` событие со значением "Cancel".
2. `OnIntent` Триггер, содержащий `намерения = "Cancel"`, срабатывает, что приводит к выполнению его действий.
3. Это приводит к выполнению действия [конфирминпут](#) , которое отображает сообщение "вы действительно хотите отменить?"
4. Затем `IfCondition` выполняется действие, чтобы ответить на `ConfirmInput` действие.
  - a. Если пользователь отвечает работоспособным, `turn.confirm` для параметра задается значение `true` И `CancelReadBack` выполняется шаблон LG, в результате чего пользователю отправляется сообщение: *Убедитесь, что отменяются все диалоги...*, затем выполняет действие [канцелаллдиалогс](#) , закрывая все диалоговые окна.
  - b. Если пользователь не отвечает работоспособным, `turn.confirm` для параметра задается значение `false` И `Cancelcancelled` выполняется шаблон LG, в результате чего пользователю отправляется сообщение: *проблема не возникает*, и диалог затем остается там, где она была отключена.

Ниже приведен `OnIntent` код триггера в рутдиалог. cs:

```

new OnIntent()
{
    Intent = "Cancel",
    Actions = new List<Dialog>()
    {
        new ConfirmInput()
        {
            Property = "turn.confirm",
            AllowInterruptions = false,
            Prompt = new ActivityTemplate("${RootCancelConfirm()}")
        },
        new IfCondition()
        {
            Condition = "turn.confirm == true",
            Actions = new List<Dialog>()
            {
                new SendActivity("${CancelReadBack()}"),
                new CancelAllDialogs()
            },
            ElseActions = new List<Dialog>()
            {
                new SendActivity("${Cancelcancelled()}")
            }
        }
    }
}

```

Ниже приведены три шаблона создания языка в **рутдиалог. lu**, которые вызываются из **ConfirmInput** действий или в **IfCondition** результате намерений "Cancel":

```

# RootCancelConfirm
- Are you sure you want to cancel?

# CancelReadBack
- Sure, cancelling all dialogs...

# Cancelcancelled
- No problem.

```

### **жетусерпрофилдиалог**

Диалоговое окно, определенное в **жетусерпрофилдиалог**, называется **userProfileDialog**. Это диалоговое окно вызывается корневым диалоговым окном в ответ на то, что пользователь вводит utterance, также известный как *фраза триггера*, связанный с **GetUserProfile** намерением, определенным в **RootDialog.lu**.

```

# GetUserProfile
- Hi
- My name is vishwac
- I'm 36 years old
- Profile

```

**userProfileDialog** Диалоговое окно является единственным дочерним диалоговым окном в этом роботе, поэтому все глобальные прерывания будут переходят непосредственно в корневое диалоговое окно. В ходе быстрого поиска в файле **GetUserProfileDialog.lu** будет показано, что не определено ни *Справка*, ни *Отмена* определенных целей. Без адаптивного механизма консультации по диалоговому окну обработка этих прерываний в **userProfileDialog** будет значительно сложнее, но из-за усилий, которые потребуются в пакете SDK для Bot-файла, эти прерывания просты в обработке. Все действия, определенные в этом диалоговом окне с **AllowInterruptions** установленным свойством или результатом

вычисления, позволяет `true` обрабатывать эти прерывания с помощью любого из его родительских диалоговых окон, в данном случае корневого диалогового окна.

Также определены два локальных прерывания. Они определяются как `OnIntent` действия, которые обрабатывают *смысл причин и значений*. Вы можете открыть файл  `GetUserProfileDialog.lu` для просмотра, добавления или обновления фразы продолжительностью, связанного с этими намерениями. Эти намерения не являются типичной частью обычного потокового потока, поэтому они обрабатываются как локальные прерывания. После завершения этих прерываний управление возвращается к действию, которое было прервано для беспрепятственного продолжения потоковой передачи, где она была остановлена.

#### Распознаватель Жетусерпрофиледиалог

Настройка распознавателя является первым делом, что происходит при  `userProfileDialog` создании. Каждое диалоговое окно настраивает собственный распознаватель независимо от всех других диалоговых окон, и каждое диалоговое окно может использовать один и тот же тип распознавателя, или каждый из них может использовать другой тип распознавателя. Каждое диалоговое окно в Bot может использовать любой тип распознавателя, определенный в пакете SDK для Bot Framework, независимо от того, какое диалоговое окно используется. Дополнительные сведения о различных доступных типах см. в разделе [типы распознавателей](#) статьи концепция распознавателей.

Как упоминалось ранее, каждое Адаптивное диалоговое окно имеет собственный распознаватель, и связанный с ним файл. `.lu` привязан к этому диалоговому окну. В файле. `.lu` определяются способы, [фразы продолжительностью и сущности](#), которые будут использоваться в этом диалоговом окне. Если пользователь вводит намерение, которое не определено в файле. `.lu` этого диалогового окна, механизм консультации по адаптивному диалогу позволяет объекту-роботу воссоздать в родительском диалоговом окне, если это возможно. В этом случае *методы справки и отмены* определяются в корневом диалоговом окне, но программа-робот может продолжать обработку этих целей, даже если  `userProfileDialog` активен диалог. Это подробно объясняется в более позднем разделе, озаглавленном [Жетусерпрофиледиалог Triggers](#).

В большинстве случаев код в Жетусерпрофиледиалог, используемый для определения распознавателя, аналогичен коду, используемому в [корневом диалоговом окне](#), единственное различие заключается в том, что необходимо ссылаться  `GetUserProfileDialog_en_us_lu` для значения, в  `ApplicationId` отличие от  `RootDialog_en_us_lu`, в  `createLuisRecognizer` методе. Дополнительные сведения о файле, на который указывает ссылка, см. в разделе [созданные файлы](#).

```
private static Recognizer CreateLuisRecognizer(IConfiguration configuration)
{
    if (string.IsNullOrEmpty(configuration["luis:GetUserProfileDialog_en_us_lu"]) ||
        string.IsNullOrEmpty(configuration["luis:LuisAPIKey"]) ||
        string.IsNullOrEmpty(configuration["luis:LuisAPIHostName"]))
    {
        throw new Exception("NOTE: LUIS is not configured for RootDialog. To enable all
capabilities, add 'LuisAppId-RootDialog', 'LuisAPIKey' and 'LuisAPIHostName' to the appsettings.json
file.");
    }

    return new LuisAdaptiveRecognizer()
    {
        ApplicationId = configuration["luis:GetUserProfileDialog_en_us_lu"],
        EndpointKey = configuration["luis:LuisAPIKey"],
        Endpoint = configuration["luis:LuisAPIHostName"]
    };
}
```

Генератор состоит из файла шаблона LG, который находится в том же каталоге, что и файл, содержащий исходный код адаптивного диалогового окна, с тем же именем и расширением файла LG, например **Жетусерпрофиледиалог. LG**.

```
_templates = Templates.ParseFile(Path.Combine(".", "Dialogs", "GetUserProfileDialog",
"GetUserProfileDialog.lg"));

// Create instance of adaptive dialog.
var userProfileDialog = new AdaptiveDialog(nameof(AdaptiveDialog))
{
    Recognizer = CreateLuisRecognizer(this.configuration),
    Generator = new TemplateEngineLanguageGenerator(_templates),
```

#### Триггеры Жетусерпрофиледиалог

При запуске этого диалогового окна его `OnBeginDialog` триггер выполняет следующие действия:

- Два `PropertyAssignment` действия выполняются при определении двух новых свойств: `user.profile.name` И `user.profile.age` .
- Эти свойства заполняются при наличии сущностей, например, когда это диалоговое окно было активировано фразой «Привет, | Вишвак». В этом случае `@Personname` сущность будет *вишвак*.
- Если какое-либо из свойств имеет значение null, программа Bot запрашивает у пользователя недостающие сведения с помощью действий *ввода текста*.
- Оба действия ввода допускают прерывания, используя [адаптивные выражения](#).

```

new SetProperties()
{
    Assignments = new List<PropertyAssignment>()
    {
        new PropertyAssignment()
        {
            Property = "user.profile.name",
            Value = "=coalesce(dialog.userName, @personName)"
        },
        new PropertyAssignment()
        {
            Property = "user.profile.age",
            Value = "=coalesce(dialog.userAge, @age)"
        }
    }
},
new TextInput()
{
    Property = "user.profile.name",
    Prompt = new ActivityTemplate("${AskFirstName()}"),
    Validations = new List<BoolExpression>()
    {
        "count(this.value) >= 3",
        "count(this.value) <= 50"
    },
    InvalidPrompt = new ActivityTemplate("${AskFirstName.Invalid()}"),
    Value = "@personName",
    AllowInterruptions = "turn.recognized.score >= 0.9 || !@personName"
},
new TextInput()
{
    Property = "user.profile.age",
    Prompt = new ActivityTemplate("${AskUserAge()}"),
    Validations = new List<BoolExpression>()
    {
        "int(this.value) >= 1",
        "int(this.value) <= 150"
    },
    InvalidPrompt = new ActivityTemplate("${AskUserAge.Invalid()}"),
    UnrecognizedPrompt = new ActivityTemplate("${AskUserAge.Unrecognized()}"),
    Value = "=coalesce(@age.number, @number)",
    AllowInterruptions = "!@age && !@number"
},

```

#### TIP

`AllowInterruptions` Свойство имеет значение `BoolExpression`. Логическое выражение — это свойство `адаптивикспрессионс`, которое является либо логическим, либо строковым выражением, которое разрешается в логическое значение. `AllowInterruptions = "turn.recognized.score >= 0.3 || !@personName"` присвойте `AllowInterruptions` свойству значение `true`, если `Оценка прогноза`, `ВОЗВРАЩЕННАЯ из Luis`, превышает 30% или сущность `персоннаме` имеет значение `null`, в противном случае возвращается значение `false`.

## Тестирование бота

- Установите `Bot Framework Emulator`, если вы этого еще не сделали.
- Выполните этот пример на локальном компьютере.
- Запустите эмулятор, подключитесь к боту и отправьте несколько сообщений, как показано ниже.

На снимке экрана ниже показано, что можно прерывать поток сообщений, запросив помошь, даже несмотря на то, что активное в настоящий момент Адаптивное диалоговое окно не содержит соответствующий триггер.

Profile

Just now

What is your first name?

Vishwac

Just now

Hi Vishwac, What is your age?

Help

Just now

I'm a sample interruptions bot

Hiya Vishwac, What is your age?

Why?

No age

Cancel



Type your message



# Создание бота с использованием декларативных адаптивных диалогов

27.03.2021 • 9 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье показано, как создать бот, реализующий **адаптивный диалог** с помощью декларативного подхода в пакете SDK для Bot Framework.

## Предварительные требования

- Понимание [основных принципов работы ботов, управления состоянием и библиотеки диалогов](#).
- Понимание принципов работы [адаптивных диалогов и декларативных диалогов](#).
- Знание основных принципов, необходимых для [создания бота с помощью адаптивных диалогов](#). Данная статья основывается на этих принципах.
- Копия примера декларативного [ечобот C#](#).

### Подготовка для добавления декларативного адаптивного диалога в бот

Чтобы загрузить и использовать декларативный диалог в боте, необходимо выполнить следующие действия.

1. Обновите все пакеты NuGet для Bot Builder до версии 4.9.x.
2. Создайте декларативные файлы.
3. Добавьте необходимые пакеты и ссылки в проект бота.
4. Зарегистрируйте декларативные компоненты Bot Framework для адаптивных диалогов.
5. Настройте в коде бота запуск или продолжение корневого диалога на каждом шаге с использованием диспетчера диалогов.
6. Создайте обозреватель ресурсов в коде бота для загрузки декларативных ресурсов.

## Сведения о примере

В этом примере бот EchoBot Bot Framework демонстрирует использование определенного декларативно [адаптивного диалога](#), который принимает входные данные от пользователя и возвращает их обратно.

## Создание декларативных файлов

Файлы декларативного диалога — это независимые от языка JSON-файлы, в которых объявляются элементы диалога. Это значит, что они одинаковы, независимо от языка, используемого для создания бота. Обычно они имеют расширение `.dialog`. Пример **ечобот** содержит только одно Адаптивное диалоговое окно с триггером для обработчика `UnknownIntent` события, которое при его срабатывании отправляет пользователю сообщение, которое сообщает о том, что говорят: `"$ {Turn.Activity.Text}"`.

### TIP

[Bot Framework Composer](#) — это интегрированное средство разработки, которое разработчики и многопрофильные команды могут использовать для создания ботов. Боты, созданные с помощью Bot Framework Composer, основаны на декларативном подходе.

Декларативный файл для примера **echoBot** :

```
{  
  "$schema": "../app.schema",  
  "$kind": "Microsoft.AdaptiveDialog",  
  "triggers": [  
    {  
      "$kind": "Microsoft.OnUnknownIntent",  
      "actions": [  
        {  
          "$kind": "Microsoft.SendActivity",  
          "activity": "You said '${turn.activity.text}'"  
        }  
      ]  
    }  
  ]  
}
```

#### NOTE

Как правило, рекомендуется хранить декларативные файлы во вложенной папке `dialogs`, однако они могут располагаться в любом месте. Их можно загрузить в бот с помощью метода `ResourceExplorer.GetResources()`.

#### Создание файла схемы

Файл схемы, на который ссылается ключевое слово `"$schema"`, содержит схемы всех компонентов, используемых ботом. Сведения о ключевом слове `"$schema"` и файле схемы, указанном в файлах с расширением `.dialog`, см. в статье [Использование декларативных ресурсов в адаптивных диалогах](#).

Чтобы создать файл схемы, указываемый в файле с расширением `.dialog`, необходим [интерфейс командной строки Bot Framework \(BF CLI\)](#). Если вы еще не установили его, можно установить BF CLI из командной строки.

```
npm i -g @microsoft/botframework-cli
```

С помощью интерфейса командной строки Bot Framework (BF CLI) в командной строке из корневого каталога проекта выполните команду `bf dialog:merge <filename.csproj>`.

```
bf dialog:merge EchoBot.csproj
```

В том же каталоге, в котором была выполнена команда, будет создан файл `app.schema`.

#### TIP

Если ключевое слово `"$schema"` отсутствует или указывает на недопустимый или несуществующий файл, предупреждения или ошибки не возникнут и это не повлияет на функциональность бота во время выполнения. Однако допустимый файл `app.schema` необходим инструментам [интеллектуального завершения кода](#), таким как [IntelliSense](#), для работы с любыми декларативными ресурсами.

## Добавление ссылок на декларативные компоненты

Декларативный подход работает только с адаптивными диалогами. Чтобы использовать адаптивный подход в боте, установите пакет NuGet `Microsoft.Bot.Builder.Dialogs.Adaptive`, а затем создайте приведенные ниже ссылки в коде `Startup.cs`.

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.BotFramework;
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Builder.Dialogs.Adaptive;
using Microsoft.Bot.Builder.Dialogs.Declarative;
using Microsoft.Bot.Builder.Dialogs.Declarative.Resources;
using Microsoft.Bot.Builder.Integration.AspNet.Core;
using Microsoft.Bot.Connector.Authentication;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
```

И добавьте их в файл EchoBot.cs.

```
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Builder.Dialogs.Adaptive;
using Microsoft.Bot.Builder.Dialogs.Declarative.Resources;
```

## Регистрация декларативных компонентов для адаптивных диалогов

```

// This method gets called by the runtime. Use this method to add services to the container.
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();

    // Register dialog. This sets up memory paths for adaptive.
    ComponentRegistration.Add(new DialogsComponentRegistration());

    // Register adaptive component
    ComponentRegistration.Add(new AdaptiveComponentRegistration());

    // Register to use language generation.
    ComponentRegistration.Add(new LanguageGenerationComponentRegistration());

    // Register declarative components for adaptive dialogs.
    ComponentRegistration.Add(new DeclarativeComponentRegistration());

    // Create the credential provider to be used with the Bot Framework Adapter.
    services.AddSingleton<ICredentialProvider, ConfigurationCredentialProvider>();

    // Create the Bot Framework Adapter with error handling enabled.
    services.AddSingleton<IBotFrameworkHttpAdapter, EchoBotAdapter>();

    // Create the storage we'll be using for User and Conversation state. (Memory is great for testing purposes.)
    services.AddSingleton<IStorage, MemoryStorage>();

    // Create the User state. (Used in this bot's Dialog implementation.)
    services.AddSingleton<UserState>();

    // Create the Conversation state. (Used by the Dialog system itself.)
    services.AddSingleton<ConversationState>();

    var resourceExplorer = new ResourceExplorer().LoadProject(this.HostingEnvironment.ContentRootPath);
    services.AddSingleton(resourceExplorer);

    // Create the bot In this case the ASP Controller is expecting an IBot.
    services.AddSingleton<IBot, EchoBot>();
}

```

## Декларативное создание диалога

Декларативные диалоги не являются обычными файлами кода. Обозреватель ресурсов может интерпретировать ресурс и создать экземпляр описанного диалога. Используйте обозреватель ресурсов для их загрузки во время выполнения.

Создайте свойства для `ResourceExplorer` И `DialogManager`, которые будет использовать бот. Используйте внедрение зависимостей, чтобы задать обозреватель ресурсов в конструкторе `EchoBot`.

```

public class EchoBot : ActivityHandler
{
    private IStatePropertyAccessor<DialogState> dialogStateAccessor;
    private readonly ResourceExplorer resourceExplorer;
    private DialogManager dialogManager;

    public EchoBot(ConversationState conversationState, ResourceExplorer resourceExplorer)
    {
        this.dialogStateAccessor = conversationState.CreateProperty<DialogState>("RootDialogState");
        this.resourceExplorer = resourceExplorer;
    }
}

```

При необходимости можно задать обновление диалога с новыми параметрами при каждом изменении базового файла с расширением `.dialog` с помощью метода `resourceExplorer.Changed`.

```
// auto reload dialogs when file changes
this.resourceExplorer.Changed += (e, resources) =>
{
    if (resources.Any(resource => resource.Id.EndsWith(".dialog")))
    {
        Task.Run(() => this.LoadRootDialogAsync());
    }
};
```

Наконец, следует создать декларативный ресурс диалога, который загружается в диспетчер диалогов, как и любой другой адаптивной диалог.

```
var resource = this.resourceExplorer.GetResource("main.dialog");
dialogManager = new DialogManager(resourceExplorer.LoadType<AdaptiveDialog>(resource));
dialogManager.UseResourceExplorer(resourceExplorer);
dialogManager.UseLanguageGeneration();
```

Метод `GetResource` обозревателя ресурсов считывает декларативный файл в объект ресурса, а метод `LoadType` приводит ресурс к объекту `AdaptiveDialog`. В нашем случае это файл `main.dialog`. Затем можно создать диспетчер диалогов так же, как для любого другого адаптивного диалога. Диалог считывается и создается из декларативного файла, а не определяется в коде.

Метод `UseResourceExplorer` диспетчера диалогов регистрирует обозреватель ресурсов, чтобы диспетчер диалогов мог использовать его позже при необходимости. Метод `UseLanguageGeneration` сообщает диспетчеру диалогов, какой генератор речи использовать.

В этом случае, так как файл шаблона для создания речи не указан, а этот проект не содержит стандартный файл `main.lg`, диспетчер диалогов будет использовать генератор речи по умолчанию без предопределенных шаблонов. Однако бот EchoBot включает в себя встроенный шаблон создания речи, используемый в диалоге. Он определен в файле `main.dialog`.

```
{
    "$schema": "../app.schema",
    "$kind": "Microsoft.AdaptiveDialog",
    "triggers": [
        {
            "$kind": "Microsoft.OnUnknownIntent",
            "actions": [
                {
                    "$kind": "Microsoft.SendActivity",
                    "activity": "You said '${turn.activity.text}'"
                }
            ]
        }
    ]
}
```

## Тестирование бота

- Установите [Bot Framework Emulator](#), если вы этого еще не сделали.
- Выполните этот пример на локальном компьютере.
- Запустите эмулятор, подключитесь к боту и отправьте несколько сообщений, как показано ниже.

hello

A minute ago

You said 'hello'

goodbye

A minute ago

You said 'goodbye'



Type your message



# Создание перекрестной обученной программы Bot для использования распознавателей LUIS и QnA Maker

27.03.2021 • 33 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье описывается процедура создания программы-робота, которая интегрирует возможности [Luis](#) и [QnA Maker](#) вместе в каждом адаптивном диалоговом окне, позволяя каждому определить наилучший ответ пользователя, используя [Создание языка](#), независимо от того, какое диалоговое окно активно.

Интеграция возможностей как LUIS, так и QnA Maker требует использования либо команды CLI, либо `luis:cross-train` `qnamaker:cross-train` для перекрестного обучения файлов. lu и. QnA, а также использования [набора перекрестных распознавателей](#) в качестве адаптивного распознавателя диалоговых окон.

## Предварительные условия

В этой статье рассматриваются шаги, необходимые для создания полнофункционального программы-робота с помощью адаптивных диалоговых окон, которые демонстрируются с помощью [Luis](#) и [QnA Maker](#) образце C# TODO Bot.

- [Общие сведения об адаптивных диалогах](#)
- [События и триггеры в адаптивных диалогах](#)
- [Действия в адаптивных диалогах](#)
- [Распознаватели в адаптивных диалогах](#)
- [Перекрестное обучение моделей LUIS и QnA Maker](#)
- [Управление прерываниями в адаптивных диалогах](#)
- [Формат файлов LU](#)
- [Формат файлов QNA](#)

Кроме того, вам потребуется [учетная запись Azure](#).

## Пример «TODO Bot with LUIS and QnA Maker»

В примере адаптивного TODO-**робота с Luis и QnA Maker (C#)** имеется пять. lu файлов и шесть файлов QnA для понимания языка. При создании перекрестно обученных языков можно объединить файлы. lu и. QnA, что позволит использовать возможности LUIS и QnA Maker вместе в одном распознавателе. Это один из подходов к обучению распознавателя, как лучше понять и ответить на запрос пользователя. После выполнения команды перекрестного обучения будут создаваться новые копии каждого файла, но они будут обновлены таким образом, чтобы файлы. lu содержали необходимые сведения о. QnA, а файлы. QnA содержат необходимые сведения. lu, включив [набор перекрестных распознавателей](#), чтобы определить, как лучше обрабатывать запросы пользователей.

Если вы еще не сделали этого, клонировать последнюю версию репозитория Samples. Вы можете использовать следующую команду Git из терминала:

```
git clone https://github.com/Microsoft/botbuilder-samples.git
```

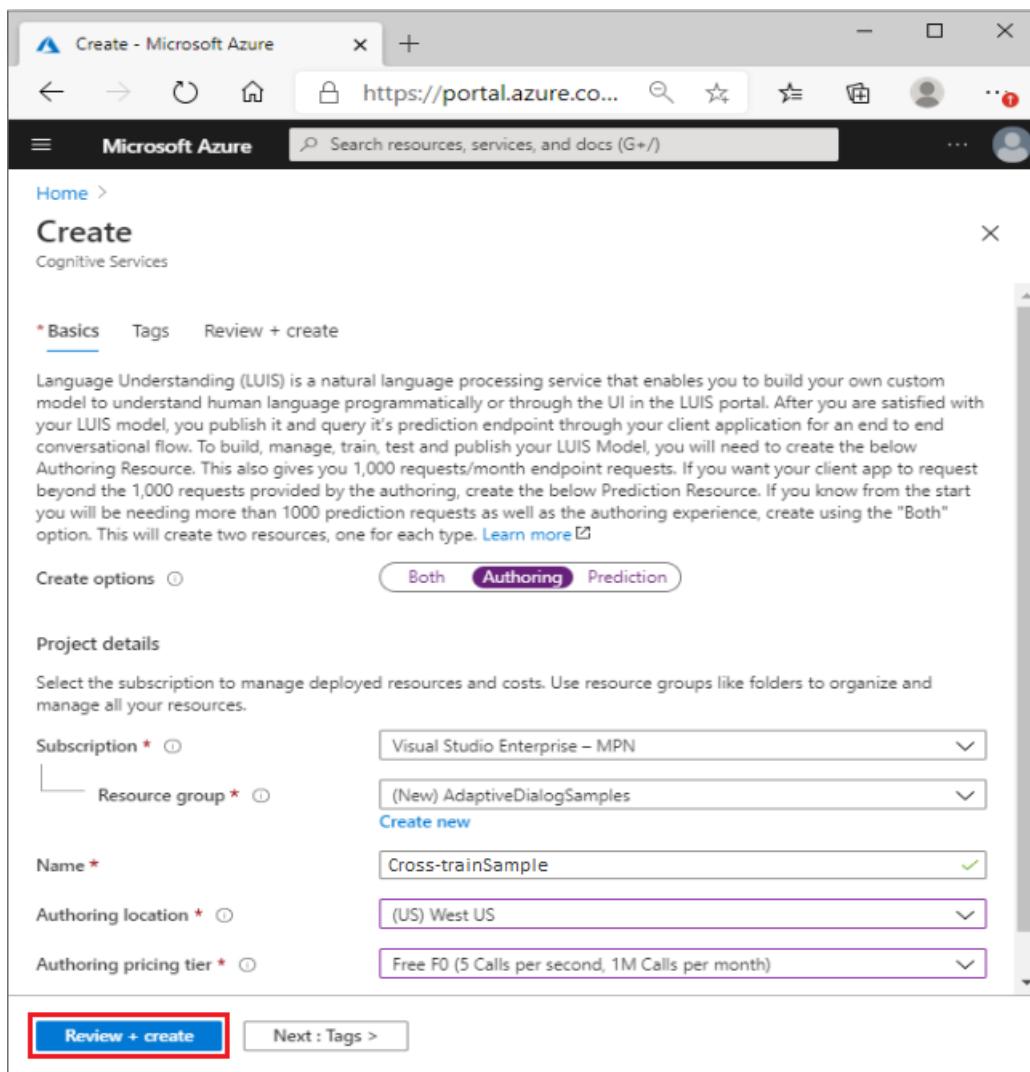
## Создание ресурса для создания LUIS в Azure

Если у вас еще нет ресурса для разработки LUIS, который вы хотите использовать, выполните следующие действия, чтобы создать его. Подробное описание этого процесса см. в статье [развертывание ресурсов Luis с помощью команд интерфейса командной строки Bot Framework Luis](#).

1. Перейдите на страницу [создания Cognitive Services Azure Luis](#).
2. В разделе **Параметры создания** выберите **Создание ресурса для создания Luis**.



3. Введите значения для каждого поля, а затем нажмите кнопку "проверить и создать".



4. Проверьте значения, чтобы убедиться, что они верны, а затем нажмите кнопку **создать**.

Ресурс разработки LUIS будет содержать сведения, необходимые при выполнении [команды сборки](#) или обновлении [файла конфигурации](#) программы-робота. После создания ресурса сведения, приведенные ниже, можно найти в колонке **ключи и конечная точка** в ресурсе разработки Luis. Скопируйте и сохраните значения для последующего использования.

- **Ключи.** Одно из двух ключевых значений используется в качестве параметра `subscriptionKey` в [команде сборки](#), а также в `luisapikey` в [файле конфигурации](#).

- **Конечная точка.** Это значение, которое используется в качестве параметра *конечной точки* в [команде сборки](#), а также *уисапихостнаме* в [файле конфигурации](#).
- **Расположение.** Это значение, которое используется в качестве параметра *Region* в [команде Build](#).

The screenshot shows the Azure Cognitive Services Keys and Endpoint page. The service name is 'Cross-trainSample'. The 'Keys and Endpoint' section is selected and highlighted with a red box. It displays two key values (KEY 1 and KEY 2) and their corresponding endpoint URLs. The endpoint URL is 'https://interruptionsbot-authoring.cognitiveservices.azure.com/'. The location is set to 'westus'.

## Создание ресурса QnA Maker в Azure Cognitive Services

Если у вас еще нет ресурса QnA Maker, можно выполнить следующие действия, чтобы создать его. Подробное описание этого процесса см. в статье [развертывание QnA Maker базы знаний с помощью команд интерфейса командной строки Bot Framework qnamaker](#).

1. Перейдите на страницу [Cognitive Services Azure Create QnA Maker](#).
2. Введите значения для каждого поля, а затем нажмите кнопку "**проверить и создать**".

Microsoft Azure  ...

Home > Create

## Create

QnA Maker

### Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription \*

Resource group \*  [Create new](#)

Name \*

Pricing tier \*

### Azure Search details - for data

When you create a QnAMaker resource, you host the data in your own Azure subscription. Azure Search is used to index your data.

Azure Search location \*

Azure Search pricing tier \*

### App Service details - for runtime

When you create a QnAMaker resource, you host the runtime in your own Azure subscription. App Service is the compute engine that runs the QnA Maker queries for you.

App name \*

Website location \*

### App insights details - for telemetry and chat logs

QnAMaker will optionally provision an instance of Application Insights and will appear in your Azure subscription. Telemetry and chatlogs will be stored here.

App insights

App insights location \*

3. Проверьте значения, чтобы убедиться, что они верны, а затем нажмите кнопку **создать**.

Ресурс QnA Maker будет содержать значение, необходимое для параметра *субскриптионкэй*, используемого при выполнении [команды сборки](#). Скопируйте и сохраните значение для последующего использования.

The screenshot shows the Microsoft Azure Cognitive Services QnAMaker Keys and Endpoint page. On the left, there's a sidebar with various options like Overview, Activity log, Access control (IAM), Tags, and Diagnose and solve problems. Below that is a RESOURCE MANAGEMENT section with Quick start, Keys and Endpoint (which is selected and highlighted with a red box), Pricing tier, Networking, Identity, Billing By Subscription, and Properties. The main content area has a heading 'Cross-trainSample | Keys and Endpoint' and a note about securely storing keys. It shows two key fields: 'KEY 1' (redacted) and 'KEY 2' (redacted). Below that is an 'ENDPOINT' field containing 'https://myqnamakerbot.cognitiveservices.azure.com/' and a 'LOCATION' field set to 'westus'. There are also 'Regenerate Key1' and 'Regenerate Key2' buttons at the top.

## Установка интерфейса командной строки для Bot Framework

Для работы с Bot Framework CLI требуется [Node.js](#).

1. Убедитесь, что установлена последняя версия NPM:

```
npm i -g npm
```

2. С помощью Node.js установите последнюю версию Bot Framework CLI из командной строки.

```
npm i -g @microsoft/botframework-cli
```

Дополнительные сведения см. в статье [Bot Framework CLI Tool](#).

## Создание перекрестно обученных моделей LU

Перед выполнением команды сборки для создания приложений LUIS и QnA Maker базы знаний в службе "Поиск в Azure" необходимо переучить файлы. lu и. QnA, чтобы включить сведения, необходимые распознавателю робота, чтобы отложить ввод пользователя на LUIS или QnA Maker для обработки. Чтобы лучше понять концепцию кросс-обучения, ознакомьтесь со статьей [перекрестное обучение программы-робота, чтобы использовать Luis и QnA Maker распознаватели](#). Дополнительные сведения о `cross-train` команде см. в разделе *перекрестно обученный набор распознавателей* в окне [справочника по адаптивным диалоговым окнам](#).

**Перекрестное обучение программы-робота Todo с LUIS и QnA Maker образцом**

Чтобы переучить программу-робот **Todo с Luis и QnA Maker** образцом:

#### IMPORTANT

Инструкции, приведенные в оставшейся части этой статьи, будут называться образцом кода, который вы получили в разделе [TODO Bot with Luis and QnA Maker](#) выше. После клонирования репозитория образец можно найти в .\Ботбуилдер-самплес\самплес\ csharp\_dotnetcore \adaptive-dialog\08.TODO-Bot-Luis-qnamaker Directory.

1. В окне терминала перейдите к каталогу 08.TODO-Bot-Luis-qnamaker .
2. Создайте каталог для хранения выходных данных команды. В этой статье *в качестве имени* для этого каталога используется.

```
md generated
```

3. Выполните команду `luis:cross-train`.

```
bf luis:cross-train -i dialogs -o generated --config dialogs\DialogLuHierarchy.config.json --force
```

После завершения работы будут выполнены перекрестные версии пяти файлов Lu и шести QnA-файлов. Здесь `--force` используется для принудительной перезаписи существующих файлов. Lu и. QnA, если они уже существуют. В противном случае, для файлов с расширением Lu, таких как `AddToDoDialog.lu`, перекрестное содержимое будет записываться в файл с именем **аддтододиалог (1). lu**. При выполнении команд сборки в следующих разделах укажите созданный каталог для входных файлов.

#### IMPORTANT

Необходимо выполнить `luis:cross-train` команду или `qnamaker:cross-train` команду; не нужно выполнять оба эти действия. При запуске `luis:cross-train` все необходимые сведения о перекрестной обучении будут включаться в результирующие файлы. Lu и. QnA.

## Создание и публикация приложений LUIS с помощью команды Build

Для каждого Lu-файла, включая файлы с расширением Lu для каждого языкового стандарта, команда Build объединяет все следующие действия в одну команду:

1. Создает одну модель LUIS для каждого языкового стандарта, найденного с помощью существующих файлов. Lu.
2. Используя эту модель, она создает новое (или обновляет существующее) приложение LUIS в указанном ресурсе Azure Cognitive Services.
  - При обновлении существующего приложения LUIS оно автоматически увеличивает значение `versionId` и при необходимости удаляет старую версию.
3. Выполните обучение нового или обновленного приложения LUIS, а затем опубликуйте его.

Подробное описание использования `luis:build` команды см. в статье [развертывание приложений Luis с помощью команд интерфейса командной строки Bot Framework Luis](#).

## IMPORTANT

Эта команда перезапишет предыдущую модель LUIS, а также содержимое, которое может присутствовать в [приложениях Luis](#).

## Использование команды Luis: Build

Вот команда LUIS Build с обязательными параметрами:

```
bf luis:build --in <input-file-or-folder> --out <output-file-or-folder> --botName <bot-name> --authoringKey <subscription-key> --region <authoring-region>
```

`luis:build` Команда создаст все необходимые ресурсы из локальных файлов. lu.

### Обязательный Luis: параметры сборки

- `--in` : Каталог, в котором будут искаться файлы. lu.
- `--out` : Каталог для сохранения выходных файлов.
- `--botName` — Имя программы-робота. Он будет использоваться в качестве префикса для имени создаваемых приложений LUIS.
- `--authoringKey` : Субскрипционкэй.
- `--region` — Регион для публикации приложений LUIS.

Дополнительные сведения о дополнительных параметрах см. в разделе [BF Luis: Build](#) статьи [readme The Bot Framework CLI](#).

Кроме того, можно включить эти обязательные данные, а также любые другие параметры в файле конфигурации и обращаться к ним с помощью `--luConfig` параметра.

### Файл конфигурации сборки LUIS

Ниже приведен пример `luconfig.json` в файле, на который можно сослаться с помощью `--luConfig` параметра.

```
{  
    "in": "generated",  
    "out": "output",  
    "botName": "<your-bot-name>",  
    "authoringKey": "<your-32-digit-subscription-key>",  
    "endpoint": "<your-endpoint>",  
    "region": "<your-region-default-is-westus>"  
}
```

После создания этого файла конфигурации необходимо просто сослаться на него в `luis:build` команде.  
Пример:

```
bf luis:build --luConfig luconfig.json
```

Подробное описание использования `luis:build` команды см. в статье [развертывание приложений Luis с помощью команд интерфейса командной строки Bot Framework Luis](#).

### Создание LUIS приложений для программы-робота Todo с LUIS и QnA Maker образцом

Чтобы создать LUIS приложения для программы-робота Todo с Luis и QnA Maker образцом:

1. В окне терминала перейдите к каталогу 08. TODO-Bot-Luis-qnamaker .
2. Создайте каталог для хранения выходных данных команды. В этой статье в качестве имени для

этого каталога используется Output .

```
md output
```

3. Создайте `luconfig.js` для файла, как показано в предыдущем разделе, и сохраните его в корневом каталоге примера. Убедитесь, что вы обновляете записи для параметров `botname`, `authoringkey` и `Region`:

#### 08.todo-bot-luis-qnamaker\luconfig.js на

```
{
  "in": "generated",
  "out": "output",
  "botName": "todo bot with LUIS and QnA Maker",
  "authoringKey": "<your-32-digit-subscription-key>",
  "endpoint": "<your-endpoint>",
  "region": "<your-region-default-is-westus>"
}
```

4. Выполните `luis:build` команду, используя `luconfig.js`.

```
bf luis:build --luConfig luconfig.json
```

#### TIP

Если вы еще не сделали этого, вам потребуется [Перейти на ключ создания ресурсов Azure](#). В противном случае вы не увидите приложения LUIS в [Luis](#), созданные с помощью `luis:build` команды.

После завершения работы у вас будет приложение LUIS для каждого из пяти файлов. Iu в [Luis](#):

App name	App type	Modified on	Culture
<a href="#">Todo bot with LUIS and QnA Maker(YourUserName)-RootDialog.en-us.luis</a>	Conversation	Mon Oct 05 2020	en-us
<a href="#">Todo bot with LUIS and QnA Maker(YourUserName)- GetUserProfileDialog.en-us.luis</a>	Conversation	Mon Oct 05 2020	en-us
<a href="#">Todo bot with LUIS and QnA Maker(YourUserName)-DeleteToDoDialog.en-us.luis</a>	Conversation	Mon Oct 05 2020	en-us
<a href="#">Todo bot with LUIS and QnA Maker(YourUserName)-ViewToDoDialog.en-us.luis</a>	Conversation	Mon Oct 05 2020	en-us
<a href="#">Todo bot with LUIS and QnA Maker(YourUserName)-AddToDoDialog.en-us.luis</a>	Conversation	Mon Oct 05 2020	en-us

**TIP**

`luis:build` Команда сохранит файл с именем `luis.settings.YourUserName.westus.json` в выходном каталоге.

Этот файл содержит идентификаторы приложений LUIS, которые необходимо добавить в файл конфигурации, описанные в разделе [Обновление файла конфигурации проекта](#), чтобы включить сведения о подключении для [Luis](#) и [QnA Maker](#).

## Создание и публикация баз знаний QnA Maker с помощью команды Build

`qnamaker:build` Команда объединяет все следующие действия в одну команду:

1. Создает одну модель QnA Maker для каждого языкового стандарта, найденного с помощью существующих QnAных файлов.
2. Создает новую QnA Maker КБ, если она не существует, в противном случае перезапишет существующую базу знаний.
3. Перед об обучением QnA Maker КБ публикует его в конечной точке производства.

Подробное описание использования `qnamaker:build` команды см. в разделе [развертывание QnA Maker базы знаний с помощью команд интерфейса командной строки Bot Framework qnamaker](#).

### Использование команды `qnamaker:Build`

Ниже приведена команда QnA Maker Build с обязательными параметрами:

```
bf qnamaker:build --in <input-file-or-folder> --out <folder-to-save-files-to> --subscriptionKey  
<Subscription-Key> --botName <bot-name>
```

**IMPORTANT**

Эта команда перезапишет предыдущую модель QnA Maker, а также любое содержимое, которое может присутствовать в QnA Maker КБ, включая все содержимое, созданное непосредственно в [QnA Maker AI](#).

#### Qnamaker: параметры сборки

- `--in` : Каталог, включая подкаталоги, в котором будут искаться QnA-файлы.
- `--out` : Каталог для сохранения выходных файлов.
- `--botName` — Имя программы-робота. Он будет использоваться для создания имени QnA Maker КБ. Это объясняется более подробно в статье [развертывание QnA Maker базы знаний с помощью команд интерфейса командной строки Bot Framework qnamaker](#) .
- `--subscriptionKey` : Тот же ключ подписки, который находится в [файле инициализации](#).

Дополнительные сведения о дополнительных параметрах см. в разделе [BF qnamaker: Build](#) статьи The Bot Framework CLI `qnamaker readme`.

Кроме того, можно включить эти обязательные параметры в файл конфигурации и предоставить их с помощью `qnaConfig` параметра.

#### Файл конфигурации `qnamaker`

Файл конфигурации `qnamaker` — это JSON-файл, который может содержать любой допустимый `qnamaker:build` параметр.

```
{  
  "in": "<location-of-qna-files>",  
  "out": "<location-to-save-output-files>",  
  "subscriptionKey": "<Enter-subscription-key-here>",  
  "botName": "<Enter-botName-here>"  
}
```

После создания нужно просто ссылаться на него в `qnamaker:build` команде, например:

```
bf qnamaker:build --qnaConfig qnaConfig.json
```

Подробное описание использования `qnamaker:build` команды см. в разделе [развертывание QnA Maker базы знаний с помощью команд интерфейса командной строки Bot Framework qnamaker](#).

**Создание базы знаний QnA Maker для программы-робота Todo с помощью LUIS и QnA Maker Sample**

Чтобы создать базу знаний QnA Maker для программы- **робота Todo с Luis и QnA Maker Sample**:

1. В окне терминала перейдите к каталогу `08.TODO-Bot-Luis-qnamaker` .
2. Создайте новый каталог с именем `Output` , если он еще не создан.

```
md output
```

3. Создайте `qnaConfig.json` для файла, как показано в предыдущем разделе, и сохраните его в каталоге `08.TODO-Bot-Luis-qnamaker` . Убедитесь, что вы обновляете записи для параметров *ботнаме*, *субскриптионкэй* и *Region*:

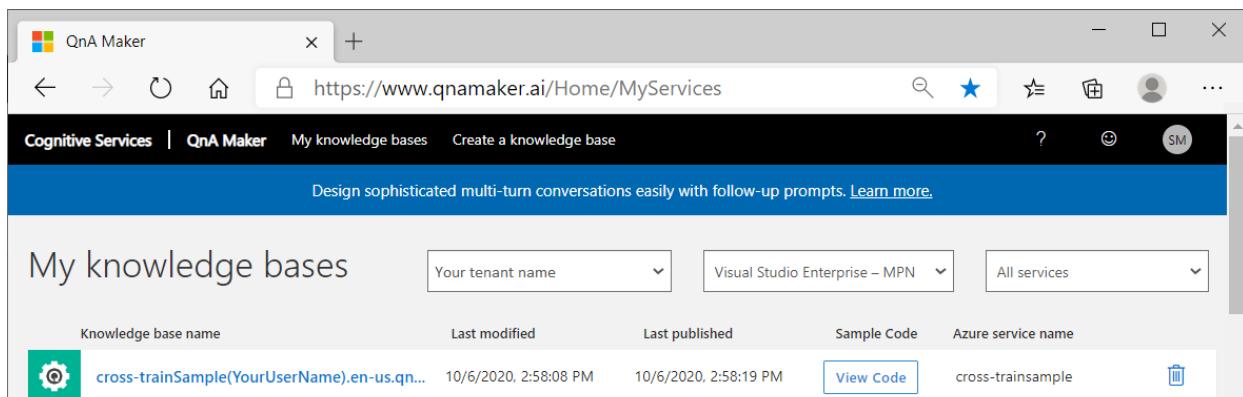
#### 08.todo-bot-luis-qnamaker\qnaConfig.json

```
{  
  "in": "generated",  
  "out": "output",  
  "botName": "<todo bot with LUIS and QnA Maker>",  
  "subscriptionKey": "<your-32-digit-subscription-key>",  
  "region": "<your-region-default-is-westus>"  
}
```

4. Выполните `qnamaker:build` команду, используя `qnaConfig.json`.

```
bf qnamaker:build --qnaConfig qnaConfig.json
```

После завершения вы получите QnA Maker базу знаний со всеми вопросами и ответами из пяти файлов QnA в [QnA Maker](#):



**TIP**

`qnamaker:build` Команда сохранит файл с именем `qnamaker.settings.<your-user-name>.westus.json` в выходном каталоге. Этот файл содержит идентификатор базы знаний, который необходимо добавить в файл конфигурации, как описано в следующем разделе.

## Обновите файл конфигурации проекта, чтобы включить сведения о подключении для LUIS и QnA Maker

Имя файла конфигурации — `appsettings.js`. Ниже показан файл конфигурации для программы-робота Todo с LUIS и QnA Maker образцом:

```
{  
    // The first two values are used to connect to your bot channel in Azure.  
    // See the "Bot channels registration" section below for more information.  
    "MicrosoftAppId": "",  
    "MicrosoftAppPassword": "",  
    // The next two values are used to connect to your Azure cognitive services LUIS  
    // authoring resource. See the "LUIS key and hostname" section below for more information.  
    "LuisAPIKey": "",  
    "LuisAPIHostName": "",  
    // The luis section contains all luis application IDs, created by the luis build  
    // command and saved to the specified output directory in the file named:  
    // "luis.settings.<username>.<authoring-region>.json"  
    // See "LUIS application IDs" section below for more details.  
    "luis": {  
        " GetUserProfileDialog_en_us_lu": "",  
        " ViewToDoDialog_en_us_lu": "",  
        " DeleteToDoDialog_en_us_lu": "",  
        " RootDialog_en_us_lu": "",  
        " AddToDoDialog_en_us_lu": ""  
    },  
    // The next two values are used to connect to your Azure cognitive services QnA Maker  
    // resource. See "QnA Maker hostname and endpoint key" section below for more details.  
    "QnAHostName": "",  
    "QnAEndpointKey": "",  
    // The qna section contains all QnA Maker knowledge base IDs, created from the .qna files by  
    // the qnamaker build command and saved to the specified output directory in the file named:  
    // "qnamaker.settings.<username>.<authoring-region>.json"  
    // See "QnA Maker knowledge base IDs" section below for more details.  
    "qna": {  
        " TodoBotWithLuisAndQnA_en_us_qna": ""  
    }  
}
```

### Сведения о файле конфигурации

В этом разделе подробно описывается `appsettings.js` файла для программы-робота Todo с LUIS и QnA Maker Sample.

#### Регистрация каналов бота

Дополнительные сведения о получении значений и при необходимости см. в статье [Регистрация каналов Bot](#) `MicrosoftAppId` `MicrosoftAppPassword`. Однако эти значения не являются обязательными для выполнения этой статьи.

#### Ключ LUIS и имя узла

Лuisапикэй — `subscriptionKey`, а лuisапихостнаме — это `ENDPOINT` значение. Оба значения находятся в колонке **ключи и конечная точка** на странице ресурсов Luis Authoring Services Azure, как показано на снимке экрана ниже.

The screenshot shows the Microsoft Azure Cognitive Services interface for a service named 'Cross-trainSample'. The left sidebar has a 'Keys and Endpoint' section selected, which is highlighted with a red box. The main content area displays two key entries: 'KEY 1' and 'KEY 2', each with a copy icon. Below them is the 'ENDPOINT' field containing the URL 'https://interruptionsbot-authoring.cognitiveservices.azure.com/'. This URL is also highlighted with a red box. At the bottom, there is a 'LOCATION' field set to 'westus'. There are also 'Regenerate Key1' and 'Regenerate Key2' buttons at the top of the main content area.

### Идентификаторы приложений LUIS

Раздел *Luis* содержит все идентификаторы приложений Luis, используемые программой Bot. Эти значения можно найти на странице **параметров приложения** для приложения LUIS в [www.Luis.Al](http://www.Luis.Al), однако они также перечислены в файле параметров, созданном `luis:build` командой, и сохраняются в расположении, указанном в качестве `--out` параметра. Этот файл параметров содержит список всех ИДЕНТИФИКАТОРов приложений LUIS, которые были созданы для каждого языкового стандарта.

Полное имя этого JSON-файла — `Luis.Settings..<Authoring-Region # C1.json`. Например, если имя пользователя, выполнившего вход, — `юуританака` и вы используете `westus` область разработки, ваше имя файла будет `luis.settings.YuuriTanaka.westus.json`. Здесь вы найдете все значения для раздела `/luis/appsettings.json` в файле.

### QnA Maker имя узла и ключ конечной точки

`QnAHostName` — Это **значение узла**, `QnAEndpointKey` которое является значением **ендпоинткэй** в QnA Maker, доступ к которому можно получить, нажав кнопку **Просмотреть код** на странице **Мои базы знаний**, как показано на снимке экрана ниже.

The screenshot shows a 'Sample HTTP Request' dialog from the Postman application. It displays a POST request to the URL 'https://microsoftdataviz101.azuredashboards.net/qnamaker/generateAnswer'. The request headers include 'Host: https://microsoftdataviz101.azuredashboards.net/qnamaker' and 'Authorization: EndpointKey [REDACTED]'. The request body is a JSON object: {"question": "<Your question>"}.

### QnA Maker идентификаторы базы знаний

`qnamaker:build` Команда сохранит файл параметров в расположении, указанном в качестве `--out` параметра. Этот файл содержит список всех QnA Maker ИДЕНТИФИКАТОРов баз знаний, созданных для каждого языкового стандарта. Полное имя этого JSON-файла — `qnamaker.settings.<username>.<authoring-region>.json`. Например, если имя пользователя, выполнившего вход, — `юуританака` и вы используете `westus` область разработки, ваше имя файла будет

`qnamaker.settings.YuuriTanaka.westus.json`. Здесь вы найдете все значения для `qna` раздела `appsettings.json` файле.

#### IMPORTANT

Файл параметров, созданный командой, `qnamaker:build` будет содержать запись для каждой из пяти QnA Maker моделей. Значение для каждого из них будет ИДЕНТИФИКАТОРом для одной QnA Maker КБ, созданной командой сборки. Поскольку каждый из них содержит одно и то же значение идентификатора, используйте любой из них для значения ключа `TodoBotWithLuisAndQnA_en_us_qna`. Если заменить это единственное значение на все пять значений из файла `qnamaker.Settings`, вы получите сообщение об ошибке: "System. Exception: NOTE: QnA Maker не настроен для Рутдиалог".

## Обновления исходного кода для перекрестно обученных моделей

В примере с LUIS и QnA Maker ([C#](#)) не требуется обновление исходного кода, чтобы воспользоваться преимуществами перекрестной обученной модели, так как пример был создан с учетом перекрестного обучения. В этом разделе описывается код в этом образце, относящийся к программы-роботы с использованием перекрестно обученных моделей, в качестве примера использования **аддтододиалог**. CS . Те же принципы применяются и к другим адаптивным диалогам в этой роботе.

### Определение распознавателя

Распознаватель, необходимый для работы с моделями, которые были перекрестно обучен, — это [кросstrainedреконизерсет](#).

В этом образце программы-робота распознаватель устанавливается путем вызова метода и передачи свойства в `configuration` качестве одного параметра:

```
Recognizer = CreateCrossTrainedRecognizer(configuration)
```

Метод `CreateCrossTrainedRecognizer` создает `CrossTrainedRecognizerSet` распознаватель, состоящий из списка, `Recognizers` содержащего распознаватель Luis и QnA Maker.

```
private static Recognizer CreateCrossTrainedRecognizer(IConfiguration configuration)
{
    return new CrossTrainedRecognizerSet()
    {
        Recognizers = new List<Recognizer>()
        {
            CreateLuisRecognizer(configuration),
            CreateQnAMakerRecognizer(configuration)
        }
    };
}
```

`CreateLuisRecognizer` Метод создает распознаватель Luis. Пояснения к коду см. в комментариях в следующем фрагменте кода:

```

public static Recognizer CreateLuisRecognizer(IConfiguration Configuration)
{
    // Verify that all required values exist in the configuration file appsettings.json and throw an error
    if (string.IsNullOrEmpty(Configuration["luis:RootDialog_en_us_lu"])) ||
        string.IsNullOrEmpty(Configuration["LuisAPIKey"]) || string.IsNullOrEmpty(Configuration["LuisAPIHostName"]))
    {
        throw new Exception("Your RootDialog LUIS application is not configured. Please see README.MD to set
up a LUIS application.");
    }
    return new LuisAdaptiveRecognizer()
    {
        // Get settings from the configuration file appsettings.json
        Endpoint = Configuration["LuisAPIHostName"],
        EndpointKey = Configuration["LuisAPIKey"],
        ApplicationId = Configuration["luis:RootDialog_en_us_lu"],

        // Id needs to be LUIS_<dialogName> for cross-trained recognizer to work.
        Id = $"LUIS_{nameof(RootDialog)}"
    };
}

```

Метод `CreateQnAMakerRecognizer` создает QnA Maker распознаватель. Пояснения к коду см. в комментариях в следующем фрагменте кода:

```

private static Recognizer CreateQnAMakerRecognizer(IConfiguration configuration)
{
    // Verify that all required values exist in the configuration file appsettings.json
    if (string.IsNullOrEmpty(configuration["qna:TodoBotWithLuisAndQnA_en_us_qna"])) ||
        string.IsNullOrEmpty(configuration["QnAHostName"]) || string.IsNullOrEmpty(configuration["QnAEndpointKey"]))
    {
        throw new Exception("NOTE: QnA Maker is not configured for RootDialog. Please follow instructions in
README.md. To enable all capabilities, add 'qnamaker:qnamakerSampleBot_en_us_qna', 'qnamaker:LuisAPIKey' and
'qnamaker:endpointKey' to the appsettings.json file.");
    }

    return new QnAMakerRecognizer()
    {
        // Get settings from the configuration file appsettings.json
        HostName = configuration["QnAHostName"],
        EndpointKey = configuration["QnAEndpointKey"],
        KnowledgeBaseId = configuration["qna:TodoBotWithLuisAndQnA_en_us_qna"],

        // property path that holds qna context
        Context = "dialog.qnaContext",

        // Property path where previous qna id is set. This is required to have multi-turn QnA working.
        QnAId = "turn.qnaIdFromPrompt",

        // Disable Personal Information telemetry logging
        LogPersonalInformation = false,

        // Enable to automatically including dialog name as meta data filter on calls to QnA Maker.
        IncludeDialogNameInMetadata = true,

        // Id needs to be QnA_<dialogName> for cross-trained recognizer to work.
        Id = $"QnA_{nameof(RootDialog)}"
    };
}

```

## Разрешение прерываний

Если ввод данных пользователем не приводит к совпадению распознавателя, перед перекрестным обучением моделей LUIS и QnA Maker, Bot автоматически отправит его родительскому элементу

диалогового окна, если `AllowInterruptions` свойство имеет значение *true*. При перекрестном обучении моделей активное диалоговое окно учитывает другие возможности обработки намеренных диалоговых окон, поэтому, если совпадений не возвращается, нет необходимости обращаться к родительскому диалоговому окну. В этом случае, как определить, должно ли активное диалоговое окно обрабатывать ответ пользователя или перепузырькить его к родительскому элементу? Рассмотрим этот сценарий, используя программу-робот Todo с LUIS и QnA Maker образцом:

#### NOTE

Вы можете следовать инструкциям, запустив Bot с помощью эмулятора. Инструкции для этого приведены в следующем разделе [тестирование программы-робота с помощью эмулятора Bot Framework](#).

- При запуске программы-робота пользователь сталкивается с *приветом, приятностью! Я использую пример программы-робота. Ниже приведены некоторые действия, с которыми можно помочь, а затем появится запрос с суждением = Добавить элемент | Просмотреть списки | Удалить элемент | Профиль | Отмена | Справка*
- Пользователь выбирает *список представлений*

на этом этапе **виевтододиалог** станет активным адаптивным диалоговым окном.

- Пользователю предлагается указать *, какой список вы хотите просмотреть?* и получить следующие три варианта: **TODO | Продуктов | Покупки | Все**
- Вместо выбора любого из представленных параметров пользователь вводит команду *Удалить TODO*.

Utterance *Удалить TODO* не принадлежит каким-либо методам в **виевтододиалог**, но поскольку модели были перекрестными обучены, Luis возвращает совпадение. Программа-робот просто должна уметь использовать механизм консультации для создания всплывающего запроса к **рутдиалог**, где этот utterance связан с намерением, которое приводит к вызову **делететододиалог**.

Ниже приведен код для запроса пользователя в **виевтододиалог**. LG:

```
new TextInput()
{
    Property = "dialog.listType",
    Prompt = new ActivityTemplate("${GetListType()}" ),
    Value = "@listType",
    AllowInterruptions = "!@listType && turn.recognized.score >= 0.7",
    Validations = new List<BoolExpression>()
    {
        // Verify using expressions that the value is one of todo or shopping or grocery
        "contains(createArray('todo', 'shopping', 'grocery', 'all'), toLower(this.value))",
    },
    OutputFormat = "=toLower(this.value)",
    InvalidPrompt = new ActivityTemplate("${GetListType.Invalid()}" ),
    MaxTurnCount = 2,
    DefaultValue = "todo",
    DefaultValueResponse = new ActivityTemplate("${GetListType.DefaultValueResponse()}" )
},
```

- `Prompt` Для этого `TextInput` вызывается `GetListType()` шаблон в **виевтододиалог**. LG.
- Значение, возвращаемое входными данными пользователя, сохраняется в `turn.recognized.entities.listType`. Краткая форма для `turn.recognized.entities.listType` — `@listType`
- Выражение для `AllowInterruptions` проверок `@listType`, которое будет существовать, если пользователь выбрал или указал допустимый тип списка. Если он не существует, он проверяет, является ли соответствие, возвращенное LUIS, значением прогноза 70% или более высокой

`turn.recognized.score >= 0.7`. Если это так, то это означает, что родительский или одноуровневый диалог имеет намерение с высокой оценкой прогноза. Это приводит `AllowInterruptions` к вычислению значения `true`, а пользователи `utterance` передаются в родительское диалоговое окно для обработки. Когда родительский диалог обрабатывает этот `utterance`, он находит совпадение в `DeleteItem` намерениях, что приводит к **делегированию диалога**.

#### NOTE

В статье [cross training of Luis and QnA Maker Models](#) описаны изменения, вносимые в файлы `.luis` и `.QnA`, когда они обучены, а в [таблице ответов распознавателя](#) в этой статье показаны все возможные отклики распознавателя и результирующее действие, предпринимаемое программой Bot.

## Тестирование бота с помощью Bot Framework Emulator

### Необходимые условия для тестирования программы-робота с помощью эмулятора Bot Framework

- [Visual Studio 2019](#) или более поздней версии или [Visual Studio Code](#)
- [.NET Core 3.1](#).
- Копия программы C# TODO Bot с [Luis](#) и [QnA Maker](#) образцом.
- [Bot Framework Emulator](#).

### Создание и запуск Bot в локальной среде

Чтобы запустить программу Bot локально, выполните команду, показанную ниже.

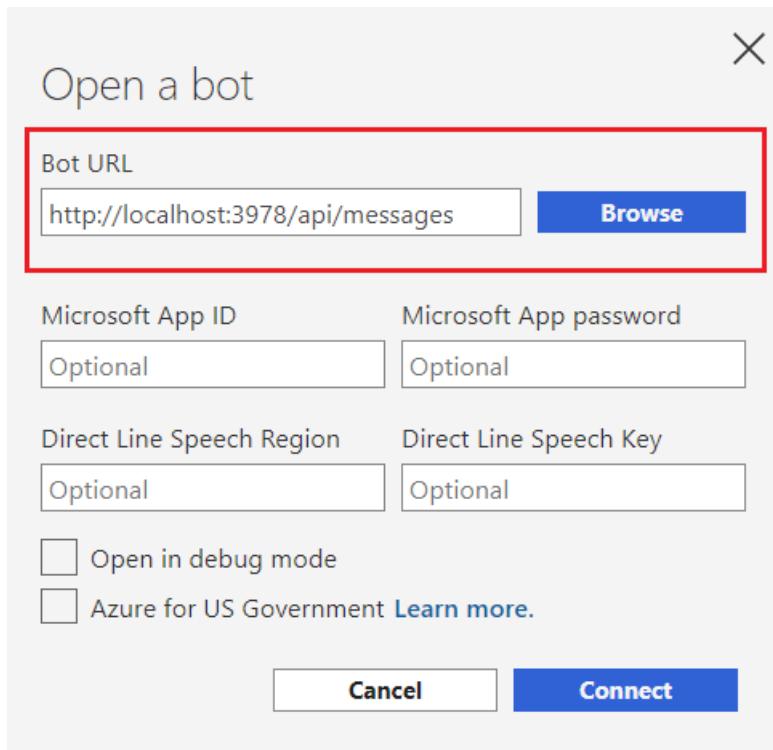
1. В окне терминала перейдите к каталогу `08. TODO-Bot-Luis-qnamaker`.
2. Запустите бот.

```
dotnet run
```

Это позволит выполнить сборку приложения, развернуть его на `localhost` и запустить веб-браузер для вывода страницы `default.htm` приложения. На этом этапе программа-робот должна выполняться локально через порт 3978, если только `applicationUrl` параметр в `launchSettings.js` не был изменен.

### Запустите эмулятор и подключите его к боту.

1. Установите Bot Framework Emulator.
2. Выберите **Открыть Bot** на вкладке **приветствия** эмулятора.
3. Введите URL-адрес робота, который является URL-адресом локального порта, с добавлением API/мессажес к пути, как правило, `http://localhost:3978/api/messages`.



4. В этом случае выберите **Подключиться**.

Теперь вы можете взаимодействовать с Bot.

# Запись данных напрямую в хранилище

27.03.2021 • 49 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Вы можете выполнять операции записи и чтения непосредственно в объекте хранилища, не используя ПО промежуточного слоя или объект контекста. Это может требоваться для данных, которые бот использует для сохранения беседы, или данных, которые поступают из источника за пределами потока общения бота. В рамках этой модели хранилища данныечитываются непосредственно из хранилища без использования диспетчера состояний. В примерах кода в этой статье показано, как выполнять чтение и запись данных в хранилище с помощью **памяти**, **Cosmos DB**, **большого двоичного объекта Azure** и хранилища записей **BLOB-объектов Azure**.

## Предварительные требования

- Если у вас еще нет подписки Azure, создайте [бесплатную](#) учетную запись Azure, прежде чем начинать работу.
- Знакомство с статьей [создание программы Bot локально для C#, JavaScript или Python](#).
- Шаблоны пакета SDK Bot Framework версии 4 для [Visual Studio \(C#\)](#), [Node.js](#) или [Yeoman](#).

### NOTE

Пакет [VSIX](#) включает версии .net Core 2.1 и .net Core 3.1 шаблонов C#. При создании ботов в Visual Studio 2019 следует использовать шаблоны .NET Core 3.1. В текущих примерах ботов используются шаблоны .NET Core 3.1. Примеры, использующие шаблоны .NET Core 2.1, можно найти в ветви [4.7-archive](#) репозитория BotBuilder-Samples. Сведения о развертывании .NET Core 3.1 программы-роботы в Azure см. в статье [развертывание программы Bot в Azure](#).

## Об этом примере

Пример кода в этой статье определяет структуру базового эхо-бота. Добавив дополнительный код (см. ниже), вы сможете расширить функции этого бота. Этот расширенный код создает список для хранения вводимых пользователем данных по мере получения. Каждый ход, полный список входных данных пользователя, сохраненный в памяти, возвращается пользователю. Структура данных, содержащая этот список входов, затем изменяется для сохранения в хранилище. Различные типы хранилища рассматриваются по мере добавления дополнительных функциональных возможностей в этот пример кода.

## Хранилище в памяти

Пакет SDK Bot Framework позволяет хранить вводимые пользователем данные с помощью размещенного в памяти хранилища. Поскольку хранилище в памяти удаляется при каждом перезапуске программы-робота, оно лучше всего подходит для тестирования и не предназначено для использования в рабочей среде. Постоянные хранилища, такие как хранилище базы данных, лучше всего подходят для ботов в рабочей среде.

## Создание базового бота

Остальная часть этой статьи описывает использование эхо-бота. Пример кода для программы Echo Bot

МОЖНО создать локально, следуя инструкциям краткого руководства по созданию echoBot в C#, JavaScript или Python.

- [C#](#)
- [JavaScript](#)
- [Python](#)

Замените код в EchoBot.cs следующим кодом:

```
using System;
using System.Threading.Tasks;
using Microsoft.Bot.Builder;
using Microsoft.Bot.Schema;
using System.Collections.Generic;
using System.Linq;
using System.Threading;

// Represents a bot saves and echoes back user input.
public class EchoBot : ActivityHandler
{
    // Create local Memory Storage.
    private static readonly MemoryStorage _myStorage = new MemoryStorage();

    // Create cancellation token (used by Async Write operation).
    public CancellationToken cancellationToken { get; private set; }

    // Class for storing a log of utterances (text of messages) as a list.
    public class UtteranceLog : IStoreItem
    {
        // A list of things that users have said to the bot
        public List<string> UtteranceList { get; } = new List<string>();

        // The number of conversational turns that have occurred
        public int TurnNumber { get; set; } = 0;

        // Create concurrency control where this is used.
        public string ETag { get; set; } = "*";
    }

    // Echo back user input.
    protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
    CancellationToken cancellationToken)
    {
        // preserve user input.
        var utterance = turnContext.Activity.Text;

        // Make empty local log-items list.
        UtteranceLog logItems = null;

        // See if there are previous messages saved in storage.
        try
        {
            string[] utteranceList = { "UtteranceLog" };
            logItems = _myStorage.ReadAsync<UtteranceLog>(utteranceList).Result?.FirstOrDefault().Value;
        }
        catch
        {
            // Inform the user an error occurred.
            await turnContext.SendActivityAsync("Sorry, something went wrong reading your stored messages!");
        }

        // If no stored messages were found, create and store a new entry.
        if (logItems is null)
        {
            // Add the current utterance to a new object.
            logItems = new UtteranceLog();
        }
    }
}
```

```

logItems.UtteranceList.Add(utterance);

// Set initial turn counter to 1.
logItems.TurnNumber++;

// Show user new user message.
await turnContext.SendActivityAsync($"{{logItems.TurnNumber}}: The list is now: {string.Join(", ", logItems.UtteranceList)}");

// Create dictionary object to hold received user messages.
var changes = new Dictionary<string, object>();
{
    changes.Add("UtteranceLog", logItems);
}
try
{
    // Save the user message to your Storage.
    await _myStorage.WriteAsync(changes, cancellationToken);
}
catch
{
    // Inform the user an error occurred.
    await turnContext.SendActivityAsync("Sorry, something went wrong storing your message!");
}
}

// Else, our storage already contained saved user messages, add new one to the list.
else
{
    // add new message to list of messages to display.
    logItems.UtteranceList.Add(utterance);
    // increment turn counter.
    logItems.TurnNumber++;

    // show user new list of saved messages.
    await turnContext.SendActivityAsync($"{{logItems.TurnNumber}}: The list is now: {string.Join(", ", logItems.UtteranceList)}");

    // Create Dictionary object to hold new list of messages.
    var changes = new Dictionary<string, object>();
    {
        changes.Add("UtteranceLog", logItems);
    };

    try
    {
        // Save new list to your Storage.
        await _myStorage.WriteAsync(changes, cancellationToken);
    }
    catch
    {
        // Inform the user an error occurred.
        await turnContext.SendActivityAsync("Sorry, something went wrong storing your message!");
    }
}
}
}

```

## Запуск бота

Запустите бот на локальном компьютере.

## Запуск эмулятора и подключение к боту

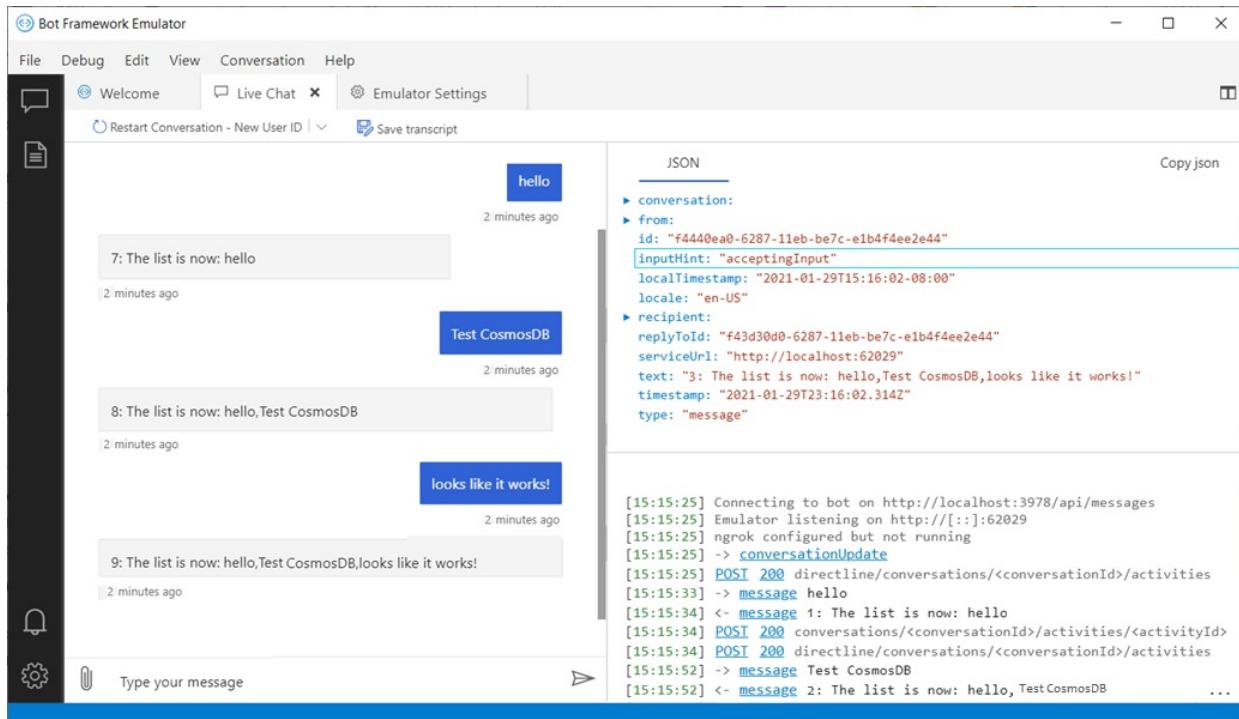
Установите [эмоджитор](#) Bot Framework, затем запустите эмулятор и подключитесь к компьютеру Bot в эмуляторе:

1. Щелкните ссылку **создать конфигурацию** Bot на вкладке **приветствия** эмулятора.

2. Заполните поля для подключения к боту, используя сведения на веб-странице, отображаемой при запуске бота.

## Взаимодействие с ботом

Отправьте сообщение боту. Он отобразит список полученных сообщений.



В оставшейся части этой статьи будет показано, как сохранить данные в постоянное хранилище вместо внутренней памяти Bot.

## Использование Cosmos DB

### IMPORTANT

Класс хранилища *Cosmos DB* является устаревшим. Контейнеры, изначально созданные с помощью Космосдбстораже, не имели набора ключей секций и получили ключ раздела по умолчанию `_ / partitionKey`.

Контейнеры, созданные с помощью хранилища *Cosmos DB*, можно использовать с *Cosmos DB секционированного хранилища*. См. сведения о [секционировании в Azure Cosmos DB](#).

Также обратите внимание, что, в отличие от устаревшего *Cosmos DB* хранения, *Cosmos DB* секционированного хранилища не создает базу данных в учетной записи *Cosmos DB* автоматически. Необходимо [создать новую базу данных вручную](#), но пропустить создание контейнера вручную, так как *космосдбартитионедстораже* создаст контейнер.

Начав использовать хранилище в памяти, мы изменим код, чтобы начать работу с *Azure Cosmos DB*. *Cosmos DB* — это глобально распределенная многомодельная база данных Майкрософт. *Azure Cosmos DB* позволяет гибко и независимо масштабировать пропускную способность и ресурсы хранилища в любом количестве регионов *Azure*. Она гарантирует пропускную способность, задержку, доступность и согласованность в соответствии с комплексными Соглашениями об уровне обслуживания (SLA).

### Настройка *Cosmos DB* ресурса

Чтобы использовать в боте *Cosmos DB*, необходимо создать ресурс базы данных, прежде чем приступить к написанию кода. Подробное описание *Cosmos DB* базы данных и создания приложений см. в кратком руководстве по [.NET](#), [Node.js](#) или [Python](#).

### Создание учетной записи базы данных

1. Создайте учетную запись для базы данных Azure Cosmos DB, [перейдя на портал Azure](#). Найдите в поиске и выберите Azure Cosmos DB .
2. На странице Azure Cosmos DB выберите **создать**, чтобы открыть страницу **Создание учетной записи** Azure Cosmos DB .

The screenshot shows the 'Create Azure Cosmos DB Account' wizard in progress. The 'Basics' tab is selected. The page includes a brief introduction about Azure Cosmos DB being a globally distributed, multi-model, fully managed database service. It also features a 'Project Details' section where users can choose a subscription and resource group, and an 'Instance Details' section where they can specify account name, API type (Core (SQL)), enable Notebooks (Preview), choose a location (US West US), and select capacity mode (Provisioned throughput). At the bottom, there are 'Review + create', 'Previous', and 'Next: Networking' buttons.

3. Укажите значения для следующих полей:
  - a. **Подписка**. Выберите подписку Azure, которую нужно использовать для этой учетной записи Azure Cosmos.
  - b. **Группа ресурсов**. Выберите существующую группу ресурсов или щелкните **создать** и введите имя для новой группы ресурсов.
  - c. **Имя учетной записи**. Введите имя для идентификации учетной записи Azure Cosmos. Так как элемент *documents.azure.com* добавляется к указанному вами имени для создания URI, используйте уникальное имя. Обратите внимание на следующие рекомендации.
    - Это имя должно быть уникальным в пределах Azure.
    - Длина имени должна составлять от 3 до 31 символов.
    - Имя может содержать только строчные буквы, цифры и символ дефиса (-).
  - d. API. Выбор **ядра (SQL)**
  - e. **Расположение**. выберите ближайшее к пользователям расположение, чтобы предоставить им самый быстрый доступ к данным.
4. Выберите **Review + Create** (Просмотреть и создать).
5. После проверки выберите **создать**.

Создание учетной записи займет несколько минут. Дождитесь, пока на портале откроется страница с

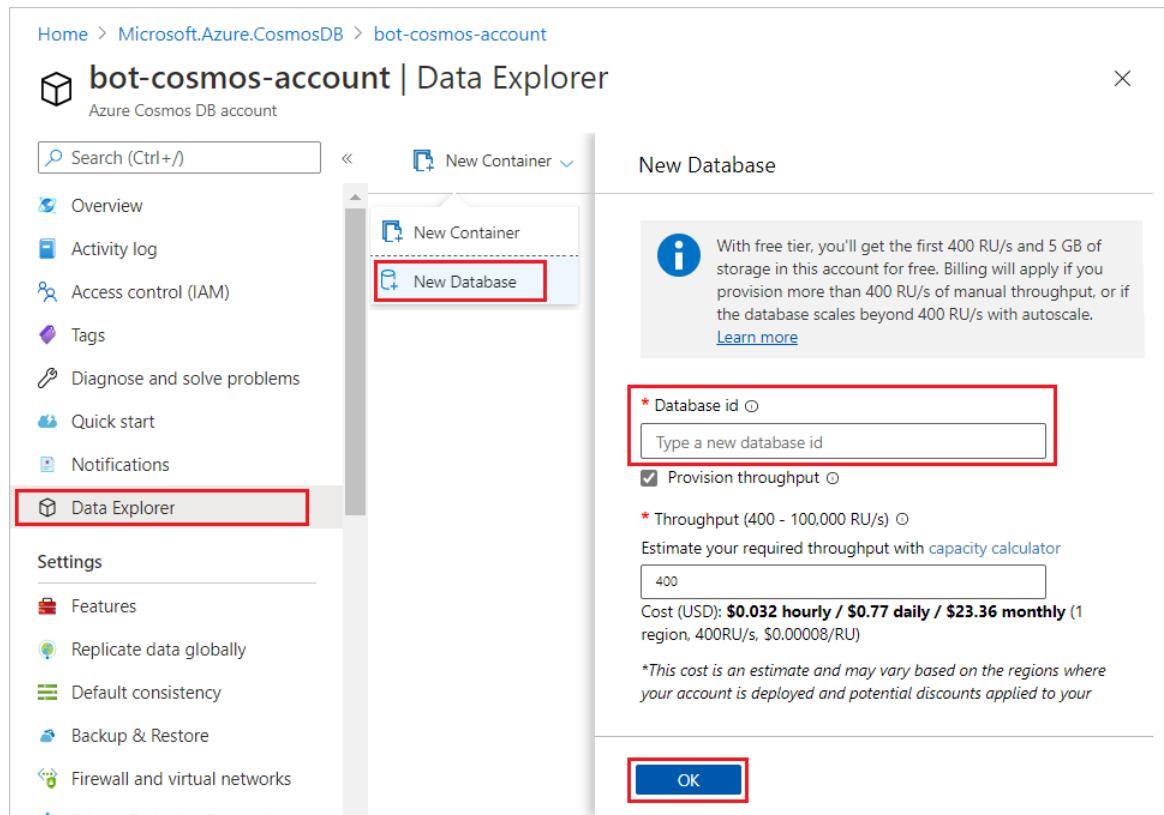
сообщением *Congratulations! Your Azure Cosmos DB account was created* (Поздравляем! Ваша учетная запись Azure Cosmos DB создана).

## Добавление базы данных

### NOTE

Не следует создавать контейнер самостоятельно. Ваш бот создаст его одновременно с внутренним клиентом Cosmos DB и правильно настроит для сохранения состояния бота.

- Перейдите на **Обозреватель данных** страницу в созданной учетной записи Cosmos DB, а затем выберите пункт **создать базу данных** в раскрывающемся списке **новый контейнер**. В правой части окна откроется панель, где можно указать сведения о новой базе данных.



- Ведите идентификатор для новой базы данных и, при необходимости, задайте пропускную способность (это можно будет изменить позже), а затем нажмите кнопку **OK**, чтобы создать базу данных. Запишите идентификатор базы данных, чтобы использовать его позже для настройки бота.
- Завершив создание учетной записи Cosmos DB и базы данных, скопируйте некоторые значения для интеграции новой базы данных в бота. Чтобы получить их, перейдите на вкладку **Ключи** в разделе параметров базы данных, которую вы создали в учетной записи Cosmos DB. На этой странице вам потребуется **URI** (*Cosmos DB конечной точки*) и **первичный ключ** (*ключ авторизации*).

Cosmos Db Endpoint value in your bot's configuration file

Cosmos Db Auth Key value in your bot's configuration file

Теперь у вас должна быть учетная запись Cosmos DB с базой данных и следующие значения, готовые к использованию в параметрах программы Bot.

- URI
- Первичный ключ
- Идентификатор базы данных

### Добавление сведений о конфигурации Cosmos DB

Используйте ранее сохраненные сведения, чтобы задать конечную точку, ключ авторизации и идентификатор базы данных. Наконец, выберите подходящее имя для контейнера, который будет создан в базе данных для хранения состояния бота. В примере ниже созданный контейнер Cosmos DB будет называться «Bot-Storage».

- C#
- JavaScript
- Python

Добавьте следующие сведения в файл конфигурации.

#### appsettings.json

```
"CosmosDbEndpoint": "<your-CosmosDb-URI>",
"CosmosDbAuthKey": "<your-primary-key>",
"CosmosDbDatabaseId": "<your-database-id>",
"CosmosDbContainerId": "bot-storage"
```

### Установка пакетов Cosmos DB

Убедитесь, что вы установили пакеты, необходимые для работы с Cosmos DB.

- C#
- JavaScript
- Python

Установите пакет NuGet Microsoft. Bot. Builder. Azure . Дополнительные сведения об использовании NuGet см. в статье [Установка пакетов и управление ими в Visual Studio с помощью диспетчера пакетов NuGet](#).

### Реализация Cosmos DB

## NOTE

В версии 4.6 появился новый поставщик хранилища Cosmos DB — класс *секционированного хранилища Cosmos DB* класса. Исходный класс *хранилища Cosmos DB* является устаревшим. Контейнеры, созданные с помощью *хранилища Cosmos DB*, можно использовать с *Cosmos DB секционированного хранилища*. См. сведения о [секционировании в Azure Cosmos DB](#).

Также обратите внимание, что, в отличие от устаревшего Cosmos DB хранения, Cosmos DB секционированного хранилища не создает базу данных в учетной записи Cosmos DB автоматически. Необходимо [создать новую базу данных вручную](#), но пропустить создание контейнера вручную, так как *космосдблартитионедстораже* создаст контейнер.

- [C#](#)
- [JavaScript](#)
- [Python](#)

Следующий пример кода выполняется с использованием того же кода Bot, что и в приведенном выше примере [хранилища памяти](#), за исключением перечисленных здесь исключений. Фрагменты кода ниже демонстрируют реализацию хранилища Cosmos DB для "myStorage", которое заменяет хранилище локальной памяти.

Сначала необходимо обновить `Startup.cs` для ссылки на библиотеку *Azure Builder*:

```
using Microsoft.Bot.Builder.Azure;
```

Затем в `ConfigureServices` методе в `Startup.cs` создайте `CosmosDbPartitionedStorage` объект. Он передается в `EchoBot` конструктор посредством внедрения зависимостей.

```
// Use partitioned CosmosDB for storage, instead of in-memory storage.  
services.AddSingleton<IStorage>(  
    new CosmosDbPartitionedStorage(  
        new CosmosDbPartitionedStorageOptions  
        {  
            CosmosDbEndpoint = Configuration.GetValue<string>("CosmosDbEndpoint"),  
            AuthKey = Configuration.GetValue<string>("CosmosDbAuthKey"),  
            DatabaseId = Configuration.GetValue<string>("CosmosDbDatabaseId"),  
            ContainerId = Configuration.GetValue<string>("CosmosDbContainerId"),  
            CompatibilityMode = false,  
        }));
```

В `EchoBot.cs` измените `_myStorage` объявление переменной

```
private static readonly MemoryStorage _myStorage = new MemoryStorage();
```

 следующим образом:

```
// variable used to save user input to CosmosDb Storage.  
private readonly IStorage _myStorage;
```

Затем передайте `IStorage` объект в `EchoBot` конструктор:

```
public EchoBot(IStorage storage)  
{  
    if (storage is null) throw new ArgumentNullException();  
    _myStorage = storage;  
}
```

# Запуск Cosmos DB Bot

Запустите бот на локальном компьютере.

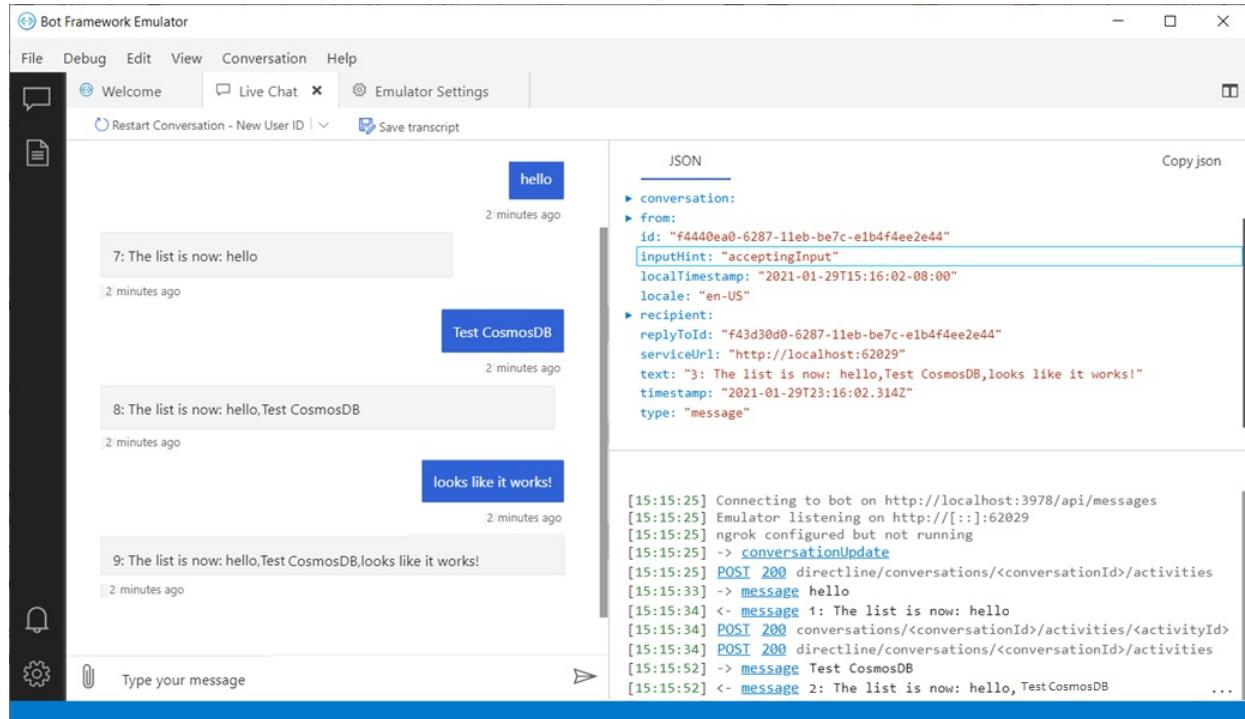
## Тестирование Cosmos DB Bot с помощью эмулятора Bot Framework

Теперь запустите эмулятор Bot Framework и подключитесь к роботу:

1. На вкладке **приветствия** эмулятора щелкните ссылку **создать новую конфигурацию Bot**.
2. Заполните поля для подключения к боту, используя сведения на веб-странице, отображаемой при запуске бота.

## Взаимодействие с Cosmos DB Bot

Отправьте сообщение боту, и он отобразит список полученных сообщений.



## Просмотр данных Cosmos DB

Если вы запустили бот и сохранили информацию, ее можно просмотреть на портале Azure на вкладке **Обозреватель данных**.

The screenshot shows the Azure Data Explorer blade. The left sidebar includes:

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems
- Quick start
- Notifications
- Data Explorer (selected)
- Settings
- Features
- Replicate data globally
- Default consistency
- Backup & Restore

The main area displays the results of a SQL query against the 'bot-storage' database:

```
SQL API
SELECT * FROM c
Edit Filter
```

id	/id
UtteranceLog	UtteranceLog

Load more

```
1   "id": "UtteranceLog",
2   "realId": "UtteranceLog",
3   "document": {
4     "py/object": "bots.echo_bot.UtteranceLog",
5     "turn_number": 3,
6     "utterance_list": [
7       "hello",
8       "Test CosmosDB",
9       "looks like it works!"
10    ],
11 },
12 "_rid": "nD8VAPJDYVsBAAAAAAA==",
13 "_self": "d8VAA==/colls/nD8VAPJDYVs/docs/nD8VAPJDYVs",
14 "_etag": "\"1f00f3ed-0000-0700-0000-601497320000\"",
15 "_attachments": "attachments/",
16 "_ts": 1611962162
```

# Использование хранилища BLOB-объектов

Хранилище BLOB-объектов Azure — это решение корпорации Майкрософт для хранения объектов в облаке. Хранилище BLOB-объектов оптимизировано для хранения больших объемов неструктурированных данных, например текстовых или двоичных данных. В этом разделе объясняется, как создать учетную запись хранилища BLOB-объектов Azure и контейнер, а затем как ссылаться на контейнер хранилища BLOB-объектов из программы-робота.

Дополнительные сведения о хранилище больших двоичных объектов см. в статье [что такое хранилище BLOB-объектов Azure?](#)

## Создание учетной записи хранилища BLOB-объектов

Для использования хранилища BLOB-объектов в боте необходимо выполнить некоторые настройки, прежде чем приступать к написанию кода.

1. На [портале Azure](#) выберите **Все службы**.
2. В разделе **основные материалы** страницы **все службы** выберите **учетные записи хранения**.
3. На странице **учетные записи хранения** выберите \*\* Создать \*\* \*.

Azure Storage is a Microsoft-managed service providing cloud storage that is highly available, secure, durable, scalable, and redundant. Azure Storage includes Azure Blobs (objects), Azure Data Lake Storage Gen2, Azure Files, Azure Queues, and Azure Tables. The cost of your storage account depends on the usage and the options you choose below. [Learn more about Azure storage accounts](#)

**Project details**

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription \* Visual Studio Enterprise – MPN

Resource group \* <Your-Resource-Group-Name>

Create new

**Instance details**

The default deployment model is Resource Manager, which supports the latest Azure features. You may choose to deploy using the classic deployment model instead. [Choose classic deployment model](#)

Storage account name \* <Your-Account-Name>

Location \* (US) West US 2

Performance Standard

Account kind BlobStorage

Replication Read-access geo-redundant storage (RA-GRS)

**Review + create** < Previous Next : Networking >

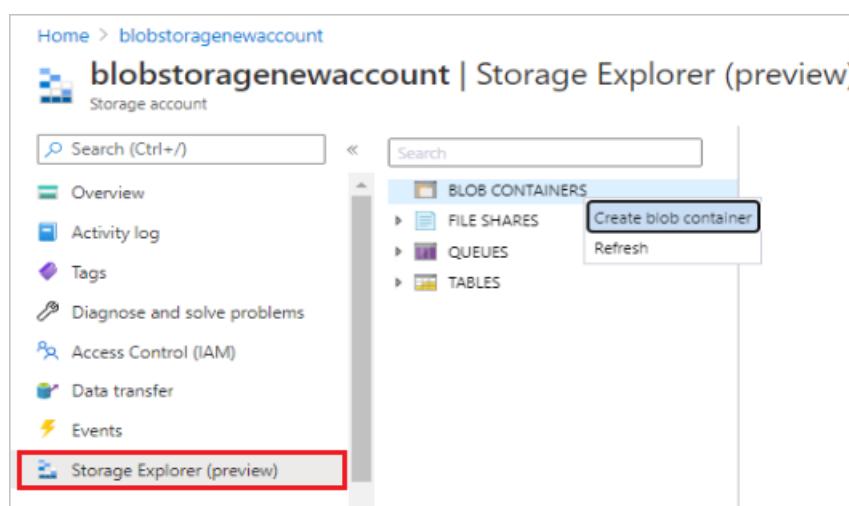
4. В поле **Подписка** выберите подписку, в которой будет создана учетная запись хранения.
5. В поле **Группа ресурсов** выберите существующую группу ресурсов или щелкните **создать** и введите имя новой группы ресурсов.
6. В поле **имя учетной записи хранения** введите имя учетной записи. Обратите внимание на следующие рекомендации.
  - Это имя должно быть уникальным в пределах Azure.
  - Длина имени должна составлять от 3 до 24 символов.
  - Имя может содержать только цифры и строчные буквы.
7. В поле **Расположение** выберите расположение для учетной записи хранения или используйте расположение по умолчанию.
8. Для остальных параметров настройте следующие параметры.
  - **Производительность** — Стандартная. Дополнительные [сведения о производительности](#).
  - **Тип учетной записи**: блобстораже. Дополнительные [сведения об учетных записях хранения](#).
  - **Репликация**. Оставьте значение по умолчанию. Дополнительные [сведения о избыточности](#).

9. В разделе **сведения о проекте** страницы **Создание учетной записи хранения** выберите нужные значения для **подписки и группы ресурсов**.
10. В разделе **сведения об экземпляре** страницы **Создание учетной записи хранения** введите **имя учетной записи хранения**, а затем выберите значения для параметров **Расположение, тип учетной записи и репликация**.
11. Выберите **проверить и создать**, чтобы проверить параметры учетной записи хранения.
12. После проверки выберите **создать**.

#### **Создание контейнера хранилища BLOB-объектов**

После создания учетной записи хранилища BLOB-объектов откройте ее, а затем:

1. Выберите **Обозреватель службы хранилища (Предварительная версия)**.
2. Затем щелкните правой кнопкой мыши **контейнеры больших двоичных объектов**
3. В раскрывающемся списке выберите **создать контейнер BLOB-объектов**.



4. Введите имя в форму **новый контейнер**. Это имя будет использоваться в качестве значения "BLOB-хранилище-контейнер-имя", чтобы предоставить доступ к учетной записи хранилища больших двоичных объектов. Обратите внимание на следующие рекомендации.
  - Это имя может содержать только строчные буквы, цифры и дефисы.
  - Это имя должно начинаться с буквы или цифры.
  - Перед каждым дефисом должен быть указан допустимый символ, отличный от дефиса.
  - Длина имени должна составлять от 3 до 63 символов.

#### **Добавление сведений о конфигурации хранилища BLOB-объектов**

Найдите ключи хранилища BLOB-объектов, необходимые для настройки хранилища BLOB-объектов для программы Bot, как показано выше:

1. В портал Azure откройте учетную запись хранилища BLOB-объектов и выберите **ключи доступа** в разделе **Параметры**.

The screenshot shows the 'Access keys' section of the Azure Storage account settings. It includes a key named 'key1' and a connection string. The connection string is highlighted with a red box.

Используйте **строку подключения** в качестве значения для *строки подключения*, чтобы предоставить доступ к учетной записи хранилища BLOB-объектов.

- [C#](#)
- [JavaScript](#)
- [Python](#)

Добавьте следующие сведения в файл конфигурации.

#### appsettings.json

```
"BlobConnectionString": "<your-blob-connection-string>",
"BlobContainerName": "<your-blob-container-name>,
```

#### Установка пакетов хранилища BLOB-объектов

Установите следующие пакеты, если они не были установлены ранее.

- [C#](#)
- [JavaScript](#)
- [Python](#)

Установите пакет NuGet **Microsoft.Bot.Builder.Azure.blobs**. Дополнительные сведения об использовании NuGet см. в статье [Установка пакетов и управление ими в Visual Studio с помощью диспетчера пакетов NuGet](#).

#### Реализация хранилища BLOB-объектов

Хранилище BLOB-объектов используется для хранения состояния Bot.

- [C#](#)
- [JavaScript](#)
- [Python](#)

#### NOTE

Начиная с версии 4,10, `Microsoft.Bot.Builder.Azure.AzureBlobStorage` является устаревшим. Используйте новый `Microsoft.Bot.Builder.Azure.Blobs.BlobsStorage` в своем месте.

Следующий пример кода выполняется с использованием того же кода Bot, что и в приведенном выше примере [хранилища памяти](#), за исключением перечисленных здесь исключений.

В приведенных ниже фрагментах кода показана реализация хранилища BLOB-объектов для "myStorage", которая заменяет хранилище локальной памяти.

Сначала необходимо обновить Startup.cs , чтобы он ссылался на библиотеку *больших двоичных объектов Azure* для Bot Builder :

### Startup.cs

```
using Microsoft.Bot.Builder.Azure.Blobs;
```

Затем в ConfigureServices методе в Startup.cs создайте BlobsStorage объект, передав значения из appsettings.json . Он передается в EchoBot конструктор посредством внедрения зависимостей.

```
//Use Azure Blob storage, instead of in-memory storage.  
services.AddSingleton<IStorage>(  
    new BlobsStorage(  
        Configuration.GetValue<string>("dataConnectionString"),  
        Configuration.GetValue<string>("containerName")  
    ));
```

Теперь сначала необходимо обновить EchoBot.cs , чтобы он ссылался на библиотеку *больших двоичных объектов Azure* для Bot Builder :

### EchoBot.cs

```
using Microsoft.Bot.Builder.Azure.Blobs;
```

Затем удалите или закомментируйте строку кода, которая создает переменную Мемористораже "private static readonly Мемористораже \_myStorage = New Мемористораже ();", и создайте новую переменную, которая будет использоваться для сохранения входных данных пользователя в хранилище BLOB-объектов.

### EchoBot.cs

```
// variable used to save user input to CosmosDb Storage.  
private readonly IStorage _myStorage;
```

Затем передайте IStorage объект в EchoBot конструктор:

```
public EchoBot(IStorage storage)  
{  
    if (storage is null) throw new ArgumentNullException();  
    _myStorage = storage;  
}
```

После того как хранилище настроено для указания учетной записи хранилища BLOB-объектов, код программы-робота теперь будет хранить и получать данные из хранилища BLOB-объектов.

После того как хранилище настроено для указания учетной записи хранилища BLOB-объектов, код программы-робота теперь будет хранить и получать данные из хранилища BLOB-объектов.

## Запуск ленты для хранилища BLOB-объектов

Запустите бот на локальном компьютере.

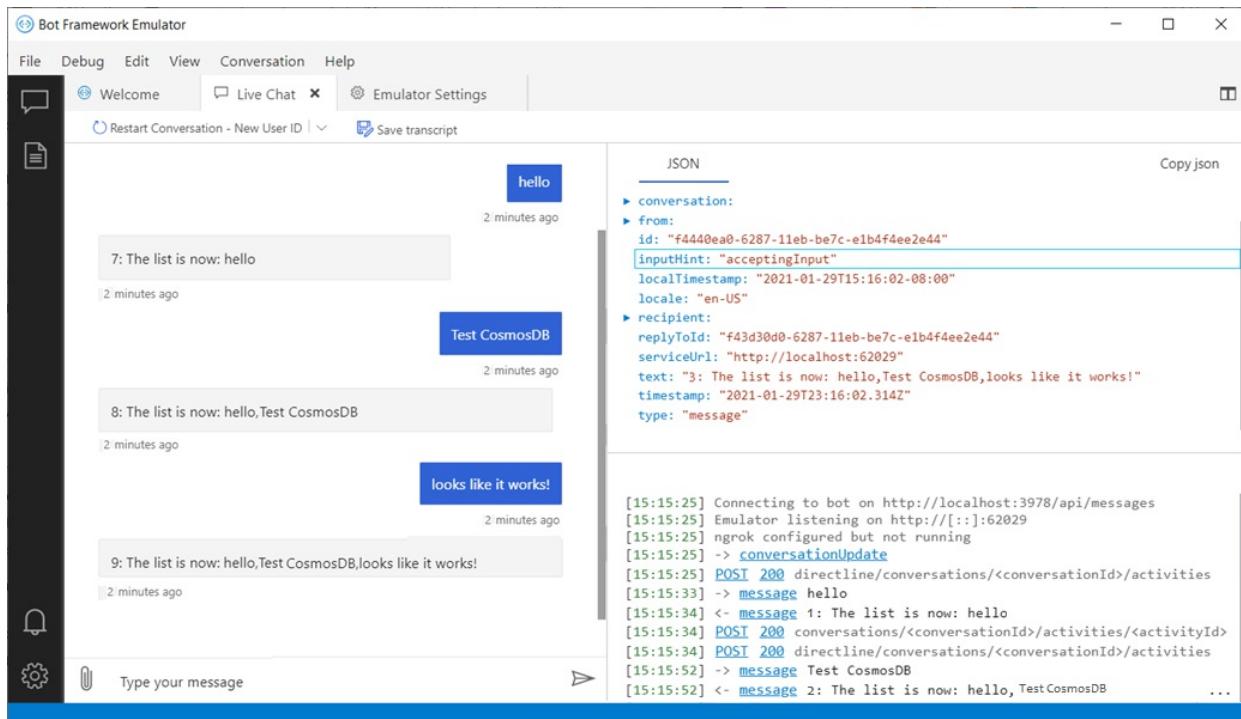
# Запуск эмулятора и подключение ленты хранилища BLOB-объектов

Затем запустите эмулятор и подключитесь к роботу в эмуляторе:

- Щелкните ссылку **создать конфигурацию Bot** на вкладке "Добро пожаловать" в эмуляторе.
- Заполните поля для подключения к боту, используя сведения на веб-странице, отображаемой при запуске бота.

## Взаимодействие с программой-роботом хранилища BLOB-объектов

Отправьте сообщение боту, и он отобразит список полученных сообщений.



### Просмотр данных хранилища BLOB-объектов

Если вы запустили бот и сохранили информацию, ее можно просмотреть на портале Azure на вкладке **Обозреватель службы хранилища**.

## Хранилище расшифровок разговоров в BLOB-объектах

Хранилище расшифровок в BLOB-объектах Azure — это специализированное хранилище, которое позволяет легко сохранять и получать разговоры пользователей в виде записей расшифровок. Использовать хранилище расшифровок разговоров в больших двоичных объектах Azure особенно удобно для автоматической записи данных, вводимых пользователями, и дальнейшего их анализа при отладке бота.

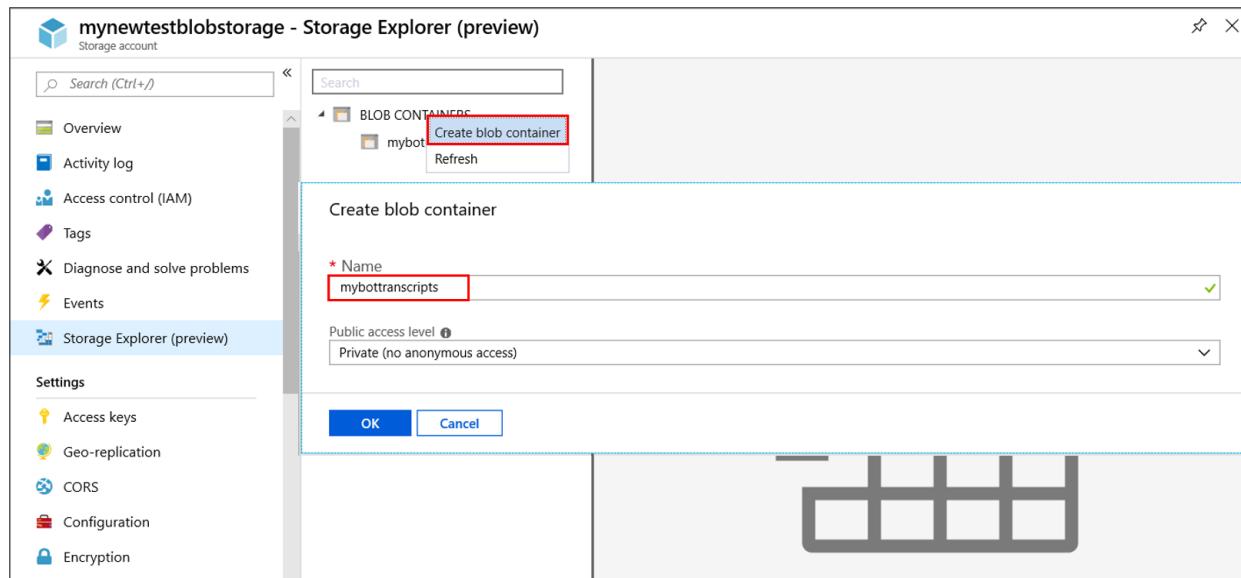
#### NOTE

В настоящее время Python не поддерживает хранилище для записи *BLOB-объектов Azure*. Хотя JavaScript поддерживает хранилище для записи BLOB-объектов, следующие направления предназначены только для C#.

### Настройка контейнера хранилища для записи BLOB-объектов

Хранилище расшифровок разговоров в BLOB-объектах Azure использует ту же учетную запись

хранилища BLOB-объектов, которая описана в разделах *Создание учетной записи хранилища BLOB-объектов* и *Добавление сведений о конфигурации* выше. Теперь добавьте контейнер для хранения расшифровок.



1. Откройте учетную запись хранилища BLOB-объектов Azure
2. Выберите **Обозреватель службы хранилища**.
3. Щелкните правой кнопкой мыши **Контейнеры больших двоичных объектов** и выберите **Создать контейнер BLOB-объектов**.
4. Введите имя для контейнера транскрипции и нажмите кнопку **OK**. (Мы указали *мивоттранскрипта*)

#### Реализация хранилища для записи большого двоичного объекта

В коде ниже указатель на хранилище расшифровок `_myTranscripts` подключается к новой учетной записи хранилища BLOB-объектов с расшифровками. Чтобы создать эту ссылку с новым именем контейнера, `<your-blob-transcript-container-name>` создает новый контейнер в хранилище BLOB-объектов для хранения файлов с транскрипциями.

Хранилище для расшифровки BLOB-объектов предназначено для хранения записей роботов.

#### NOTE

Начиная с версии 4.10, `Microsoft.Bot.Builder.Azure.AzureBlobTranscriptStore` является устаревшим. Используйте новый `Microsoft.Bot.Builder.Azure.Blobs.BlobsTranscriptStore` в своем месте.

#### echoBot.cs

```
using Microsoft.Bot.Builder.Azure.Blobs;

public class EchoBot : ActivityHandler
{
    ...

    private readonly BlobsTranscriptStore _myTranscripts = new BlobsTranscriptStore("<your-azure-storage-connection-string>", "<your-blob-transcript-container-name>");

    ...
}
```

#### Хранение расшифровок разговоров пользователей в виде BLOB-объектов Azure

Создав контейнер BLOB-объектов для хранения расшифровок, можно будет сохранять разговоры пользователей с ботом. Эти диалоги можно позже использовать для отладки, чтобы проанализировать взаимодействие пользователей с ботом. Каждый *диалог перезапуска* эмулятора инициирует создание нового списка разговора. Следующий код сохраняет диалог с пользователем в сохраненном файле с расшифровками.

- Текущая расшифровка сохраняется с использованием `LogActivityAsync`.
- Сохраненные расшифровки можно получить с использованием `ListTranscriptsAsync`. В этом примере кода идентификатор каждой сохраненной расшифровки включается в список `storedTranscripts`. Позже этот список используется для управления количеством расшифровок, хранимых в больших двоичных объектах.

#### echoBot.cs

```
protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    await _myTranscripts.LogActivityAsync(turnContext.Activity);

    List<string> storedTranscripts = new List<string>();
    PagedResult<Microsoft.Bot.Builder.TranscriptInfo> pagedResult = null;
    var pageSize = 0;
    do
    {
        pagedResult = await _myTranscripts.ListTranscriptsAsync("emulator", pagedResult?.ContinuationToken);
        pageSize = pagedResult.Items.Count();

        // transcript item contains ChannelId, Created, Id.
        // save the channelIds found by "ListTranscriptsAsync" to a local list.
        foreach (var item in pagedResult.Items)
        {
            storedTranscripts.Add(item.Id);
        }
    } while (pagedResult.ContinuationToken != null);

    ...
}
```

#### Управление расшифровками, хранимыми в больших двоичных объектах

Хотя хранимые расшифровки можно использовать как средство отладки, со временем их количество превышает возможности хранилища. Дополнительный код ниже использует `DeleteTranscriptAsync` для удаления из хранилища BLOB-объектов всех извлеченных элементов расшифровок, кроме последних трех.

#### echoBot.cs

```

protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    await _myTranscripts.LogActivityAsync(turnContext.Activity);

    List<string> storedTranscripts = new List<string>();
    PagedResult<Microsoft.Bot.Builder.TranscriptInfo> pagedResult = null;
    var pageSize = 0;
    do
    {
        pagedResult = await _myTranscripts.ListTranscriptsAsync("emulator", pagedResult?.ContinuationToken);
        pageSize = pagedResult.Items.Count();

        // transcript item contains ChannelId, Created, Id.
        // save the channelIds found by "ListTranscriptsAsync" to a local list.
        foreach (var item in pagedResult.Items)
        {
            storedTranscripts.Add(item.Id);
        }
    } while (pagedResult.ContinuationToken != null);

    // Manage the size of your transcript storage.
    for (int i = 0; i < pageSize; i++)
    {
        // Remove older stored transcripts, save just the last three.
        if (i < pageSize - 3)
        {
            string thisTranscriptId = storedTranscripts[i];
            try
            {
                await _myTranscripts.DeleteTranscriptAsync("emulator", thisTranscriptId);
            }
            catch (System.Exception ex)
            {
                await turnContext.SendActivityAsync("Debug Out: DeleteTranscriptAsync had a problem!");
                await turnContext.SendActivityAsync("exception: " + ex.Message);
            }
        }
    }
    ...
}

```

Дополнительные сведения о классе см. в статье о [хранилище записей BLOB-объектов Azure](#).

## Дополнительные сведения

### Управление параллелизмом с помощью тегов eTag

В нашем примере кода бота мы установили свойство `eTag` каждого экземпляра `IStoreItem` равным `*`.

Член `eTag` (тег сущности) объекта хранилища используется в Cosmos DB для управления параллелизмом. `eTag` указывает базе данных, что делать, если другой экземпляр бота изменил объект в том же хранилище, в которое записывает данные ваш бот.

#### Приоритет последней записи — разрешить перезапись

Если указать звездочку (`*`) в качестве значения свойства `eTag`, это означает, что приоритет принадлежит последнему боту, который выполняет запись. При создании хранилища данных можно указать значение `*` для свойства `eTag`. Это означает, что вы не сохранили записываемые данные, или что вы хотите, чтобы последний бот перезаписал все ранее сохраненные свойства. Если параллелизм не является проблемой для вашего бота, укажите для свойства `eTag` значение `*`. Это разрешает перезапись.

## Поддержание параллелизма и предотвращение перезаписи

Если нужно предотвратить параллельный доступ к свойству и избежать перезаписи данных другим экземпляром бота, при сохранении данных в Cosmos DB используйте другое значение свойства `eTag`, отличное от `*`. Если бот попытается сохранить данные о состоянии, а параметр `eTag` не соответствует параметру `eTag` в хранилище, бот получит сообщение об ошибке `etag conflict key=`.

По умолчанию хранилище Cosmos DB проверяет равенство свойства `eTag` в объекте хранилища каждый раз, когда бот записывает данные в этот элемент, а затем включает в это свойство новое уникальное значение после каждой операции записи. Если значение свойства `eTag` во время записи не соответствует значению свойства `eTag` в хранилище, это означает, что другой бот или поток изменили данные.

Например, предположим, что ваш бот должен изменять сохраненные заметки, но вы не хотите, чтобы он перезаписывал изменения, выполненные другим экземпляром бота. Если другой экземпляр бота внес изменения, пользователь должен изменять последнюю версию со всеми внесенными изменениями.

- [C#](#)
- [JavaScript](#)
- [Python](#)

Во-первых, создайте класс, который реализует `IStoreItem`.

### EchoBot.cs

```
public class Note : IStoreItem
{
    public string Name { get; set; }
    public string Contents { get; set; }
    public string ETag { get; set; }
}
```

Затем создайте начальную заметку, создав объект хранилища и добавив объект в это хранилище.

### EchoBot.cs

```
// create a note for the first time, with a non-null, non-* ETag.
var note = new Note { Name = "Shopping List", Contents = "eggs", ETag = "x" };

var changes = Dictionary<string, object>();
{
    changes.Add("Note", note);
};
await NoteStore.WriteAsync(changes, cancellationToken);
```

Впоследствии обновите заметку, сохранив значение `eTag`, прочитанное из хранилища.

### EchoBot.cs

```
var note = NoteStore.ReadAsync<Note>("Note").Result?.FirstOrDefault().Value;

if (note != null)
{
    note.Contents += ", bread";
    var changes = new Dictionary<string, object>();
    {
        changes.Add("Note1", note);
    };
    await NoteStore.WriteAsync(changes, cancellationToken);
}
```

Если заметка в хранилище была изменена перед записью ваших изменений, то при вызове `Write` возникнет исключение.

Для поддержания параллелизма всегда считывайте значение свойства из хранилища, затем изменяйте прочитанное свойство, чтобы сохранить `eTag`. При считывании сведений о пользователях из хранилища ответ будет содержать свойство `eTag`. Если вы изменили данные и записываете обновленные данные в хранилище, ваш запрос должен включать свойство `eTag`, содержащее то же значение, которое было прочитано ранее. Однако при записи объекта со свойством `eTag`, имеющим значение `*`, будет разрешена перезапись любых других изменений.

## Дальнейшие действия

Теперь, когда вы знаете, как напрямую считывать данные из хранилища и записывать данные в хранилища, посмотрим, как сделать это с помощью диспетчера состояний.

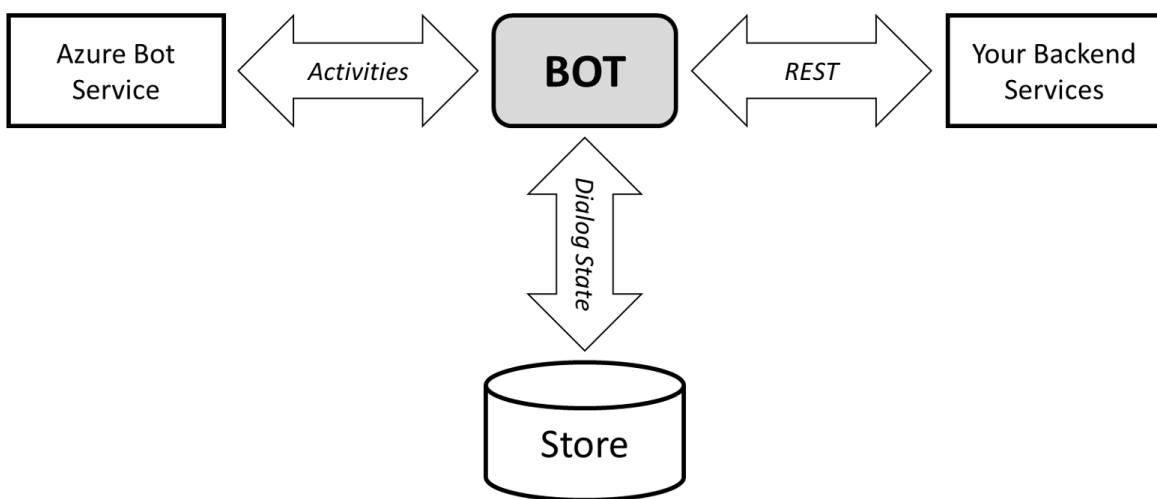
[Сохранение состояния с помощью свойств `conversation` и `user`](#)

# Реализация пользовательского хранилища для бота

27.10.2020 • 28 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Взаимодействия бота делятся на три категории: во-первых, обмен действиями со службой Azure Bot; во-вторых, загрузка и сохранение состояний диалога в хранилище; и, наконец, взаимодействие со всеми остальными внутренними службами, которые потребуются для выполнения задач бота.



## Предварительные требования

- Полный пример кода, используемый в этой статье можно найти здесь: [Пример C#](#).

В этой статье мы изучим семантику взаимодействий бота со службой Azure Bot и хранилищем.

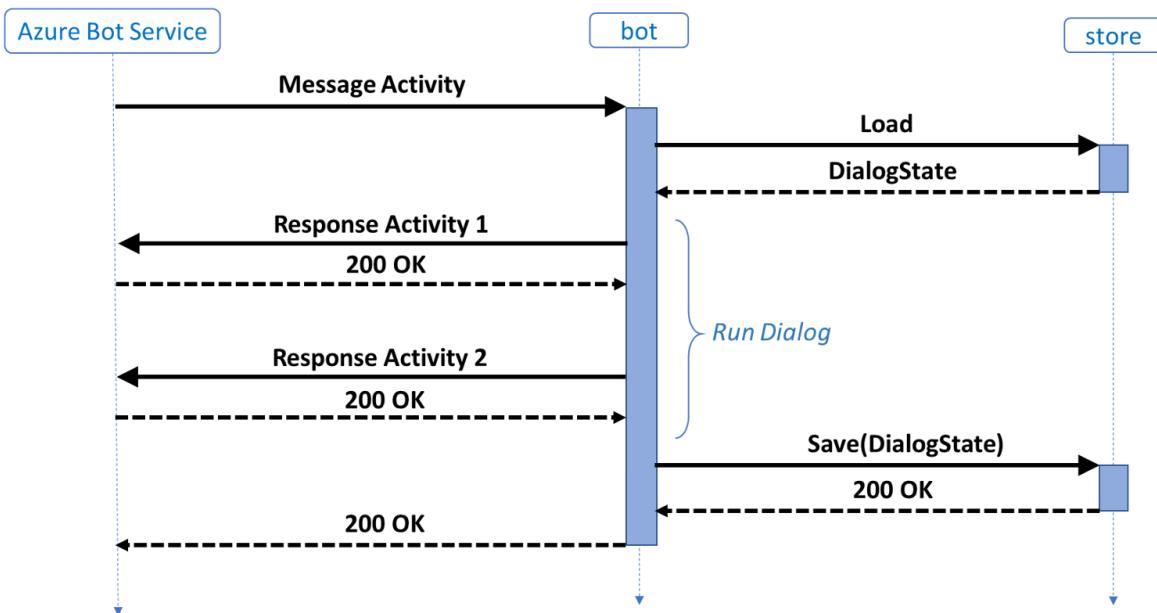
Bot Framework включает реализацию по умолчанию, которая хорошо подойдет для большинства приложений. Вам достаточно будет объединить все компоненты в нескольких строках кода инициализации. Многие из этих примеров демонстрируют именно такой вариант.

Эти примеры в основном приводятся для того, чтобы описать варианты действий на случай, если стандартная семантика реализации в вашем приложении будет работать не совсем так, как ожидается. Главное, что важно понять — это платформа, а не готовое приложение с предопределенным поведением. Другими словами, многие механизмы на этой платформе служат просто примерами реализации по умолчанию, но ни в коем случае не единственным возможным вариантом.

В частности, платформа не навязывает конкретных взаимосвязей между обменом действиями со службой Azure Bot и сведениями о состоянии бота, которые она загружает и (или) сохраняет. В ней просто реализован вариант по умолчанию. Мы дополнитель но подкрепим это напоминание, создав альтернативную реализацию этого механизма с другой семантикой. Альтернативное решение ничем не хуже для самой платформы, а для разрабатываемого приложения может оказаться даже лучше. Здесь все зависит от конкретного сценария.

## Поведение для адаптера BotFrameworkAdapter по умолчанию и поставщиков хранилища

Сначала рассмотрим реализацию по умолчанию, которая предоставляется в пакетах платформы, как показано на следующей схеме:



Получив определенное действие, бот загружает состояние соответствующей беседы. Затем он выполняет логику диалога, определенную для текущего состояния и (или) полученного действия. В процессе выполнения диалога создаются одно или несколько действий, которые немедленно отправляются пользователю. Завершив обработку диалога, бот сохраняет новое состояние взамен предыдущего.

Здесь важно учесть несколько моментов, которые могут привести к ошибкам.

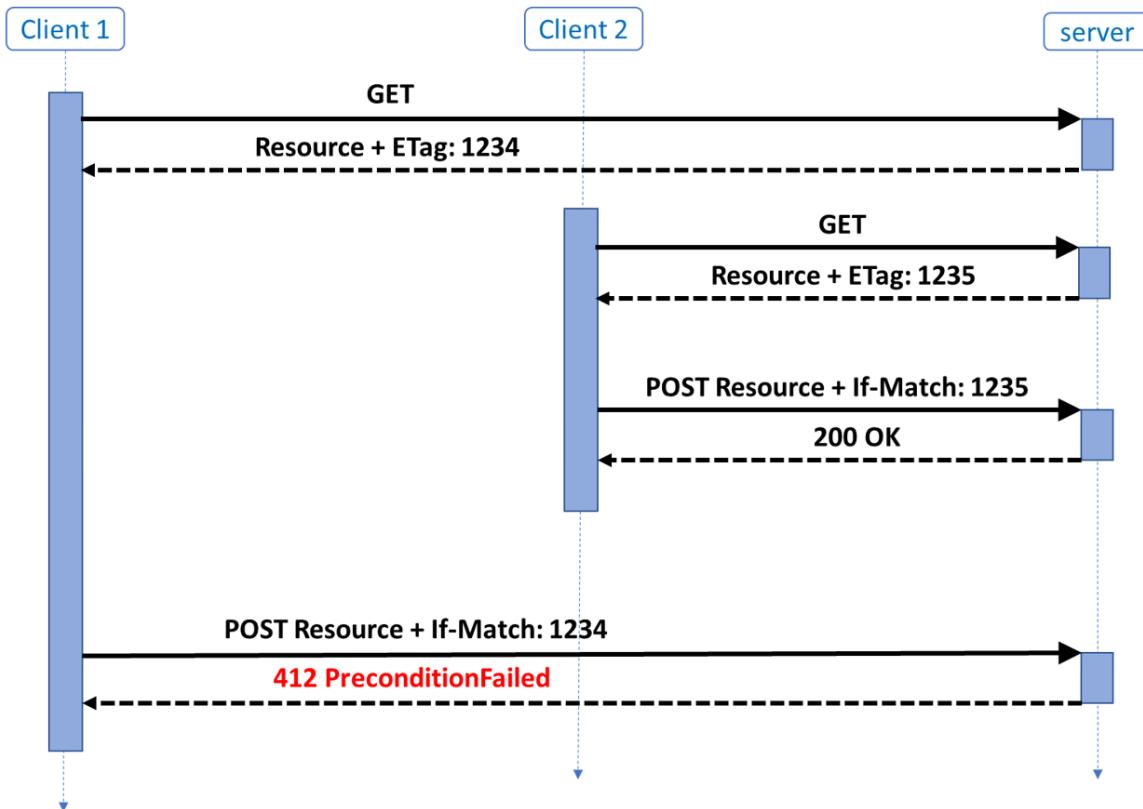
Во-первых, если операция сохранения по любой причине завершится ошибкой, неявно нарушится синхронизация бота. Пользователь увидит в канале сообщение и будет уверен, что состояние беседы изменилось, тогда как фактически оно останется прежним. Такая ситуация обычно хуже, чем успешное сохранение текущего состояния и ответного сообщения. Такую ситуацию можно учесть в структуре диалога, например создать дополнительные запросы для подтверждения, которые будут выглядеть избыточными в других ситуациях.

Во-вторых, если реализация развертывается на нескольких узлах, состояние может быть случайно перезаписано. При этом диалог даже успешно отправит в канал подтверждающие сообщения, что внесет еще больше путаницы. Например, если при работе бота приема заказов на пиццу, развернутого в нескольких экземплярах, пользователь в ответ на вопрос о начинке укажет грибы и сразу же укажет сыр, эти последовательные действия могут быть отправлены в разные экземпляры бота. В таком случае возникает состояние гонки, при котором один из компьютеров перезапишет сохраненное другим компьютером состояние. В нашем примере это приведет к тому, что пользователь получит уже отправленные ответы с подтверждением обеих начинок: грибов и сыра. Но в пицце, которую ему доставят, будет только одна из этих начинок: грибы или сыр.

## Оптимистическая блокировка

Решение заключается в том, чтобы создать блокировку для работы с состояниями. Здесь мы применим стиль, который называется оптимистической блокировкой. Это означает, что каждый компонент системы будет работать так, как если бы других экземпляров не существовало, а мы будем дополнительно отслеживать нарушения параллелизма уже после обработки. На первый взгляд это выглядит сложным, но реализация на основе облачных технологий хранения и правильных расширений для платформы ботов будет очень простой.

Мы применим стандартный механизм HTTP на основе заголовка тега сущности (ETag). Чтобы понять приведенный ниже код, важно сначала как следует разобраться в основном механизме. На схеме ниже показана вся последовательность действий.



На схеме представлен пример с двумя клиентами, которые выполняют обновление одного ресурса. Когда клиент отправляет запрос GET, полученный в ответе от сервера ресурс дополняется заголовком ETag. Заголовок ETag содержит непрозрачное значение, которое обозначает состояние ресурса. ETag обновляется при любом изменении ресурса. Когда клиент завершит обновление сведений о состоянии, он отправляет ресурс обратно на сервер в запросе POST, присоединяя к нему ранее полученное значение ETag в заголовке предварительного условия If-Match. Если значение ETag не соответствует последнему отправленному сервером значению (учитываются любые ответы любому клиенту), проверка предварительного условия завершается сбоем с кодом состояния "412: ошибка в предусловии". Такое сообщение информирует клиента, от которого поступил запрос POST, что состояние ресурса уже изменилось. Обычно в ответ на эту ошибку клиент повторно получает ресурс с помощью запроса GET, повторно применяет нужные изменения и повторно отправляет ресурс на сервер в запросе POST. Скорее всего, второй запрос POST будет выполнен успешно, если за время выполнения этой процедуры другие клиенты не изменили этот же ресурс снова. Но даже если это так, клиент просто повторит попытку еще раз.

Такой процесс называется "оптимистическим", так как клиент при получении ресурса для обработки не "блокирует" этот ресурс в прямом смысле, то есть не ограничивает обращения к нему от других клиентов. Любые конфликты между клиентами в отношении состояния ресурса никак не выявляются до завершения обработки. Для распределенных систем такая стратегия обычно дает более оптимальный результат, чем "пессимистическая блокировка".

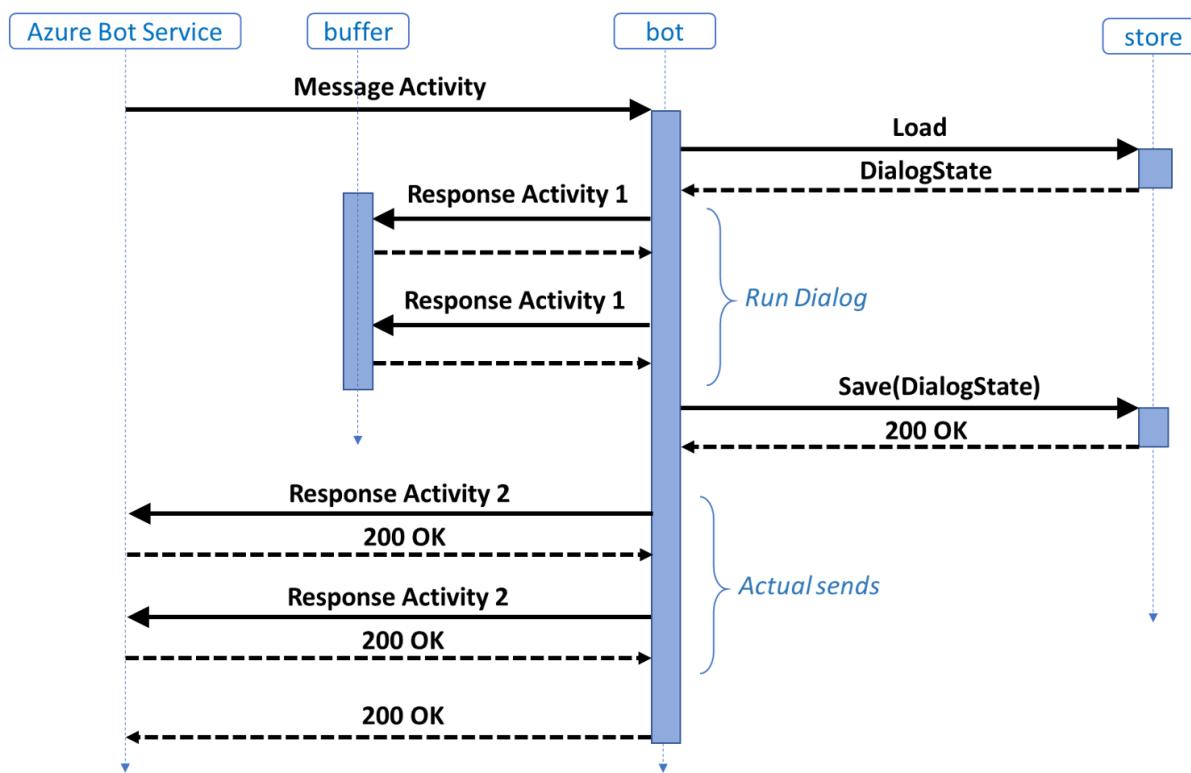
Оптимистический механизм блокировки, который мы рассмотрели выше, основан на безопасности повторного выполнения логики обработки. Как можно легко догадаться, эта безопасность зависит от характера вызовов к внешним службам. Для этой схемы лучше всего, если внешние службы являются идемпотентными. В компьютерной теории идемпотентной называется такая операция, которая не приводит ни к каким дополнительным последствиям при повторном выполнении с теми же входными параметрами. Этому определению соответствуют "чистые" службы HTTP REST, которые поддерживают запросы GET, PUT и (или) DELETE. Логика этих зависимостей интуитивно понятна: наш бот может

повторять попытки обработки и для него очень удобно, что повторение вызовов внешних служб не имеет никаких побочных эффектов. В рамках дальнейшего обсуждения мы предполагаем, что оказались в идеальном мире и все серверные службы из правой части рисунка (в начале этой статьи) являются идемпотентными службами HTTP REST. Это позволит нам забыть про них и сосредоточиться на обмене действиями.

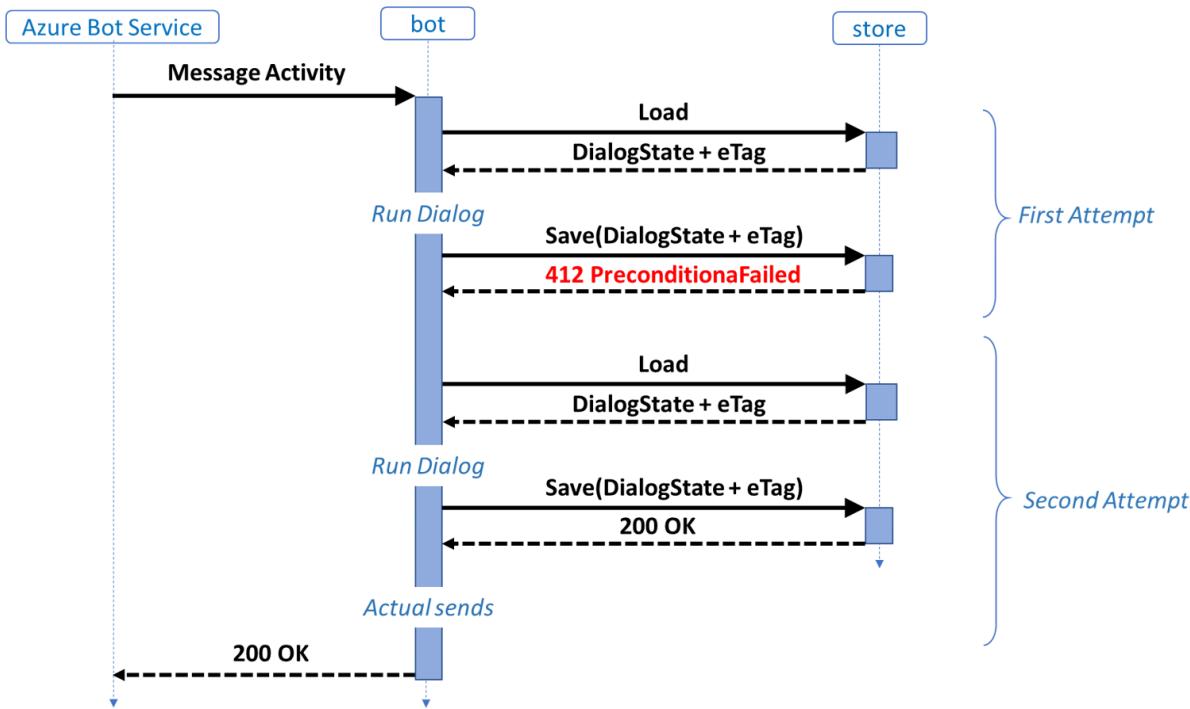
## Буферизация исходящих действий

Отправка действия не является идемпотентной операцией, и мы даже не можем описать идемпотентную логику для этого сценария. Ведь эти действия часто содержат передаваемые сообщения, которые добавляются в представление и (или) проговариваются агентом преобразования текста в речь.

При отправке сообщений для нас важно избежать повторной и многократной отправки. Проблема здесь заключается в том, что механизм оптимистической блокировки требует возможности многократно запускать логику обработки. Но у нас есть простое решение: помещать все исходящие действия диалога в буфер, пока не станет ясно, что повторять выполнение логики больше не потребуется. Например, пока операция сохранения не будет успешно выполнена. Нам нужна примерно такой поток операций:



Если нам удастся создать цикл повторных попыток для выполнения диалога, то при наличии обстоятельств, приводящих к сбою операции сохранения, мы получим следующее поведение:



Добавление этого механизма к схеме из предыдущего примера позволит полностью избавиться от ошибочных подтверждений при добавлении в заказ начинок для пиццы. Таким образом, несмотря на масштабирование развертывания на несколько компьютеров, мы фактически сериализовали все обновления состояния с помощью механизма оптимистической блокировки. Теперь в нашем боте для заказа пиццы можно даже включить сообщение о правильном состоянии в подтверждение добавления элемента. Например, если пользователь быстро вводит слова "cheese" (сыр) и "mushroom" (грибы), не дожидаясь промежуточного ответа бота, мы сможем сформировать ответы "pizza with cheese" (пицца с сыром) и далее "pizza with cheese and mushroom" (пицца с сыром и грибами).

На схеме последовательности можно заметить, что ответы могут теряться после успешной операции сохранения, но они точно так же могут теряться на любом этапе коммуникации. С этой проблемой инфраструктура управления состояниями ничего сделать не может. Для ее решения нам потребуется протокол более высокого уровня и, возможно, даже с участием пользователя канала. Например, если со стороны пользователя будет казаться, что бот ничего не ответил, пользователь может повторить последнее действие или сделать что-то еще в этом роде. Здесь важно, что мы вправе ожидать некоторое количество временных сбоев в сценарии из-за таких проблем, но не вправе требовать, чтобы пользователь был готов к ошибочным положительным подтверждениям и (или) другим непреднамеренным сообщениям.

Собрав все эти фрагменты в единую схему в пользовательском решении для хранения, мы получим три дополнительные функции, которые отсутствуют в стандартной реализации. Во-первых, мы применим ETag для обнаружения конфликтов, во-вторых, будем повторять логику обработки при обнаружении несоответствий в ETag и, в-третьих, поместим в буфер все исходящие действия до успешного сохранения состояния. В оставшейся части этой статьи мы опишем реализацию этих трех функциональных компонентов.

## Реализация поддержки ETag

Начнем с определения интерфейса для нашего нового магазина с поддержкой тегов ETag. Кроме того, интерфейс значительно упростит применение механизмов введения зависимостей, который реализован в ASP.NET. Наличие интерфейса означает, что мы можем реализовать версию для рабочей среды. Также можно реализовать версию для модульных тестов, выполняемых в памяти, без необходимости обращения к сети.

Интерфейс содержит методы Load и Save. Оба они принимают в качестве параметра ключ, который определяет текущее состояние. Метод Load будет возвращать данные и связанный с ними ETag. Метод Save будет принимать те же данные. Выходные данные метода Save будут содержать логическое значение. Оно подтверждает, что значения ETag совпали и сохранение выполнено успешно. Это значение следует рассматривать не как индикатор ошибок общего характера, а только как индикатор несоблюдения конкретного предварительного условия. Поэтому мы реализуем его как код возврата, а не как исключение, и создадим вокруг него логику потока управления в форме цикла повторных попыток.

Так как самый нижний уровень хранилища должен быть подключаемым, мы намеренно отказались от любых требований к сериализации, но хотим указать для содержимого обязательный формат JSON, чтобы в реализации хранилища можно было задать значение content-type. В .NET это проще и логичнее всего реализовать через типы аргументов, в частности этому аргументу содержимого мы присвоим тип JObject. В JavaScript и TypeScript для этого используется регулярный собственный объект.

В результате мы получаем следующий интерфейс:

### IStore.cs

```
public interface IStore
{
    Task<(JObject content, string etag)> LoadAsync(string key);

    Task<bool> SaveAsync(string key, JObject content, string etag);
}
```

Его реализация в хранилище BLOB-объектов Azure не составляет никаких трудностей.

### BlobStore.cs

```
public class BlobStore : IStore
{
    private readonly CloudBlobContainer _container;

    public BlobStore(string accountName, string accountKey, string containerName)
    {
        if (string.IsNullOrWhiteSpace(accountName))
        {
            throw new ArgumentException(nameof(accountName));
        }

        if (string.IsNullOrWhiteSpace(accountKey))
        {
            throw new ArgumentException(nameof(accountKey));
        }

        if (string.IsNullOrWhiteSpace(containerName))
        {
            throw new ArgumentException(nameof(containerName));
        }

        var storageCredentials = new StorageCredentials(accountName, accountKey);
        var cloudStorageAccount = new CloudStorageAccount(storageCredentials, useHttps: true);
        var client = cloudStorageAccount.CreateCloudBlobClient();
        _container = client.GetContainerReference(containerName);
    }

    public async Task<(JObject content, string etag)> LoadAsync(string key)
    {
        if (string.IsNullOrWhiteSpace(key))
        {
            throw new ArgumentException(nameof(key));
        }
```

```

var blob = _container.GetBlockBlobReference(key);
try
{
    var content = await blob.DownloadTextAsync();
    var obj = JObject.Parse(content);
    var etag = blob.Properties.ETag;
    return (obj, etag);
}
catch (StorageException e)
    when (e.RequestInformation.HttpStatusCode == (int)HttpStatusCode.NotFound)
{
    return (new JObject(), null);
}
}

public async Task<bool> SaveAsync(string key, JObject obj, string etag)
{
    if (string.IsNullOrWhiteSpace(key))
    {
        throw new ArgumentException(nameof(key));
    }

    if (obj == null)
    {
        throw new ArgumentNullException(nameof(obj));
    }

    var blob = _container.GetBlockBlobReference(key);
    blob.Properties.ContentType = "application/json";
    var content = obj.ToString();
    if (etag != null)
    {
        try
        {
            await blob.UploadTextAsync(content, Encoding.UTF8, new AccessCondition {IfMatchETag = etag},
new BlobRequestOptions(), new OperationContext());
        }
        catch (StorageException e)
            when (e.RequestInformation.HttpStatusCode == (int)HttpStatusCode.PreconditionFailed)
        {
            return false;
        }
    }
    else
    {
        await blob.UploadTextAsync(content);
    }

    return true;
}
}

```

Как вы видите, основную работу выполняет хранилище BLOB-объектов. Обратите внимание, как перехватываются конкретные исключения и как они приводятся в соответствие с ожиданиями вызывающего кода. Мы хотим, чтобы исключение "Не найдено" при загрузке возвращало значение null, а исключение "Необходимое условие не выполнено" при сохранении возвращало логическое значение.

Весь исходный код доступен в соответствующем [примере](#), который включает также реализацию хранилища в памяти.

## Реализация цикла повторных попыток

Основной вид этого цикла напрямую определяется поведением, которое представлено на схеме последовательности.

Получив действие, мы создаем ключ для состояния соответствующей беседы. Зависимость между действием и беседой никак не изменяется. Поэтому мы сохраним логику создания ключа из реализации состояния по умолчанию.

После создания ключа мы пытаемся загрузить соответствующее состояние. Затем выполняем все нужные диалоги бота и сохраняем состояние. Если сохранение проходит успешно, мы отправляем накопленные в этом диалоге исходящие действия и завершаем работу. В противном случае возвращаемся назад и повторяем весь процесс, начиная с загрузки данных. Повторная загрузка предоставит нам новое значение ETag, что позволяет надеяться на успешное сохранение при очередной попытке.

В итоге полная реализация OnTurn выглядит следующим образом:

### ScaleoutBot.cs

```
protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    // Create the storage key for this conversation.
    var key = $"{turnContext.Activity.ChannelId}/conversations/{turnContext.Activity.Conversation?.Id}";

    // The execution sits in a loop because there might be a retry if the save operation fails.
    while (true)
    {
        // Load any existing state associated with this key
        var (oldState, etag) = await _store.LoadAsync(key);

        // Run the dialog system with the old state and inbound activity, the result is a new state and
        // outbound activities.
        var (activities, newState) = await DialogHost.RunAsync(_dialog, turnContext.Activity, oldState,
        cancellationToken);

        // Save the updated state associated with this key.
        var success = await _store.SaveAsync(key, newState, etag);

        // Following a successful save, send any outbound Activities, otherwise retry everything.
        if (success)
        {
            if (activities.Any())
            {
                // This is an actual send on the TurnContext we were given and so will actually do a send this
                // time.
                await turnContext.SendActivitiesAsync(activities, cancellationToken);
            }

            break;
        }
    }
}
```

Обратите внимание на то, что выполнение диалога здесь оформлено как вызов функции. Можно создать и более сложные реализации, определив интерфейс и создав внедряемую зависимость. Но для нашего примера включение диалога в статическую функцию хорошо согласуется с функциональной природой выбранного подхода. В общем случае такая реализация решения, в которой критически важные части становятся функциональными компонентами, станет хорошей основой для успешной работы решения в сетевой среде.

## Реализация буферизации исходящих действий

Следующее требование — помещать в буфер все исходящие действия, пока не будет выполнено успешное сохранение. Для этого мы применим пользовательскую реализацию BotAdapter. В этом коде реализована абстрактная функция SendActivity, которая добавляет действие в список вместо

немедленной отправки. Это никак не повлияет на размещаемый диалог. В этом конкретном сценарии не поддерживаются операции UpdateActivity и DeleteActivity. Поэтому эти методы просто создают исключение Not Implemented (Не реализовано). Кроме того, мы не учитываем возвращаемое значение SendActivity. Оно используется некоторыми каналами в таких сценариях, когда нужны обновления действий, например для отключения кнопок в карточках, отображаемых в канале. Обмен такими сообщениями может быть довольно сложным, особенно при отслеживании состояния, и эти сложности выходят за рамки нашей статьи. Полная реализация пользовательского BotAdapter выглядит следующим образом:

### DialogHostAdapter.cs

```
public class DialogHostAdapter : BotAdapter
{
    private List<Activity> _response = new List<Activity>();

    public IEnumerable<Activity> Activities => _response;

    public override Task<ResourceResponse[]> SendActivitiesAsync(ITurnContext turnContext, Activity[] activities, CancellationToken cancellationToken)
    {
        foreach (var activity in activities)
        {
            _response.Add(activity);
        }

        return Task.FromResult(new ResourceResponse[0]);
    }

    #region Not Implemented
    public override Task DeleteActivityAsync(ITurnContext turnContext, ConversationReference reference, CancellationToken cancellationToken)
    {
        throw new NotImplementedException();
    }

    public override Task<ResourceResponse> UpdateActivityAsync(ITurnContext turnContext, Activity activity, CancellationToken cancellationToken)
    {
        throw new NotImplementedException();
    }
    #endregion
}
```

## Интеграция

Теперь нам осталось собрать все составляющие части и подключить их к существующим компонентам платформы. Основной цикл повтора размещен в функции IBot OnTurn. Он содержит пользовательскую реализацию IStore, которую для мы выполнили в виде внедряемой зависимости в целях тестирования. Весь код размещаемого диалога находится в классе с именем DialogHost, который предоставляет одну общедоступную статическую функцию. Эта функция принимает входящее действие и старое значение состояния, а затем возвращают результирующие действия и новое состояние.

В этой функции сначала создается пользовательский объект BotAdapter, который мы описали выше. Затем мы выполняем диалог точно так же, как обычно, создавая DialogSet и DialogContext и реализуя обычный поток с использованием Continue или Begin. Нам осталось рассмотреть только один компонент — пользовательский метод доступа Accessor. По сути, это очень простая оболочка совместимости, которая упрощает передачу состояния диалога в нашу систему обработки. Метод доступа использует семантику ref при работе с системой обработки диалога. Поэтому ему достаточно передать дескриптор. Чтобы рабочий поток был максимально простым, используемый шаблон класса

поддерживает только семантику ref.

Мы соблюдаем крайнюю осторожность при расположении слоев. Метод JsonSerializer мы добавим прямо в код размещения, так как не хотим выносить его в подключаемый слой хранилища, разные реализации которого могут использовать разные методы сериализации.

Ниже приведен полный код драйвера.

DialogHost.cs

```

public static class DialogHost
{
    // The serializer to use. Moving the serialization to this layer will make the storage layer more
    // pluggable.
    private static readonly JsonSerializer StateJsonSerializer = new JsonSerializer() { TypeNameHandling =
    TypeNameHandling.All };

    /// <summary>
    /// A function to run a dialog while buffering the outbound Activities.
    /// </summary>
    /// <param name="dialog">The dialog to run.</param>
    /// <param name="activity">The inbound Activity to run it with.</param>
    /// <param name="oldState">The existing or old state.</param>
    /// <returns>An array of Activities 'sent' from the dialog as it executed. And the updated or new state.
    </returns>
    public static async Task<(Activity[], JObject)> RunAsync(Dialog dialog, IMessageActivity activity,
    JObject oldState, CancellationToken cancellationToken)
    {
        // A custom adapter and corresponding TurnContext that buffers any messages sent.
        var adapter = new DialogHostAdapter();
        var turnContext = new TurnContext(adapter, (Activity)activity);

        // Run the dialog using this TurnContext with the existing state.
        var newState = await RunTurnAsync(dialog, turnContext, oldState, cancellationToken);

        // The result is a set of activities to send and a replacement state.
        return (adapter.Activities.ToArray(), newState);
    }

    /// <summary>
    /// Execute the turn of the bot. The functionality here closely resembles that which is found in the
    /// IBot.OnTurnAsync method in an implementation that is using the regular BotFrameworkAdapter.
    /// Also here in this example the focus is explicitly on Dialogs but the pattern could be adapted
    /// to other conversation modeling abstractions.
    /// </summary>
    /// <param name="dialog">The dialog to be run.</param>
    /// <param name="turnContext">The ITurnContext instance to use. Note this is not the one passed into the
    IBot OnTurnAsync.</param>
    /// <param name="state">The existing or old state of the dialog.</param>
    /// <returns>The updated or new state of the dialog.</returns>
    private static async Task<JObject> RunTurnAsync(Dialog dialog, ITurnContext turnContext, JObject state,
    CancellationToken cancellationToken)
    {
        // If we have some state, deserialize it. (This mimics the shape produced by BotState.cs.)
        var dialogStateProperty = state?[nameof(DialogState)];
        var dialogState = dialogStateProperty?.ToObject<DialogState>(StateJsonSerializer);

        // A custom accessor is used to pass a handle on the state to the dialog system.
        var accessor = new RefAccessor<DialogState>(dialogState);

        // Run the dialog.
        await dialog.RunAsync(turnContext, accessor, cancellationToken);

        // Serialize the result (available as Value on the accessor), and put its value back into a new
        JObject.
        return new JObject { { nameof(DialogState), JObject.FromObject(accessor.Value, StateJsonSerializer) } };
    }
}

```

И, наконец, пользовательский метод доступа реализует только метод Set, так как состояние передается по ссылке:

RefAccessor.cs

```
public class RefAccessor<T> : IStatePropertyAccessor<T>
    where T : class
{
    public RefAccessor(T value)
    {
        Value = value;
    }

    public T Value { get; private set; }

    public string Name => nameof(T);

    public Task<T> GetAsync(ITurnContext turnContext, Func<T> defaultValueFactory = null, CancellationToken cancellationToken = default(CancellationToken))
    {
        if (Value == null)
        {
            if (defaultValueFactory == null)
            {
                throw new KeyNotFoundException();
            }

            Value = defaultValueFactory();
        }

        return Task.FromResult(Value);
    }

    #region Not Implemented
    public Task DeleteAsync(ITurnContext turnContext, CancellationToken cancellationToken = default(CancellationToken))
    {
        throw new NotImplementedException();
    }

    public Task SetAsync(ITurnContext turnContext, T value, CancellationToken cancellationToken = default(CancellationToken))
    {
        throw new NotImplementedException();
    }
    #endregion
}
```

## Дополнительные сведения

Исходный код примеров для этой статьи на языке [C#](#) можно найти на сайте GitHub.

# Использование шаблонов создания речи в боте

27.03.2021 • 12 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Шаблоны создания речи позволяют разработчикам ботов легко передавать разнообразные сообщения и мультимедиа пользователям. В этой статье показано, как использовать шаблоны создания речи для отправки простых текстовых сообщений и карт, а также как оценивать введенный пользователями текст.

## Предварительные требования

- Учетная запись [LUIS](#).
- Копия примера основного бота создания речи на языке [C#](#) или [JavaScript](#).
- Знание основ [работы ботов и создания речи](#).

## Сведения о примере

Этот пример основного бота создания речи реализует логику приложения для бронирования авиабилетов. С помощью службы LUIS он распознает пользовательский ввод и возвращает наиболее вероятное из обнаруженных LUIS намерений.

В этой статье при использовании шаблонов создания речи в ботах применяется принцип "снизу вверх". Вы научитесь:

- создавать объект `Templates` и [добавлять ссылки на шаблоны в логику бота](#);
- создавать [шаблон простого ответа](#);
- создавать [шаблон условного ответа](#);
- создавать [шаблон карт](#);
- добавлять [функции LUIS в бот](#);
- [тестировать бот](#).

## Вызов шаблонов в файлах

Чтобы использовать шаблоны в LG-файлах, необходимо добавить ссылки на них в логику бота. В приведенных ниже инструкциях показано, как создать ссылки на шаблоны создания речи в основной логике бота, загрузив их в объект `Templates`. В этом примере используются шаблоны из файла `welcomeCard.lg`.

- [C#](#)
- [JavaScript](#)

Убедитесь, что у вас установлен пакет `Microsoft.Bot.Builder.LanguageGeneration`. Добавьте указанный ниже фрагмент кода для загрузки этого пакета.

`Bots/DialogAndWelcomeBot.cs`

```
using Microsoft.Bot.Builder.LanguageGeneration;
```

После загрузки пакета создайте частный объект `Templates _templates`.

```
private Templates _templates;
```

Объект `_templates` используется для ссылки на шаблоны в LG-файлах.

Объедините путь для кроссплатформенной поддержки и проанализируйте путь, содержащий `welcomeCard.lg`, добавив в код приведенный ниже фрагмент.

```
string[] paths = { ".", "Resources", "welcomeCard.lg" };
string fullPath = Path.Combine(paths);
_templates = Templates.ParseFile(fullPath);
```

Теперь можно ссылаться на шаблоны из `welcomeCard.lg` по имени, как показано ниже.

```
foreach (var member in membersAdded)
{
    // Greet anyone that was not the target (recipient) of this message.
    // To learn more about Adaptive Cards, see https://aka.ms/msbot-adaptivecards for more details.
    if (member.Id != turnContext.Activity.Recipient.Id)
    {
        await turnContext.SendActivityAsync(ActivityFactory.FromObject(_templates.Evaluate("WelcomeCard",
actions)));
    }
}
```

Обратите внимание на то, как шаблон `WelcomeCard` указан в вызове `SendActivityAsync()`.

Теперь, когда бот может ссылаться на шаблоны, пора создавать шаблоны в LG-файлах. С помощью создания речи можно легко разнообразить общение.

## Создание шаблона простого ответа

Шаблон простого ответа может содержать один или несколько вариантов текста, используемых для составления и расширения ответов. Один из доступных вариантов выбирается случайным образом библиотекой создания речи.

Шаблоны простого ответа в `BookingDialog.lg`, такие как `PromptForDestinationCity`, `PromptForDepartureCity` И `ConfirmPrefix`, увеличивают разнообразие запросов на бронирование авиабилетов.

### Resources/BookingDialog.lg

```
# PromptForDestinationCity
- Where would you like to travel to?
- What is your destination city?
```

Например, при вызове `PromptForDepartureCity`, показанном выше, будет отображаться один из двух возможных текстовых запросов:

- *Where would you like to travel to?*(Куда вам нужно отправиться?)
- *What is your destination city?*(В какой город вам нужно?)

### Ссылки на память

Как и более сложные шаблоны, шаблоны простого ответа могут ссылаться на память. В `BookingDialog.LG` простой шаблон ответа `ConfirmMessage` ссылается на свойства `Destination`, `Origin` И `TravelDate`.

## Resources/BookingDialog.ig

```
# ConfirmMessage
- I have you traveling to: ${Destination} from: ${Origin} on: ${TravelDate}
- on ${TravelDate}, travelling from ${Origin} to ${Destination}
```

Если пользователь введет значение *Seattle* для `Origin`, значение *Paris* для `Destination` и значение *05/24/2020* для `TravelDate`, то бот выдаст один из следующих результатов.

- *I have you traveling to: Paris from: Seattle on: 05/24/2020*
- *on 05/24/2020, travelling from Seattle to Paris*

## Создание шаблона условного ответа

Шаблон [условного ответа](#) позволяет создавать содержимое, выбираемое на основе условия. Эти условия выражаются с помощью [адаптивных выражений](#).

Шаблон `PromptForMissingInformation` в `BookingDialog.ig` — пример [шаблона IF-ELSE](#). Шаблон IF-ELSE позволяет создать шаблон, который выбирает коллекцию на основе каскадной структуры условий. В шаблоне пользователю предлагается вводить фрагменты информации, если их свойства имеют значение `null`.

## Resources/BookingDialog.ig

```
# PromptForMissingInformation
- IF: ${Destination == null}
  - ${PromptForDestinationCity()}
- ELSEIF: ${Origin == null}
  - ${PromptForDepartureCity()}
- ELSEIF: ${TravelDate == null}
  - ${PromptForTravelDate()}
- ELSE:
  - ${ConfirmBooking()}
```

Если свойство имеет значение `NULL`, то бот вызывает шаблон, связанный с этим свойством. Если все свойства имеют значения, отличные от `NULL`, то вызывается шаблон `ConfirmBooking`.

### Ссылки на другие шаблоны

Варианты в шаблонах могут ссылаться на другие шаблоны. Рассмотрим приведенный выше пример. Если свойство имеет значение `null`, то шаблон вызывает соответствующий шаблон для запроса отсутствующих сведений.

Например, если `Destination` имеет значение `null`, то будет вызван шаблон `PromptforDestinationCity` с помощью  `${PromptForDestinationCity() }` для получения сведений о пункте назначения рейса. Если ни одно из свойств не равно `NULL`, шаблон вызывает запрос `ConfirmBooking`.

## Создание шаблона карт

Шаблоны создания речи могут использовать карты и мультимедиа для расширения возможностей общения. В `welcomeCard.ig` используются четыре шаблона для создания [адаптивной карты](#), которая отображается при первом запуске бота.

`Adaptive Card` определяет объект JSON адаптивной карты.

## Resources/welcomeCard.ig

```

# AdaptiveCard
```
{
  "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",
  "type": "AdaptiveCard",
  "version": "1.0",
  "body": [
    {
      "type": "Image",
      "url": "https://encrypted-tbn0.gstatic.com/images?
q=tbn:ANd9GcQtB3AwMUeNoq4gUBGe60cj8kyh3bXa9ZbV7u1fVKQoyKFHdkqU",
      "size": "stretch"
    },
    {
      "type": "TextBlock",
      "spacing": "medium",
      "size": "default",
      "weight": "bolder",
      "text": "${HeaderText()}",
      "wrap": true,
      "maxLines": 0
    },
    {
      "type": "TextBlock",
      "size": "default",
      "isSubtle": true,
      "text": "Now that you have successfully run your bot, follow the links in this Adaptive Card to expand
your knowledge of Bot Framework.",
      "wrap": true,
      "maxLines": 0
    }
  ],
  "actions": [
    ${join(foreach(actions, item, cardActionTemplate(item.title, item.url, item.type)), ',')}
  ]
}
```

```

Эта карта содержит изображение и использует шаблоны создания речи для отображения в заголовке карты набора предлагаемых действий.

- [C#](#)
- [JavaScript](#)

`actions` заполняется путем вызова `cardActionTemplate(title, url, type)` и получения `title`, `url` и `type` из метода `OnMembersAddedAsync()` в `DialogAndWelcomeBot.cs`.

[Bots/DialogAndWelcomeBot.cs](#)

```
// Actions to include in the welcome card. These are passed to LG and are then included in the generated Welcome card.
var actions = new {
    actions = new List<Object>() {
        new {
            title = "Get an overview",
            url = "https://docs.microsoft.com/en-us/azure/bot-service/?view=azure-bot-service-4.0"
        },
        new {
            title = "Ask a question",
            url = "https://stackoverflow.com/questions/tagged/botframework"
        },
        new {
            title = "Learn how to deploy",
            url = "https://docs.microsoft.com/en-us/azure/bot-service/bot-builder-howto-deploy-azure?view=azure-bot-service-4.0"
        }
    }
}
```

`title` — это текст на кнопке предлагаемого действия, а `url` — URL-адрес, открываемый при нажатии этой кнопки.

Наконец, `WelcomeCard` вызывает шаблон `AdaptiveCard`, чтобы получить объект JSON адаптивной карты.

### Resources/welcomeCard.lg

```
# WelcomeCard
[Activity
    Attachments = ${json(AdaptiveCard())}
]
```

Чтобы узнать больше о функции `ActivityAttachment()`, ознакомьтесь с [внедрением функций из библиотеки создания речи](#).

## Добавление функций LUIS в бот

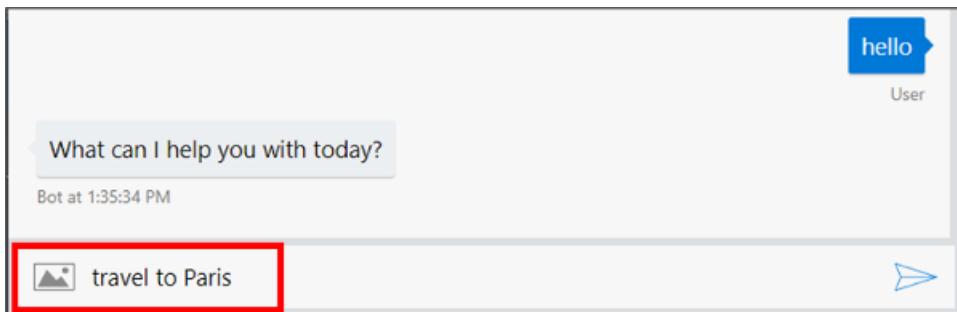
После обновления логики бота и шаблонов создания речи вы можете добавить в бот функции LUIS. Выполните инструкции в следующих разделах, чтобы добавить функции LUIS в бот.

- [Создание приложения LUIS на портале LUIS](#)
- [Получение сведений о приложении на портале LUIS](#)
- [Обновление файла параметров бота](#)

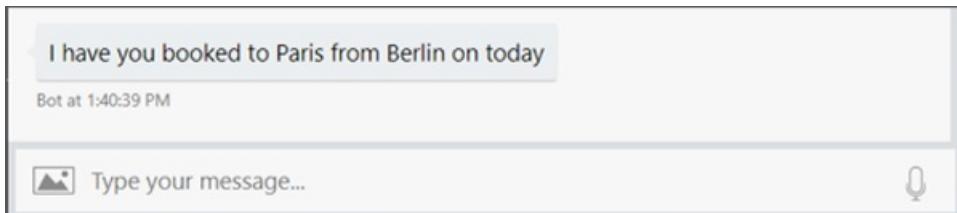
## Тестирование бота

Скачайте и установите последнюю версию [Bot Framework Emulator](#).

1. Выполните этот пример на локальном компьютере. Дополнительные инструкции, включая примеры для [C#](#), [JS](#) или [Python](#), см. в файле README.
2. В эмуляторе введите сообщение, например "Путешествие в Париж" или "переход от Париж к Берлин". Используйте все речевые фрагменты, включенные в файл FlightBooking.json, для обучения намерения Book flight (Бронирование авиабилетов).



Если наиболее вероятное намерение LUIS соответствует элементу Book flight (Бронирование авиабилетов), бот будет задавать дополнительные вопросы, пока не получит достаточно сохраненных сведений для создания бронирования. После этого он возвращает всю собранную информацию о бронировании пользователю.



На этом этапе логика кода обнуляет состояние бота, и вы можете создать новое резервирование.

## Дополнительные сведения

- [шаблоны структурированного ответа](#)
- [Формат файлов LG](#)
- [Справочные материалы по встроенным функциям адаптивных выражений](#)

# Использование настраиваемых функций при создании текста

27.03.2021 • 11 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Разработчики могут использовать как [встроенные функции](#), поддерживаемые адаптивными выражениями, так и пользовательские функции в [шаблонах создания языка \(LG\)](#). В этой статье показано, как добавить пользовательскую функцию в программу Bot для адаптивных выражений и использовать функцию в шаблоне LG.

## Предварительные условия

- Знание [основ Bot, адаптивных выражений, формирования языкаи формата LG-файла](#). Также полезно ознакомиться с предварительно [созданными функциями](#).
- Копия примера пользовательских функций в [C#](#) или [JavaScript](#).

## Сведения о примере

Этот пример пользовательских функций LG является примером добавления простой пользовательской функции к адаптивным выражениям и последующего использования этого выражения в шаблоне LG. Программа-робот запрашивает у пользователя число, а если входные данные допустимы, возвращает квадратный корень. Функция, которая вычисляет квадратный корень, `contoso.sqrt`, определяется в логике Bot и используется в шаблоне LG, который создает ответы Bot.

В этой статье используется подход снизу вверх для добавления и использования пользовательских функций в шаблонах LG. Вы узнаете, как:

- Добавление [пакетов](#), необходимых для использования адаптивных выражений и LG в Bot
- [Добавление пользовательской функции в адаптивные выражения](#) в логике Bot
- [Использование пользовательской функции в шаблоне LG](#)
- [Загрузка и использование шаблона в Bot](#)
- [Тестирование ленты](#)

## Пакеты

- [C#](#)
- [JavaScript](#)

Чтобы использовать адаптивные выражения и LG, установите пакеты Microsoft. Bot. Builder. [лангажеженератион](#) и [адаптивикспрессионс](#). В примере уже установлен пакет. Затем добавьте следующий фрагмент кода в основной файл программы Bot.

## Программы-роботы/Кастомфункционбот. CS

```
using Microsoft.Bot.Builder.LanguageGeneration;
using AdaptiveExpressions;
```

# Добавление пользовательской функции в адаптивные выражения

Чтобы использовать пользовательские функции в Bot, их необходимо добавить в адаптивные выражения. В этом разделе показано, как добавить пользовательскую функцию с именем `contoso.sqrt` в адаптивные выражения.

- [C#](#)
- [JavaScript](#)

Начните с добавления строковой константы с именем пользовательской функции в конструктор Bot. Имя функции должно быть кратким, но распознаваемым. В этом примере пользовательская функция называется `contoso.sqrt`:

## Программы-роботы/Кустомфункционбот. CS

```
const string mySqrtFnName = "contoso.sqrt";
```

### IMPORTANT

Передайте функции, чтобы избежать конфликтов пространств имен.

В имени функции `contoso` является префиксом и `sqrt` является коротким элементом для квадратного корня, возвращаемого функцией.

Теперь можно определить логику функции в конструкторе Bot и добавить ее в адаптивные выражения с помощью `Expression.Functions.Add()` функции. Добавление пользовательской функции в Адаптивное выражение позволяет использовать функцию в шаблонах LG, как и в случае с любой из готовых функций.

В приведенном ниже фрагменте кода показано, как добавить функцию, определенную как `mySqrtFnName`, в адаптивные выражения. Эта функция возвращает квадратный корень одного аргумента, `args`, если он допустим, и `null`. Если нет:

```
// Add custom sqrt function
Expression.Functions.Add(mySqrtFnName, (args) =>
{
    object retVal = null;
    if (args[0] != null)
    {
        double dblValue;
        if (double.TryParse(args[0], out dblValue))
        {
            retVal = Math.Sqrt(dblValue);
        }
    }
    return retVal;
});
```

## Использование пользовательской функции в шаблоне LG

После добавления пользовательской функции к адаптивным выражениям ее можно использовать в шаблонах LG. В этом разделе описывается настройка ответа Bot в зависимости от того, были ли введены данные пользователем.

- [C#](#)

- [JavaScript](#)

В Main. LG есть два определения шаблонов: [простой шаблон ответа](#) `sqrtReadBack` и [шаблон условного отклика](#) `sqrtTemplate`:

### Resources/Main. LG

```
# sqrtReadBack
- You said '${text}'. ${sqrtTemplate()}.
```

Этот шаблон создает ответ, содержащий введенные пользователем данные, `${text}` и результат второго шаблона условного [if-else](#) `sqrtTemplate`:

```
# sqrtTemplate
- IF : ${contoso.sqrt(text) != null}
  - It's square root is ${coalesce(contoso.sqrt(text), 'NaN')}
- ELSE :
  - Sqrt not possible.
```

В этом шаблоне результат `contoso.sqrt(text)` используется для определения ответа:

- Если результат не равен `null`, то строка под `IF` блоком используется в `sqrtReadBack`. Обратите внимание, что в этой строке используется встроенное выражение `${coalesce(contoso.sqrt(text), 'NaN')}`. Встроенная функция [объединения](#) возвращает результат, `contoso.sqrt(text)` если он не равен `null`, и `NaN` если он равен `null`.
- Если результат равен `null`, то строка под `ELSE` блоком используется в `sqrtReadBack`.

## Загрузка и использование шаблона LG в Bot

Теперь, когда шаблон ответа Bot создан `sqrtReadBack` в Main. LG, можно использовать шаблон в логике Bot.

- [C#](#)
- [JavaScript](#)

Для начала создайте закрытый объект с `Templates` именем `_templates`.

### Программы-роботы/Кастомфункционбот. CS

```
protected Templates _templates;
```

`_templates` Объект используется для ссылки на шаблоны в LG-файлах.

Затем объедините путь для кросс-платформенной поддержки. Обязательно включите Main. LG , добавив следующее:

```
var lgFilePath = Path.Join(Directory.GetCurrentDirectory(), "Resources", "main.lg");
```

Теперь можно выполнить синтаксический анализ файлов в `lgFilePath` и загрузить шаблоны LG, как показано ниже. По умолчанию используется `Expression.Function`, который включает пользовательскую функцию.

```
_templates = Templates.ParseFile(lgFilePath);
```

Теперь ваши шаблоны загружены, и вы можете ссылаться на них по имени в работе. В этом примере результат вычисления `sqrtReadBack` используется в качестве `replyText` отправленного пользователю.

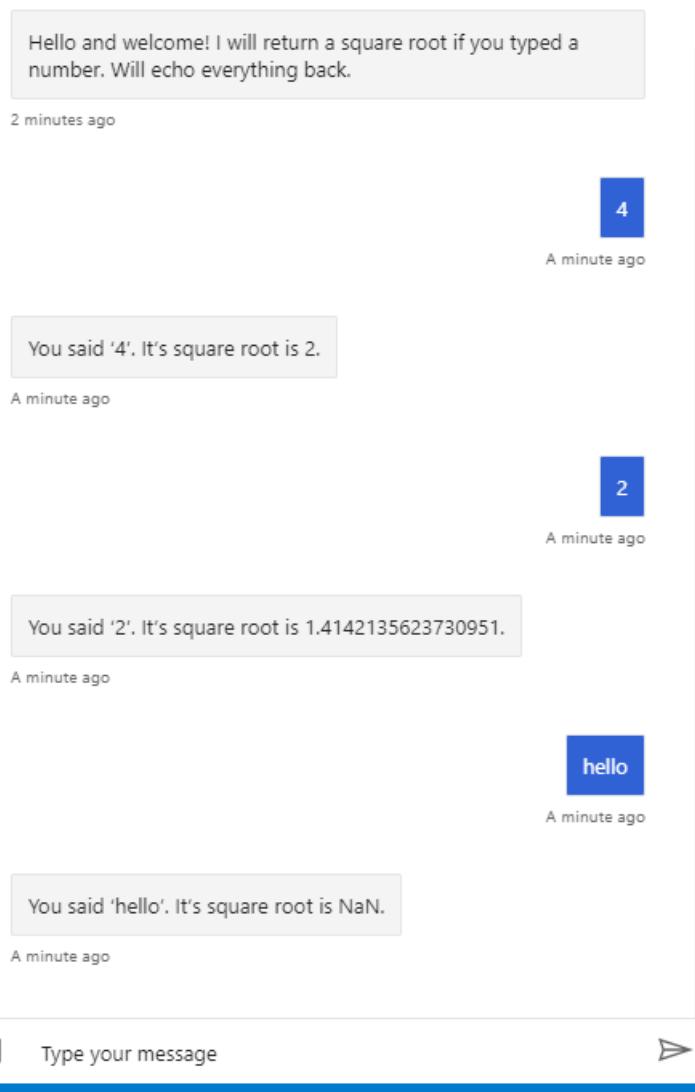
```
protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    var replyText = _templates.Evaluate("sqrtReadBack", turnContext.Activity);
    await turnContext.SendActivityAsync(ActivityFactory.FromObject(replyText), cancellationToken);
}
```

Теперь вы готовы к тестированию программы Bot.

## Тестирование бота

Скачайте и установите последнюю версию [эмулятора Bot Framework](#).

1. Выполните этот пример на локальном компьютере. Если вам нужны инструкции, ознакомьтесь с файлом README для примера [C#](#) или [JavaScript](#).
2. В эмуляторе введите что угодно. Вы увидите, что эмулятор вернет квадратный корень из числа введенных и `Nan` для всех остальных входных данных.



## Дополнительные сведения

- [шаблоны структурированного ответа](#)

- Формат файлов LG
- Справочные материалы по встроенным функциям адаптивных выражений

# Реализация навыка

27.03.2021 • 19 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Вы можете использовать навыки для расширения функциональности другого бота. *Навыком* здесь называется бот, который может выполнять ряд задач для другого бота.

- Интерфейс навыка описывается манифестом. Разработчики, у которых нет доступа к исходному коду навыка, могут использовать информацию из этого манифеста для разработки потребителя навыка.
- Навык может использовать проверку утверждений для управления доступом от других ботов и пользователей.

В этой статье демонстрируется реализация навыка, который выводит на экран вводимые пользователем данные.

## Предварительные требования

- [Базовые знания о ботах и навыках](#).
- Подписка Azure (для развертывания навыков). Если у вас еще нет подписки Azure, создайте [бесплатную учетную запись](#), прежде чем начать работу.
- Копия примера [простого навыка для ботов](#) для языка [C#](#), [JavaScript](#) или [Python](#).

### NOTE

Начиная с версии 4.11 для проверки навыка локально в эмуляторе не требуетсяся идентификатор приложения и пароль. Подписка Azure по-прежнему необходима для развертывания вашего навыка в Azure.

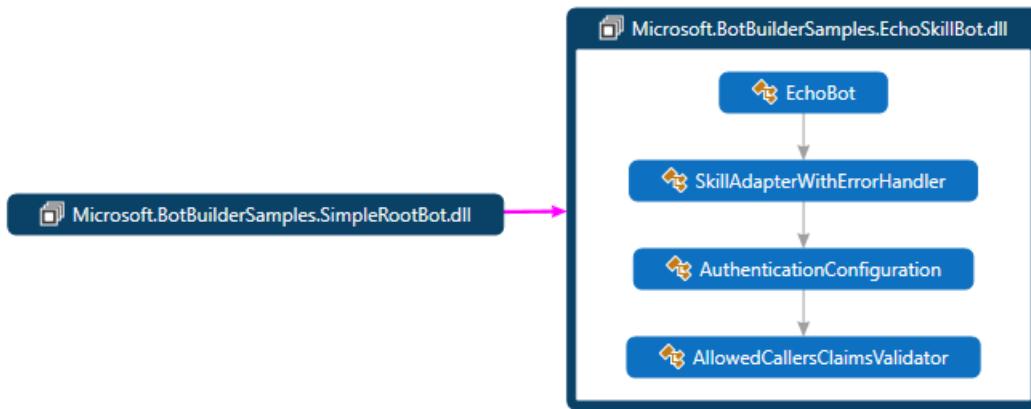
## Об этом примере

В пример [простого навыка для ботов](#) включены проекты двух ботов:

- *бот эхо-навыка*, который реализует этот навык;
- *простой корневой бот*, который реализует бот для использования этого навыка.

Эта статья посвящена навыку, в том числе логике поддержки в боте и адаптере.

- [C#](#)
- [JavaScript](#)
- [Python](#)



Сведения о простом корневом боте см. в статье [о реализации потребителя навыка](#).

## Ресурсы

Для развернутого программы-роботы проверка подлинности «Bot-to-Bot» требует, чтобы каждый участвующий робот получил допустимые идентификатор приложения и пароль. Тем не менее вы можете проверить навыки и квалификацию потребителей локально с помощью эмулятора без идентификатора приложения и пароля.

Чтобы сделать навык доступным для программы-роботы пользователей, зарегистрируйте навык в Azure. Для этого можно применить службу "Регистрация каналов бота". Дополнительные сведения см. в статье [о регистрации бота в службе Azure Bot](#).

## Конфигурация приложений

При необходимости добавьте идентификатор и пароль приложения навыка в файл конфигурации навыка. (Если потребитель или навык используют идентификатор приложения и пароль, оба должны быть.)

Массив *allowed callers* позволяет ограничить круг потребителей, имеющих доступ к этому навыку. Добавьте элемент "\*", чтобы принимать вызовы от любого потребителя навыков.

### NOTE

Если вы тестируете свой навык локально без идентификатора и пароля приложения, ни навык, ни потребитель опыта не выполняют код для проверки утверждения.

- [C#](#)
- [JavaScript](#)
- [Python](#)

### EchoSkillBot\appsettings.json

Добавьте в файл appsettings.json идентификатор приложения и пароль для этого навыка.

```
{
    "MicrosoftAppId": "",
    "MicrosoftAppPassword": "",

    // This is a comma separate list with the App IDs that will have access to the skill.
    // This setting is used in AllowedCallersClaimsValidator.
    // Examples:
    //     [ "*" ] allows all callers.
    //     [ "AppId1", "AppId2" ] only allows access to parent bots with "AppId1" and "AppId2".
    "AllowedCallers": [ "*" ]
}
```

## Логика обработчика действий

### Прием входных параметров

Потребитель навыка может отправить информацию в этот навык. Один из способов принять такую информацию — через свойство *value* входящего сообщения. Другой способ — обработка действий события и вызова.

В нашем примере навык не принимает входные параметры.

### Продолжение или завершение беседы

Когда навык отправляет действие, потребитель навыка должен перенаправить это действие пользователю.

Но при завершении работы навыка необходимо отправить действие `endOfConversation`, иначе потребитель навыка будет и далее пересыпать действия пользователя в навык. Можно также поместить возвращаемое значение в свойство *value*, а причину завершения навыка — в свойство *code* обрабатываемого действия.

- [C#](#)
- [JavaScript](#)
- [Python](#)

### EchoSkillBot\Bots\EchoBot.cs

```
protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    if (turnContext.Activity.Text.Contains("end") || turnContext.Activity.Text.Contains("stop"))
    {
        // Send End of conversation at the end.
        var messageText = $"ending conversation from the skill...";
        await turnContext.SendActivityAsync(MessageFactory.Text(messageText, messageText,
InputHints.IgnoringInput), cancellationToken);
        var endOfConversation = Activity.CreateEndOfConversationActivity();
        endOfConversation.Code = EndOfConversationCodes.CompletedSuccessfully;
        await turnContext.SendActivityAsync(endOfConversation, cancellationToken);
    }
    else
    {
        var messageText = $"Echo: {turnContext.Activity.Text}";
        await turnContext.SendActivityAsync(MessageFactory.Text(messageText, messageText,
InputHints.IgnoringInput), cancellationToken);
        messageText = "Say \"end\" or \"stop\" and I'll end the conversation and back to the parent.";
        await turnContext.SendActivityAsync(MessageFactory.Text(messageText, messageText,
InputHints.ExpectingInput), cancellationToken);
    }
}
```

## Отмена выполнения навыка

В навыках с несколькими репликами необходимо также принимать от потребителя навыков действия `endOfConversation`, которые позволяют потребителю отменить текущую беседу.

Логика навыка в нашем примере не меняется при последовательных репликах. Если реализуемый вами навык выделяет ресурсы для беседы, добавьте в обработчик завершения беседы код для очистки этих ресурсов.

- [C#](#)
- [JavaScript](#)
- [Python](#)

### EchoSkillBot\Bots\EchoBot.cs

```
protected override Task OnEndOfConversationActivityAsync(ITurnContext<IEndOfConversationActivity>
turnContext, CancellationToken cancellationToken)
{
    // This will be called if the root bot is ending the conversation. Sending additional messages should
    be
    // avoided as the conversation may have been deleted.
    // Perform cleanup of resources if needed.
    return Task.CompletedTask;
}
```

## Средство проверки утверждений

В нашем примере для проверки утверждений используется список разрешенных вызывающих объектов. Этот список определен в файле конфигурации навыка и считывается в объект средства проверки при его создании.

Необходимо добавить *проверяющий элемент управления утверждения* в конфигурацию проверки подлинности. Утверждения оцениваются после заголовка проверки подлинности. Код средства проверки должен создавать исключение или ошибку, чтобы отклонить запрос. Отклонение запроса, уже прошедшего проверку подлинности, может потребоваться по разным причинам. Пример:

- навык предоставляется как часть платной службы; Пользователь не должен иметь доступ к базе данных.
- доступ к навыку ограничен правообладателем; вызвать навык могут только определенные пользователи.

### IMPORTANT

Если вы не предложите проверяющий элемент управления утверждениями, программа Bot создаст ошибку или исключение при получении действия от потребителя навыков.

- [C#](#)
- [JavaScript](#)
- [Python](#)

Создайте класс средства проверки утверждений, производный от класса `ClaimsValidator`. Для отклонения входящего запроса создается исключение `UnauthorizedAccessException`. Обратите внимание, что метод `IConfiguration.Get` возвращает значение NULL, если значение в файле конфигурации является пустым массивом.

### EchoSkillBot\Authentication\AllowedCallersClaimsValidator.cs

```

public class AllowedCallersClaimsValidator : ClaimsValidator
{
    private const string ConfigKey = "AllowedCallers";
    private readonly List<string> _allowedCallers;

    public AllowedCallersClaimsValidator(IConfiguration config)
    {
        if (config == null)
        {
            throw new ArgumentNullException(nameof(config));
        }

        // AllowedCallers is the setting in the appsettings.json file
        // that consists of the list of parent bot IDs that are allowed to access the skill.
        // To add a new parent bot, simply edit the AllowedCallers and add
        // the parent bot's Microsoft app ID to the list.
        // In this sample, we allow all callers if AllowedCallers contains an "*".
        var section = config.GetSection(ConfigKey);
        var appsList = section.Get<string[]>();
        if (appsList == null)
        {
            throw new ArgumentNullException($"\"{ConfigKey}\" not found in configuration.");
        }

        _allowedCallers = new List<string>(appsList);
    }

    public override Task ValidateClaimsAsync(IList<Claim> claims)
    {
        // If _allowedCallers contains an "*", we allow all callers.
        if (SkillValidation.IsSkillClaim(claims) && !_allowedCallers.Contains("*"))
        {
            // Check that the appId claim in the skill request is in the list of callers configured for this
            bot.
            var appId = JwtTokenValidation.GetAppIdFromClaims(claims);
            if (!_allowedCallers.Contains(appId))
            {
                throw new UnauthorizedAccessException($"Received a request from a bot with an app ID of \"
{appId}\". To enable requests from this caller, add the app ID to your configuration file.");
            }
        }

        return Task.CompletedTask;
    }
}

```

## Адаптер навыка

При возникновении ошибки адаптер навыка должен очистить состояние беседы для этого навыка и отправить потребителю действие `endOfConversation`. Используйте свойство `code` в этом действии, чтобы сообщить о завершении навыка из-за ошибки.

- [C#](#)
- [JavaScript](#)
- [Python](#)

[EchoSkillBot\SkillAdapterWithErrorHandler.cs](#)

```

public SkillAdapterWithErrorHandler(IConfiguration configuration, ICredentialProvider credentialProvider,
AuthenticationConfiguration authConfig, ILogger<BotFrameworkHttpAdapter> logger)
    : base(configuration, credentialProvider, authConfig, logger: logger)
{
    _logger = logger ?? throw new ArgumentNullException(nameof(logger));
    OnTurnError = HandleTurnError;
}

private async Task HandleTurnError(ITurnContext turnContext, Exception exception)
{
    // Log any leaked exception from the application.
    _logger.LogError(exception, $"[OnTurnError] unhandled error : {exception.Message}");

    await SendErrorMessageAsync(turnContext, exception);
    await SendEoCToParentAsync(turnContext, exception);
}

private async Task SendErrorMessageAsync(ITurnContext turnContext, Exception exception)
{
    try
    {
        // Send a message to the user.
        var errorMessageText = "The skill encountered an error or bug.";
        var errorMessage = MessageFactory.Text(errorMessageText, errorMessageText,
InputHints.IgnoringInput);
        await turnContext.SendActivityAsync(errorMessage);

        errorMessageText = "To continue to run this bot, please fix the bot source code.";
        errorMessage = MessageFactory.Text(errorMessageText, errorMessageText, InputHints.ExpectingInput);
        await turnContext.SendActivityAsync(errorMessage);

        // Send a trace activity, which will be displayed in the Bot Framework Emulator.
        // Note: we return the entire exception in the value property to help the developer;
        // this should not be done in production.
        await turnContext.TraceActivityAsync("OnTurnError Trace", exception.ToString(),
"https://www.botframework.com/schemas/error", "TurnError");
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, $"Exception caught in SendErrorMessageAsync : {ex}");
    }
}

private async Task SendEoCToParentAsync(ITurnContext turnContext, Exception exception)
{
    try
    {
        // Send an EndOfConversation activity to the skill caller with the error to end the conversation,
        // and let the caller decide what to do.
        var endOfConversation = Activity.CreateEndOfConversationActivity();
        endOfConversation.Code = "SkillError";
        endOfConversation.Text = exception.Message;
        await turnContext.SendActivityAsync(endOfConversation);
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, $"Exception caught in SendEoCToParentAsync : {ex}");
    }
}

```

## Регистрация службы

*Адаптер Bot Framework* использует объект *конфигурации проверки подлинности*, который настраивается при создании адаптера, для проверки заголовков проверки подлинности во входящих запросах.

В нашем примере в конфигурацию проверки подлинности добавляется проверка утверждений, а также применяется *адаптер навыка с обработчиком ошибок*, который описан в предыдущем разделе.

- [C#](#)
- [JavaScript](#)
- [Python](#)

### EchoSkillBot\Startup.cs

```
// Register AuthConfiguration to enable custom claim validation.  
services.AddSingleton(sp => new AuthenticationConfiguration { ClaimsValidator = new  
    AllowedCallersClaimsValidator(sp.GetService< IConfiguration>()) });  
  
// Create the Bot Framework Adapter with error handling enabled.  
services.AddSingleton<IBotFrameworkHttpAdapter, SkillAdapterWithErrorHandler>();
```

## Манифест навыка

*Манифест навыка* — это файл в формате JSON с описанием действий, которые может выполнять навык, его входных и выходных параметров, а также конечных точек навыка. Манифест содержит сведения, которые вам нужны для доступа к навыку из другого бота. Последняя версия схемы — [v 2.1](#).

- [C#](#)
- [JavaScript](#)
- [Python](#)

### EchoSkillBot\wwwroot\manifest\echoskillbot-manifest-1.0.json

```
{  
  "$schema": "https://schemas.botframework.com/schemas/skills/skill-manifest-2.0.0.json",  
  "$id": "EchoSkillBot",  
  "name": "Echo Skill bot",  
  "version": "1.0",  
  "description": "This is a sample echo skill",  
  "publisherName": "Microsoft",  
  "privacyUrl": "https://echoskillbot.contoso.com/privacy.html",  
  "copyright": "Copyright (c) Microsoft Corporation. All rights reserved.",  
  "license": "",  
  "iconUrl": "https://echoskillbot.contoso.com/icon.png",  
  "tags": [  
    "sample",  
    "echo"  
,  
  ],  
  "endpoints": [  
    {  
      "name": "default",  
      "protocol": "BotFrameworkV3",  
      "description": "Default endpoint for the skill",  
      "endpointUrl": "http://echoskillbot.contoso.com/api/messages",  
      "msAppId": "00000000-0000-0000-0000-000000000000"  
    }  
  ]  
}
```

*Схема манифеста навыка* представляет собой файл в формате JSON с описанием схемы манифеста навыков. Текущая версия схемы — [2.1.0](#).

## Тестирование навыка

На этом этапе вы можете проверить работу навыка в эмуляторе, как для любого обычного бота. Но прежде, чем протестировать его как навык, необходимо [реализовать потребитель навыка](#).

Скачайте и установите последнюю версию [Bot Framework Emulator](#).

1. Запустите бот с эхо-навыком на локальном компьютере. Дополнительные инструкции для [C#](#), [JavaScript](#) и [Python](#) см. в файле сведений соответствующего примера.
2. Примените эмулятор для тестирования бота, как показано ниже. Обратите внимание, что при отправке навыку сообщения "end" или "stop" он дополняет ответное сообщение действием `endOfConversation`. Навык отправляет действие `endOfConversation`, чтобы обозначить завершение своей работы.

http://localhost:39783/api/messages

The screenshot shows a conversation in the Bot Framework Emulator. The user sends a message "hi". The bot replies with "Echo (dotnet) : hi". The user then sends "end". The bot replies with "ending conversation from the skill...". On the right side, there is a log panel showing the following activity:

```
[10:35:18] POST 200 directline.conversationUpdate
[10:35:18] POST 201 directline.conversations
[10:35:18] Emulator listening on http://[::]:59293
[10:35:18] ngrok not configured (only needed when connecting to remotely hosted bots)
[10:35:18] Connecting to bots hosted remotely
[10:35:18] Edit ngrok settings
[10:35:30] -> message hi
[10:35:31] <- message Echo (dotnet) : hi
[10:35:31] POST 200 conversations.:conversationId.activities.:activityId
[10:35:31] <- message Say "end" or "stop" and I'll end the conversation ...
[10:35:31] POST 200 conversations.:conversationId.activities.:activityId
[10:35:31] POST 200 directline.conversations.:conversationId.activities
[10:35:33] -> message end
[10:35:33] <- message ending conversation from the skill...
[10:35:33] POST 200 conversations.:conversationId.activities.:activityId
[10:35:33] <- endOfConversation
[10:35:33] POST 200 conversations.:conversationId.activities.:activityId
[10:35:33] POST 200 directline.conversations.:conversationId.activities
```

Click on a log item in the panel below to inspect activity.  
You can also inspect the JSON responses from your LUIS and QnA Maker services by selecting a "trace" activity. [Learn More](#).

## Дальнейшие действия

[Реализация навыка в Power Virtual Agents](#)

# Как создать манифест навыка версии 2.1

27.03.2021 • 14 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

*Манифест навыка* — это файл в формате JSON с описанием действий, которые может выполнять навык, его входных и выходных параметров, а также конечных точек навыка. Манифест содержит сведения, которые нужны разработчику для обращения к навыку из другого бота. При использовании версии 2.1 схемы манифеста навыка манифест может также описывать упреждающие действия, которые может отправлять навык, и используемые этим навыком модели отправки.

В этой статье описывается [версия 2.1](#) схемы манифеста для навыков платформы Bot. Описание версии 2.0 см. в статье [написание манифеста уровня v 2.0](#).

Схема манифеста навыка Bot Framework использует черновик 7 словаря схемы JSON.

## Предварительные требования

- Знания о [навыках и ботах навыка](#).
- Общие знания о [схеме JSON](#) и формате JSON.

## Манифест навыка

Манифест навыка содержит различные категории информации:

- метаданные, описывающие навык на общем уровне;
- список конечных точек, предоставляемых навыком;
- необязательные списки действий, которые навык может получать и упреждающе отправлять;
- необязательный объект определений, содержащий схемы для объектов, на которые ссылается другие части документа;
- необязательный список моделей отправки, поддерживаемых навыком.

Ниже приведена полная схема для версии 2.1 манифеста навыка Bot Framework.

Категория или поле	Тип	Обязательно	Описание
<b>Метаданные</b>			
\$id	строка	Обязательно	Идентификатор манифеста навыка.
\$schema	строка	Обязательно	Универсальный код ресурса (URI) схемы JSON в формате HTTPS, описывающий формат манифеста. Для версии 2.1.0 универсальный код ресурса (URI) — <code>https://schemas.botframework.com/schemas/skillmanifest.json</code>
copyright	строка	Необязательно	Уведомление об авторских правах на навык.
description	строка	Необязательно	Понятное описание навыка.
iconUrl	строка	Необязательно	Универсальный код ресурса (URI) значка, отображаемого для навыка.

Категория или поле	Тип	Обязательно	Описание
license	строка	Необязательно	Лицензионное соглашение для навыка.
name	строка	Обязательно	Имя навыка.
version	строка	Обязательно	Версия навыка, который описывает манифест.
privacyUrl	строка	Необязательно	Универсальный код ресурса (URI) описания конфиденциальности для навыка.
publisherName	строка	Обязательно	Имя издателя навыка.
tags	массив строк	Необязательно	Набор тегов для навыка. Если он указан, то каждый тег должен быть уникальным.
<b>Конечные точки</b>			
конечные точки	Массив <a href="#">конечных точек</a>	Обязательно	Список конечных точек, поддерживаемых навыком. Необходимо определить хотя бы одну конечную точку. Имя каждой конечной точки должно быть уникальным.
<b>Действия</b>			
Действия	Объект, содержащий именованные <a href="#">объекты действий</a>	Обязательно	Набор начальных действий, принимаемых навыком.
activitiesSent	Объект, содержащий именованные <a href="#">объекты действий</a>	Необязательно	Описывает упреждающие действия, которые могут быть отправлены навыком.
<b>Определения</b>			
definitions	объект	Необязательно	Объект, содержащий вложенные схемы для объектов, используемых в манифесте.
<b>Модели отправки</b>			
dispatchModels	Объект <a href="#">dispatchModels</a>	Необязательно	Описание языковых моделей и намерений верхнего уровня, поддерживаемых навыком. <a href="#">Ниже</a> приведена схема для этого объекта.

## Конечные точки

Каждый объект конечной точки описывает конечную точку, поддерживаемую навыком.

В этом примере приведены две конечные точки для навыка.

```

"endpoints": [
  {
    "name": "americas",
    "protocol": "BotFrameworkV3",
    "description": "Production endpoint for SkillBot in the Americas",
    "endpointUrl": "http://myskill.contoso.com/api/messages",
    "msAppId": "00000000-0000-0000-0000-000000000000"
  },
  {
    "name": "eu",
    "protocol": "BotFrameworkV3",
    "description": "Production endpoint for SkillBot in Europe",
    "endpointUrl": "http://myskill.contoso.com/api/messages",
    "msAppId": "11111111-0000-0000-0000-000000000000"
  }
],

```

## Объект endpoint

Описывает конечную точку, поддерживаемую навыком.

ПОЛЕ	ТИП	ОБЯЗАТЕЛЬНО	ОПИСАНИЕ
description	строка	Необязательно	Описание конечной точки.
endpointUrl	строка	Обязательно	Универсальный код ресурса (URI) конечной точки для навыка.
msAppId	строка	Обязательно	Идентификатор Microsoft AppId (GUID) для навыка, используемый для аутентификации запросов.
name	строка	Обязательно	Уникальное имя конечной точки.
protocol	строка	Необязательно	Поддерживаемый протокол. Значение по умолчанию — BotFrameworkV3.

## Действия

Каждый объект действия описывает действие, принимаемое навыком, или действие, которое навык может упреждающе отправить. Для входящих действий навык запустит действие или задачу в зависимости от полученного начального действия. Имя, связанное с объектом действия, указывает действие или задачу, которая будет выполнена навыком.

Некоторые типы действий имеют свойство Value, которое можно использовать, чтобы предоставить дополнительные входные данные для навыка. Когда навык завершает действие, он может предоставить возвращаемое значение в свойстве Value связанного действия завершения диалога.

В схеме манифеста навыка версии 2.1.preview-1 разрешены следующие типы действий: сообщение, событие, вызов и другие действия. Навык может получать действие вызова, но не может его отправлять.

Ниже приведен пример описания действия.

```

"bookFlight": {
  "description": "Books a flight",
  "type": "event",
  "name": "BookFlight",
  "value": {
    "$ref": "#/definitions/bookingInfo"
  },
  "resultValue": {
    "$ref": "#/definitions/bookingInfo"
  }
},

```

### Объект eventActivity

Описывает действие события, принимаемое или отправляемое навыком.

ПОЛЕ	ТИП	ОБЯЗАТЕЛЬНО	ОПИСАНИЕ
description	строка	Необязательно	Описание действия.
name	строка	Обязательно	Значение свойства name действия события.
resultValue	объект	Необязательно	Определение схемы JSON типа объекта, который может вернуть связанное действие.
type	строка	Обязательно	Тип действия. Должен иметь значение "event".
value	объект	Необязательно	Определение схемы JSON типа объекта, получение которого ожидает это действие.

### Объект invokeActivity

Описывает действие вызова, принимаемое навыком.

ПОЛЕ	ТИП	ОБЯЗАТЕЛЬНО	ОПИСАНИЕ
description	строка	Необязательно	Описание действия.
name	строка	Обязательно	Значение свойства name действия вызова.
resultValue	объект	Необязательно	Определение схемы JSON типа объекта, который может вернуть связанное действие.
type	строка	Обязательно	Тип действия. Должен иметь значение быть "invoke".
value	объект	Необязательно	Определение схемы JSON типа объекта, получение которого ожидает это действие.

### Объект messageActivity

Описывает действие сообщения, принимаемое или отправляемое навыком. Свойство text действия сообщения содержит речевой фрагмент пользователя.

ПОЛЕ	ТИП	ОБЯЗАТЕЛЬНО	ОПИСАНИЕ
description	строка	Необязательно	Описание действия.
resultValue	объект	Необязательно	Определение схемы JSON типа объекта, который может вернуть связанное действие.
type	строка	Обязательно	Тип действия. Должен иметь значение "message".
value	объект	Необязательно	Определение схемы JSON типа объекта, получение которого ожидает это действие.

### Объект otherActivities

Описывает действие, принимаемое или отправляемое навыком.

ПОЛЕ	ТИП	ОБЯЗАТЕЛЬНО	ОПИСАНИЕ
type	строка	Обязательно	Тип действия. Должен иметь значение одного из прочих типов действий Bot Framework: "contactRelationUpdate", "conversationUpdate", "deleteUserData", "endOfConversation", "handoff", "installationUpdate", "messageDelete", "messageUpdate", "messageReaction", "suggestion", "trace" или "typing".

Объект otherActivities может содержать и другие свойства, но схема манифеста навыка не определяет их значение.

## Определения

Каждое определение описывает вложенную схему, которую можно использовать в других частях документа.

Ниже приведен пример вложенной схемы для сведений о бронировании авиабилетов.

```
"bookingInfo": {  
    "type": "object",  
    "required": [  
        "origin"  
    ],  
    "properties": {  
        "origin": {  
            "type": "string",  
            "description": "this is the origin city for the flight"  
        },  
        "destination": {  
            "type": "string",  
            "description": "this is the destination city for the flight"  
        },  
        "date": {  
            "type": "string",  
            "description": "The date for the flight in YYYY-MM-DD format"  
        }  
    }  
},
```

## Модели отправки

Модель отправки содержит список языковых моделей и список намерений верхнего уровня, поддерживаемых навыком. Это дополнительная функция, позволяющая разработчику объекта-получателя навыков создать языковую модель, объединяющую признаки объекта-получателя и ботов навыка.

Имя языкового стандарта представляет собой сочетание двухбуквенного кода ISO 639 языка и региональных параметров в нижнем регистре, связанного с языком, и необязательного двухбуквенного кода ISO 3166 подкатегории языка и региональных параметров в верхнем регистре, связанного со страной или регионом, например "ru" или "ru-RU".

ПОЛЕ	ТИП	ОБЯЗАТЕЛЬНО	ОПИСАНИЕ
намерениях,	массив строк	Необязательно	Список намерений верхнего уровня, поддерживаемых навыком. Все намерения должны быть уникальными.

ПОЛЕ	ТИП	ОБЯЗАТЕЛЬНО	ОПИСАНИЕ
языки	Объект, содержащий именованные массивы <a href="#">languageModel</a>	Обязательно	Список языковых моделей, поддерживаемых навыком. Каждое имя — это языковой стандарт, для которого предназначены языковые модели, а массив содержит языковые модули для этого языкового стандарта. Модель отправки должна поддерживать по крайней мере один языковой стандарт. Каждый языковой стандарт в поле languages должен быть уникальным.

Ниже приведен пример модели отправки, которая содержит две языковые модели для трех языковых стандартов. В ней также описаны два намерения высшего уровня, которые могут быть распознаны навыком.

```
"dispatchModels": {
  "languages": {
    "en": [
      {
        "name": "SkillBot LU (English)",
        "contentType": "application/lu",
        "url": "http://sample.com/SkillBot-en.lu",
        "description": "English language model for the skill"
      },
      {
        "name": "SkillBot QnA LU (English)",
        "contentType": "application/qna",
        "url": "http://sample.com/SkillBot-QnA-en.qna",
        "description": "English language model for the skill (QnAMaker)"
      }
    ],
    "es-ES": [
      {
        "name": "SkillBot LU (Spanish-Spain)",
        "contentType": "application/lu",
        "url": "http://sample.com/SkillBot-es-ES.lu",
        "description": "Spanish (Spain) language model for the skill"
      },
      {
        "name": "SkillBot QnA LU (Spanish-Spain)",
        "contentType": "application/qna",
        "url": "http://sample.com/SkillBot-QnA-es-ES.qna",
        "description": "Spanish (Spain) language model for the skill (QnAMaker)"
      }
    ],
    "es-MX": [
      {
        "name": "SkillBot LU (Spanish-Mexico)",
        "contentType": "application/lu",
        "url": "http://sample.com/SkillBot-es-MX.lu",
        "description": "Spanish (Mexico) language model for the skill"
      },
      {
        "name": "SkillBot QnA LU (Spanish-Mexico)",
        "contentType": "application/qna",
        "url": "http://sample.com/SkillBot-QnA-es-MX.qna",
        "description": "Spanish (Mexico) language model for the skill (QnAMaker)"
      }
    ],
    "intents": [
      "bookFlight",
      "getWeather"
    ]
  }
},
```

### Объект [languageModel](#)

Описывает языковую модель для данного языка и региональных параметров. Именем является имя

языкового стандарта.

ПОЛЕ	ТИП	ОБЯЗАТЕЛЬНО	ОПИСАНИЕ
contentType	строка	Обязательно	Тип языковой модели.
description	строка	Необязательно	Описание языковой модели.
name	строка	Обязательно	Имя языковой модели.
url	строка	Обязательно	URL-адрес языковой модели.

## Пример манифеста

Это полный пример манифеста версии 2.1 для навыка, предоставляющего несколько действий.

```
{
    "$schema": "https://schemas.botframework.com/schemas/skills/v2.1/skill-manifest.json",
    "$id": "SkillBot",
    "name": "Sample skill definition that can handle multiple types of activities",
    "version": "1.0",
    "description": "This is a sample skill definition for multiple activity types",
    "publisherName": "Microsoft",
    "privacyUrl": "https://myskill.contoso.com/privacy.html",
    "copyright": "Copyright (c) Microsoft Corporation. All rights reserved.",
    "license": "",
    "iconUrl": "https://myskill.contoso.com/icon.png",
    "tags": [
        "sample",
        "travel",
        "weather"
    ],
    "endpoints": [
        {
            "name": "americas",
            "protocol": "BotFrameworkV3",
            "description": "Production endpoint for SkillBot in the Americas",
            "endpointUrl": "http://myskill.contoso.com/api/messages",
            "msAppId": "00000000-0000-0000-0000-000000000000"
        },
        {
            "name": "eu",
            "protocol": "BotFrameworkV3",
            "description": "Production endpoint for SkillBot in Europe",
            "endpointUrl": "http://myskill.contoso.com/api/messages",
            "msAppId": "11111111-0000-0000-000000000000"
        }
    ],
    "dispatchModels": {
        "languages": {
            "en": [
                {
                    "name": "SkillBot LU (English)",
                    "contentType": "application/lu",
                    "url": "http://sample.com/SkillBot-en.lu",
                    "description": "English language model for the skill"
                },
                {
                    "name": "SkillBot QnA LU (English)",
                    "contentType": "application/qna",
                    "url": "http://sample.com/SkillBot-QnA-en.qna",
                    "description": "English language model for the skill (QnAMaker)"
                }
            ],
            "es-ES": [
                {
                    "name": "SkillBot LU (Spanish-Spain)",
                    "contentType": "application/lu",
                    "url": "http://sample.com/SkillBot-es-ES.lu",
                    "description": "Spanish (Spain) language model for the skill"
                },
                {
                    "name": "SkillBot QnA LU (Spanish-Spain)",
                    "contentType": "application/qna",
                    "url": "http://sample.com/SkillBot-QnA-es-ES.qna",
                    "description": "Spanish (Spain) language model for the skill (QnAMaker)"
                }
            ]
        }
    }
}
```

```

        ],
        "es-MX": [
            {
                "name": "SkillBot LU (Spanish-Mexico)",
                "contentType": "application/lu",
                "url": "http://sample.com/SkillBot-es-MX.lu",
                "description": "Spanish (Mexico) language model for the skill"
            },
            {
                "name": "SkillBot QnA LU (Spanish-Mexico)",
                "contentType": "application/qna",
                "url": "http://sample.com/SkillBot-QnA-es-MX.qna",
                "description": "Spanish (Mexico) language model for the skill (QnAMaker)"
            }
        ],
        "intents": [
            "bookFlight",
            "getWeather"
        ],
        "activities": {
            "bookFlight": {
                "description": "Books a flight",
                "type": "event",
                "name": "Bookflight",
                "value": {
                    "$ref": "#/definitions/bookingInfo"
                },
                "resultValue": {
                    "$ref": "#/definitions/bookingInfo"
                }
            },
            "getWeather": {
                "description": "Retrieves and returns the weather for the user's location",
                "type": "invoke",
                "name": "GetWeather",
                "value": {
                    "$ref": "#/definitions/location"
                },
                "resultValue": {
                    "$ref": "#/definitions/weatherReport"
                }
            },
            "message": {
                "type": "message",
                "description": "Receives the user's utterance and attempts to resolve it using the skill's LU
models"
            },
            "typing": {
                "type": "typing"
            },
            "conversationUpdate": {
                "type": "conversationUpdate"
            }
        },
        "definitions": {
            "localeValue": {
                "type": "object",
                "properties": {
                    "locale": {
                        "type": "string",
                        "description": "The current user's locale ISO code"
                    }
                }
            },
            "bookingInfo": {
                "type": "object",
                "required": [
                    "origin"
                ],
                "properties": {
                    "origin": {
                        "type": "string",
                        "description": "this is the origin city for the flight"
                    },
                    "destination": {
                        "type": "string",
                        "description": "this is the destination city for the flight"
                    },
                    "date": {
                        "type": "string",
                        "description": "The date for the flight in YYYY-MM-DD format"
                    }
                }
            }
        }
    }
}

```

```
        }
    },
    "weatherReport": {
        "type": "array",
        "description": "Array of forecasts for the next week.",
        "items": [
            {
                "type": "string"
            }
        ]
    },
    "location": {
        "type": "object",
        "description": "Location metadata",
        "properties": {
            "latitude": {
                "type": "number",
                "title": "Latitude"
            },
            "longitude": {
                "type": "number",
                "title": "Longitude"
            },
            "postalCode": {
                "type": "string",
                "title": "Postal code"
            }
        }
    },
    "activitiesSent": {
        "flightUpdated": {
            "type": "event",
            "name": "FlightUpdated",
            "description": "Event which is sent by the skill when there is an update in flight info",
            "value": {
                "type": "object",
                "description": "Flight update information",
                "properties": {
                    "flightNumber": {
                        "type": "string"
                    },
                    "departureDate": {
                        "type": "string",
                        "description": "The departure date for the flight in YYYY-MM-DD format"
                    },
                    "departureTime": {
                        "type": "string",
                        "description": "The departure time for the flight in HH-MM format"
                    }
                }
            }
        }
    }
}
```

# Как создать манифест навыка версии 2.0

27.03.2021 • 9 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

*Манифест навыка* — это файл в формате JSON с описанием действий, которые может выполнять навык, его входных и выходных параметров, а также конечных точек навыка. Манифест содержит сведения, которые нужны разработчику для обращения к навыку из другого бота.

В этой статье описывается [версия 2.0.0](#) схемы манифеста навыка Bot Framework. Описание версии 2.1 см. в статье [написание манифеста с версией 2.1](#).

Схема манифеста навыка Bot Framework основана на словаре схемы JSON черновой версии 7.

## Предварительные требования

- Знание принципов работы [навыков и ботов навыков](#).
- Общее понимание [схемы JSON](#) и формата JSON.

## Манифест навыка

Манифест навыка содержит разные категории информации:

- метаданные, описывающие навык на общем уровне;
- список конечных точек, предоставляемых навыком;
- необязательные списки действий, которые навык может получать;
- необязательный объект определений, содержащий схемы для объектов, на которые ссылается другие части документа.

Ниже приведена полная схема для манифеста навыка Bot Framework версии 2.0.

Категория или поле	Тип	Обязательно	Описание
<b>Метаданные</b>			
\$id	строка	Обязательно	Идентификатор манифеста навыка.
\$schema	строка	Обязательно	Универсальный код ресурса (URI) HTTPS ресурса схемы JSON, описывающий формат манифеста. Для версии 2.0.0 URI имеет значение <a href="https://schemas.botframework.com/schemas/skillmanifest.json">https://schemas.botframework.com/schemas/skillmanifest.json</a>
copyright	строка	Необязательно	Уведомление об авторских правах на этот навык.
description	строка	Необязательно	Понятное описание навыка.
iconUrl	строка	Необязательно	Универсальный код ресурса (URI) значка, отображаемого для навыка.
license	строка	Необязательно	Лицензионное соглашение для навыка.
name	строка	Обязательно	Имя навыка.

Категория или поле	Тип	Обязательно	Описание
version	строка	Обязательно	Версия навыка, который описывает манифест.
privacyUrl	строка	Необязательно	Универсальный код ресурса (URI) сведений о конфиденциальности для навыка.
publisherName	строка	Обязательно	Имя издателя навыка.
tags	массив строк	Необязательно	Набор тегов для навыка. Если он есть, каждый тег должен быть уникальным.
<b>Конечные точки</b>			
конечные точки	Массив элементов <a href="#">endpoint</a> .	Обязательно	Список конечных точек, поддерживаемых навыком. Необходимо определить хотя бы одну конечную точку. Имя каждой конечной точки должно быть уникальным.
<b>Действия</b>			
Действия	Объект, содержащий именованные <a href="#">объекты действий</a> .	Обязательно	Набор начальных действий, принимаемых навыком.
<b>Определения</b>			
definitions	объект	Необязательно	Объект, содержащий вложенные схемы для объектов, используемых в манифесте.

## Конечные точки

Каждый объект конечной точки описывает конечную точку, поддерживаемую навыком.

В этом примере приведены две конечные точки для навыка.

```
"endpoints": [
  {
    "name": "americas",
    "protocol": "BotFrameworkV3",
    "description": "Production endpoint for SkillBot in the Americas",
    "endpointUrl": "http://myskill.contoso.com/api/messages",
    "msAppId": "00000000-0000-0000-0000-000000000000"
  },
  {
    "name": "eu",
    "protocol": "BotFrameworkV3",
    "description": "Production endpoint for SkillBot in Europe",
    "endpointUrl": "http://myskill.contoso.com/api/messages",
    "msAppId": "11111111-0000-0000-0000-000000000000"
  }
],
```

### Объект endpoint

Поле	Тип	Обязательно	Описание
description	строка	Необязательно	Описание конечной точки.

ПОЛЕ	ТИП	ОБЯЗАТЕЛЬНО	ОПИСАНИЕ
endpointUrl	строка	Обязательно	Универсальный код ресурса (URI) конечной точки для навыка.
msAppId	строка	Обязательно	Идентификатор Microsoft AppId (GUID) для навыка, используемый при аутентификации запросов.
name	строка	Обязательно	Уникальное имя конечной точки.
protocol	строка	Необязательно	Поддерживаемый протокол. Значение по умолчанию — BotFrameworkV3.

## Действия

Каждый объект действия описывает действие, принимаемое навыком. На основе полученного исходного действия навык запустит действие или задачу. Имя, связанное с объектом действия, указывает на действие или задачу, которая будет выполнена навыком.

Некоторые типы действий имеют свойство value, которое позволяет предоставлять дополнительные входные данные для навыка. Когда навык завершает действие, он может предоставить возвращаемое значение в свойстве value связанного действия end-of-conversation.

В схеме манифеста навыка версии 2.0.0 разрешены следующие типы действий: сообщение, событие и вызов действий.

Ниже приведен пример описания действия.

```
"bookFlight": {
    "description": "Books a flight",
    "type": "event",
    "name": "BookFlight",
    "value": {
        "$ref": "#/definitions/bookingInfo"
    },
    "resultValue": {
        "$ref": "#/definitions/bookingInfo"
    }
},
```

### Объект eventActivity

Описывает действие события, принимаемое навыком.

ПОЛЕ	ТИП	ОБЯЗАТЕЛЬНО	ОПИСАНИЕ
description	строка	Необязательно	Описание действия.
name	строка	Обязательно	Значение свойства name для действия события.
resultValue	объект	Необязательно	Определение схемы JSON для типа объекта, который связанное действие может возвращать.
type	строка	Обязательно	Тип действия. Ожидается значение event.

ПОЛЕ	ТИП	ОБЯЗАТЕЛЬНО	ОПИСАНИЕ
value	объект	Необязательно	Определение схемы JSON для типа объекта, который это действие ожидает в качестве входных данных.

### Объект invokeActivity

Описывает действие вызова, которое навык принимает.

ПОЛЕ	ТИП	ОБЯЗАТЕЛЬНО	ОПИСАНИЕ
description	строка	Необязательно	Описание действия.
name	строка	Обязательно	Значение свойства name для действия вызова.
resultValue	объект	Необязательно	Определение схемы JSON для типа объекта, который связанное действие может возвращать.
type	строка	Обязательно	Тип действия. Ожидается значение invoke.
value	объект	Необязательно	Определение схемы JSON для типа объекта, который это действие ожидает в качестве входных данных.

### Объект messageActivity

Описывает действие сообщения, принимаемое навыком. Свойство text для действия сообщения содержит речевой фрагмент, полученный от пользователя.

ПОЛЕ	ТИП	ОБЯЗАТЕЛЬНО	ОПИСАНИЕ
description	строка	Необязательно	Описание действия.
resultValue	объект	Необязательно	Определение схемы JSON для типа объекта, который связанное действие может возвращать.
type	строка	Обязательно	Тип действия. Ожидается значение message.
value	объект	Необязательно	Определение схемы JSON для типа объекта, который это действие ожидает в качестве входных данных.

## Определения

Каждое определение описывает вложенную схему, которую можно использовать в других частях документа.

Ниже приведен пример вложенной схемы для сведений о бронировании авиабилетов.

```

"bookingInfo": {
    "type": "object",
    "required": [
        "origin"
    ],
    "properties": {
        "origin": {
            "type": "string",
            "description": "this is the origin city for the flight"
        },
        "destination": {
            "type": "string",
            "description": "this is the destination city for the flight"
        },
        "date": {
            "type": "string",
            "description": "The date for the flight in YYYY-MM-DD format"
        }
    }
},

```

## Пример манифеста

Это полный пример манифеста версии 2.0 для навыка, предоставляющего несколько действий.

```

{
    "$schema": "https://schemas.botframework.com/schemas/skills/v2.0/skill-manifest.json",
    "$id": "SkillBot",
    "name": "Sample skill definition that can handle multiple types of activities",
    "version": "1.0",
    "description": "This is a sample skill definition for multiple activity types",
    "publisherName": "Microsoft",
    "privacyUrl": "https://myskill.contoso.com/privacy.html",
    "copyright": "Copyright (c) Microsoft Corporation. All rights reserved.",
    "license": "",
    "iconUrl": "https://myskill.contoso.com/icon.png",
    "tags": [
        "sample",
        "travel",
        "weather"
    ],
    "endpoints": [
        {
            "name": "americas",
            "protocol": "BotFrameworkV3",
            "description": "Production endpoint for SkillBot in the Americas",
            "endpointUrl": "http://myskill.contoso.com/api/messages",
            "msAppId": "00000000-0000-0000-0000-000000000000"
        },
        {
            "name": "eu",
            "protocol": "BotFrameworkV3",
            "description": "Production endpoint for SkillBot in Europe",
            "endpointUrl": "http://myskill.contoso.com/api/messages",
            "msAppId": "11111111-0000-0000-0000-000000000000"
        }
    ],
    "activities": {
        "bookFlight": {
            "description": "Books a flight",
            "type": "event",
            "name": "BookFlight",
            "value": {
                "$ref": "#/definitions/bookingInfo"
            },
            "resultValue": {
                "$ref": "#/definitions/bookingInfo"
            }
        },
        "getWeather": {
            "description": "Retrieves and returns the weather for the user's location",
            "type": "invoke",
            "name": "GetWeather",
            "value": {
                "$ref": "#/definitions/location"
            },
            "resultValue": {
                "$ref": "#/definitions/weatherReport"
            }
        }
    }
},

```

```

    "message": {
        "type": "message",
        "description": "Receives the user's utterance and attempts to resolve it using the skill's LU
models"
    },
    "definitions": {
        "localeValue": {
            "type": "object",
            "properties": {
                "locale": {
                    "type": "string",
                    "description": "The current user's locale ISO code"
                }
            }
        },
        "bookingInfo": {
            "type": "object",
            "required": [
                "origin"
            ],
            "properties": {
                "origin": {
                    "type": "string",
                    "description": "this is the origin city for the flight"
                },
                "destination": {
                    "type": "string",
                    "description": "this is the destination city for the flight"
                },
                "date": {
                    "type": "string",
                    "description": "The date for the flight in YYYY-MM-DD format"
                }
            }
        },
        "weatherReport": {
            "type": "array",
            "description": "Array of forecasts for the next week.",
            "items": [
                {
                    "type": "string"
                }
            ]
        },
        "location": {
            "type": "object",
            "description": "Location metadata",
            "properties": {
                "latitude": {
                    "type": "number",
                    "title": "Latitude"
                },
                "longitude": {
                    "type": "number",
                    "title": "Longitude"
                },
                "postalCode": {
                    "type": "string",
                    "title": "Postal code"
                }
            }
        }
    }
}

```

# Использование диалогов в навыке

27.03.2021 • 23 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье показано, как создать навык, который поддерживает несколько действий. Для этого в нем используются диалоги. Основной диалог получает исходные входные данные от потребителя навыка, а затем запускает соответствующее действие. Сведения о реализации потребителя навыков и соответствующий пример кода см. в статье об [использовании навыков с помощью диалогов](#).

В этой статье предполагается, что вы уже умеете создавать навыки. Сведения о создании бота навыка см. в статье о [реализации навыка](#).

## Предварительные требования

- Понимание основ работы с ботами, [принципа работы ботов с навыками](#) и [принципа реализации навыка](#).
- Подписка Azure (для развертывания навыков). Если у вас еще нет подписки Azure, создайте [бесплатную учетную запись](#), прежде чем начать работу.
- Учетная запись [LUIS](#) (необязательно). (См. сведения о [добавлении функции распознавания естественного языка в бота](#).)
- Копия примера `skills skillDialog` для [C#](#), [JavaScript](#) или [Python](#).

### NOTE

Начиная с версии 4.11 для проверки навыка локально в эмуляторе не требуется идентификатор приложения и пароль. Подписка Azure по-прежнему необходима для развертывания вашего навыка в Azure.

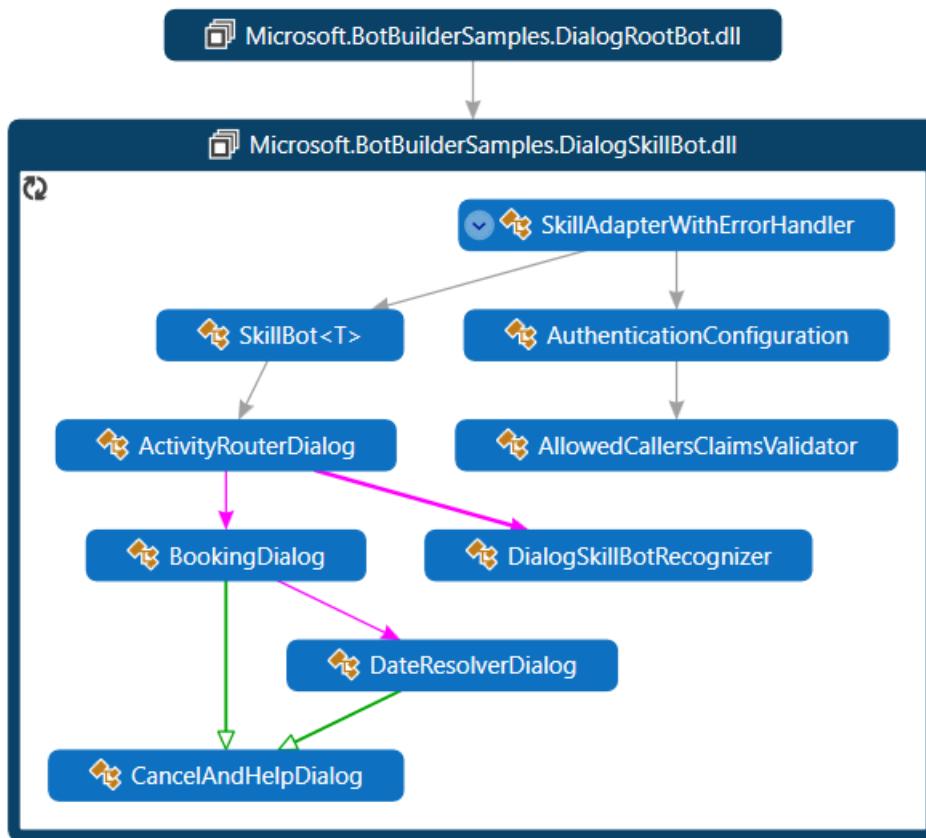
## Об этом примере

В пример `skills skillDialog` включены проекты двух ботов:

- *Корневой бот диалога*, который использует *диалог навыка* для применения навыка.
- *Бот навыка диалога*, который использует диалог для обработки действий, поступающих от потребителей навыков. Этот навык является адаптацией примера `core bot`. (Сведения о примере `core bot` см. в статье о [добавлении функции распознавания естественного языка в бота](#).)

В этой статье основное внимание уделяется использованию диалогов в боте навыка для управления несколькими действиями.

- [C#](#)
- [JavaScript](#)
- [Python](#)



Сведения о боте, выполняющем роль потребителя навыка, см. в статье [об использовании навыка с помощью диалогов](#).

## Ресурсы

Для развернутого программы-роботы проверка подлинности «Bot-to-Bot» требует, чтобы каждый участвующий робот получил допустимые идентификатор приложения и пароль. Тем не менее вы можете проверить навыки и квалификацию потребителей локально с помощью эмулятора без идентификатора приложения и пароля.

Чтобы сделать навык доступным для программы-роботы пользователей, зарегистрируйте навык в Azure. Для этого можно применить службу "Регистрация каналов бота". Дополнительные сведения см. в статье о [регистрации бота в службе Azure Bot](#).

Кроме того, бот навыка может использовать модель LUIS для бронирования авиабилетов. Чтобы применить эту модель, с помощью файла CognitiveModels/FlightBooking.json создайте, обучите и опубликуйте модель LUIS.

## Конфигурация приложений

- При необходимости добавьте идентификатор и пароль приложения навыка в файл конфигурации навыка. (Если потребитель или навык используют идентификатор приложения и пароль, оба должны быть.)
  - C#
  - JavaScript
  - Python
- Если вы используете модель LUIS, добавьте идентификатор приложения LUIS, ключ и имя узла API.

DialogSkillBot\appsettings.json

```
{
  "MicrosoftAppId": "",
  "MicrosoftAppPassword": "",
  "ConnectionName": "",

  "LuisAppId": "",
  "LuisAPIKey": "",
  "LuisAPIHostName": "",

  // This is a comma separate list with the App IDs that will have access to the skill.
  // This setting is used in AllowedCallersClaimsValidator.
  // Examples:
  //   [ "*" ] allows all callers.
  //   [ "AppId1", "AppId2" ] only allows access to parent bots with "AppId1" and "AppId2".
  "AllowedCallers": [ "*" ]
}
```

## Логика activity-routing

Этот навык поддерживает несколько разных функций. Он может забронировать авиабилет и получить прогноз погоды для города. Кроме того, если он получает сообщение за пределами одного из этих контекстов, он может использовать LUIS, чтобы попытаться интерпретировать сообщение. Манифест навыка описывает эти действия, входные и выходные параметры, а также конечные точки навыка. Обратите внимание, что навык умеет обрабатывать события BookFlight и GetWeather. Он также обрабатывает действия сообщений.

Навык определяет диалог activity-routing, который используется для выбора запускаемого действия в зависимости от исходного действия, полученного от потребителя навыков. Если предоставлена модель LUIS, она может распознавать в исходном сообщении намерения бронирования авиабилетов и получения прогноза погоды.

Действие book-flight — это многоэтапный процесс, который реализован в отдельном диалоге. Начавшись, это действие обрабатывает все входящие действия. Действие get-weather содержит логику заполнителя, которую для полноценной реализации бота нужно будет заменить.

Диалог activity-routing содержит код для выполнения следующих действий:

- [инициализация диалога](#);
- [обработка начального действия](#);
- [обработка действий сообщения](#);
- [запуск многошагового действия](#);
- [возврат результата](#).

Используемые в этом навыке диалоги унаследованы от класса *компонентного диалога*. Дополнительные сведения о компонентных диалогах см. в статье об [управлении сложностью диалогов](#).

### Инициализация диалога

Диалог activity-routing включает дочерний диалог для бронирования авиабилетов. В главном каскадном диалоге есть один шаг, который запускает действие на основе полученного исходного действия.

Он также может принимать распознаватель LUIS. Если этот распознаватель инициализирован, диалог попытается применить его для оценки намерения в исходном действии сообщения.

- [C#](#)
- [JavaScript](#)
- [Python](#)

## DialogSkillBot\Dialogs\ActivityRouterDialog.cs

```
private readonly DialogSkillBotRecognizer _luisRecognizer;

public ActivityRouterDialog(DialogSkillBotRecognizer luisRecognizer)
    : base(nameof(ActivityRouterDialog))
{
    _luisRecognizer = luisRecognizer;

    AddDialog(new BookingDialog());
    AddDialog(new WaterfallDialog(nameof(WaterfallDialog), new WaterfallStep[] { ProcessActivityAsync }));

    // The initial child Dialog to run.
    InitialDialogId = nameof(WaterfallDialog);
}
```

### Обработка начального действия

На первом (и только первом) шаге в главном каскадном диалоге навык проверяет тип входящего действия.

- Действия событий передаются *обработчику действий события*, который запускает соответствующее действие на основе имени события.
- Действия с сообщениями передаются *обработчику действий сообщений*, который выполняет дополнительную обработку, прежде чем решать, что делать дальше.

Если навык не может распознать тип входящего действия или имя события, он выдает сообщение об ошибке и завершает работу.

- [C#](#)
- [JavaScript](#)
- [Python](#)

## DialogSkillBot\Dialogs\ActivityRouterDialog.cs

```
private async Task<DialogTurnResult> ProcessActivityAsync(WaterfallStepContext stepContext,
CancellationToken cancellationToken)
{
    // A skill can send trace activities, if needed.
    await stepContext.Context.TraceActivityAsync($"{GetType().Name}.ProcessActivityAsync()", label: $"Got
ActivityType: {stepContext.Context.Activity.Type}", cancellationToken: cancellationToken);

    switch (stepContext.Context.Activity.Type)
    {
        case ActivityTypes.Event:
            return await OnEventActivityAsync(stepContext, cancellationToken);

        case ActivityTypes.Message:
            return await OnMessageActivityAsync(stepContext, cancellationToken);

        default:
            // We didn't get an activity type we can handle.
            await stepContext.Context.SendActivityAsync(MessageFactory.Text($"Unrecognized ActivityType: \"
{stepContext.Context.Activity.Type}\".", inputHint: InputHints.IgnoringInput), cancellationToken);
            return new DialogTurnResult(DialogTurnStatus.Complete);
    }
}
```

```
// This method performs different tasks based on the event name.
private async Task<DialogTurnResult> OnEventActivityAsync(WaterfallStepContext stepContext,
CancellationToken cancellationToken)
{
    var activity = stepContext.Context.Activity;
    await stepContext.Context.TraceActivityAsync($"{GetType().Name}.OnEventActivityAsync()", label: $"Name: {activity.Name}. Value: {GetObjectAsJsonString(activity.Value)}", cancellationToken: cancellationToken);

    // Resolve what to execute based on the event name.
    switch (activity.Name)
    {
        case "BookFlight":
            return await BeginBookFlight(stepContext, cancellationToken);

        case "GetWeather":
            return await BeginGetWeather(stepContext, cancellationToken);

        default:
            // We didn't get an event name we can handle.
            await stepContext.Context.SendActivityAsync(MessageFactory.Text($"Unrecognized EventName: \"{activity.Name}\".", inputHint: InputHints.IgnoringInput), cancellationToken);
            return new DialogTurnResult(DialogTurnStatus.Complete);
    }
}
```

## Обработка действий сообщения

Если настроен распознаватель LUIS, навык вызывает LUIS и запускает действие на основе намерения. Если распознаватель LUIS не настроен или намерение не поддерживается, навык выдает сообщение об ошибке и завершает работу.

- [C#](#)
- [JavaScript](#)
- [Python](#)

[DialogSkillBot\Dialogs\ActivityRouterDialog.cs](#)

```

// This method just gets a message activity and runs it through LUIS.
private async Task<DialogTurnResult> OnMessageActivityAsync(WaterfallStepContext stepContext,
CancellationToken cancellationToken)
{
    var activity = stepContext.Context.Activity;
    await stepContext.Context.TraceActivityAsync($"{GetType().Name}.OnMessageActivityAsync()", label:
$"Text: \'{activity.Text}\'. Value: {GetObjectAsJsonString(activity.Value)}", cancellationToken:
cancellationToken);

    if (!luisRecognizer.IsConfigured)
    {
        await stepContext.Context.SendActivityAsync(MessageFactory.Text("NOTE: LUIS is not configured. To
enable all capabilities, add 'LuisAppId', 'LuisAPIKey' and 'LuisAPIHostName' to the appsettings.json file."),
inputHint: InputHints.IgnoringInput), cancellationToken);
    }
    else
    {
        // Call LUIS with the utterance.
        var luisResult = await luisRecognizer.RecognizeAsync<FlightBooking>(stepContext.Context,
cancellationToken);

        // Create a message showing the LUIS results.
        var sb = new StringBuilder();
        sb.AppendLine($"LUIS results for \'{activity.Text}\':");
        var (intent, intentScore) = luisResult.Intents.FirstOrDefault(x =>
x.Value.Equals(luisResult.Intents.Values.Max()));
        sb.AppendLine($"Intent: \'{intent}\' Score: {intentScore.Score}");

        await stepContext.Context.SendActivityAsync(MessageFactory.Text(sb.ToString()), inputHint:
InputHints.IgnoringInput), cancellationToken);

        // Start a dialog if we recognize the intent.
        switch (luisResult.TopIntent().intent)
        {
            case FlightBooking.Intent.BookFlight:
                return await BeginBookFlight(stepContext, cancellationToken);

            case FlightBooking.Intent.GetWeather:
                return await BeginGetWeather(stepContext, cancellationToken);

            default:
                // Catch all for unhandled intents.
                var didntUnderstandMessageText = $"Sorry, I didn't get that. Please try asking in a
different way (intent was {luisResult.TopIntent().intent})";
                var didntUnderstandMessage = MessageFactory.Text(didntUnderstandMessageText,
didntUnderstandMessageText, InputHints.IgnoringInput);
                await stepContext.Context.SendActivityAsync(didntUnderstandMessage, cancellationToken);
                break;
        }
    }

    return new DialogTurnResult(DialogTurnStatus.Complete);
}

```

## Запуск многошагового действия

Действие бронирования авиабилетов book-flight запускает многошаговый диалог для получения сведений о бронировании от пользователя.

Действие получения прогноза погоды get-weather пока не реализовано. Сейчас оно возвращает заполнитель для сообщения и завершает работу.

- [C#](#)
- [JavaScript](#)
- [Python](#)

## DialogSkillBot\Dialogs\ActivityRouterDialog.cs

```
private async Task<DialogTurnResult> BeginBookFlight(WaterfallStepContext stepContext, CancellationToken cancellationToken)
{
    var activity = stepContext.Context.Activity;
    var bookingDetails = new BookingDetails();
    if (activity.Value != null)
    {
        bookingDetails = JsonConvert.DeserializeObject<BookingDetails>(JsonConvert.SerializeObject(activity.Value));
    }

    // Start the booking dialog.
    var bookingDialog = FindDialog(nameof(BookingDialog));
    return await stepContext.BeginDialogAsync(bookingDialog.Id, bookingDetails, cancellationToken);
}
```

```
private static async Task<DialogTurnResult> BeginGetWeather(WaterfallStepContext stepContext,
CancellationToken cancellationToken)
{
    var activity = stepContext.Context.Activity;
    var location = new Location();
    if (activity.Value != null)
    {
        location = JsonConvert.DeserializeObject<Location>(JsonConvert.SerializeObject(activity.Value));
    }

    // We haven't implemented the GetWeatherDialog so we just display a TODO message.
    var getWeatherMessageText = $"TODO: get weather for here (lat: {location.Latitude}, long: {location.Longitude})";
    var getWeatherMessage = MessageFactory.Text(getWeatherMessageText, getWeatherMessageText,
InputHints.IgnoringInput);
    await stepContext.Context.SendActivityAsync(getWeatherMessage, cancellationToken);
    return new DialogTurnResult(DialogTurnStatus.Complete);
}
```

### Возврат результата

Навык запускает диалог бронирования для действия book-flight. Так как диалог activity-routing содержит только один шаг, при завершении диалога бронирования завершится и диалог activity-routing, а результат выполнения диалога бронирования автоматически становится результатом диалога activity-routing.

Действие get-weather просто завершается без возвращаемого значения.

### Отмена многошагового действия

Диалог бронирования и его дочерний диалог date-resolver являются производными от базового диалога cancel-and-help, который проверяет получение сообщений от пользователя.

- При вводе help или ? он отображает справочную информацию и возвращается на следующем шаге в поток беседы.
- При вводе cancel или quit он завершает все диалоги, завершая работу навыка.

Дополнительные сведения см. в статье об [обработке прерываний пользователя](#).

### Регистрация службы

Для этого навыка нужны те же службы, что и для любого стандартного бота навыка. Требуемые службы

обсуждаются в статье о реализации навыка.

## Манифест навыка

*Манифест навыка* — это файл в формате JSON с описанием действий, которые может выполнять навык, его входных и выходных параметров, а также конечных точек навыка. Манифест содержит сведения, которые вам нужны для доступа к навыку из другого бота.

- [C#](#)
- [JavaScript](#)
- [Python](#)

`DialogSkillBot\wwwroot\manifest\dialogchildbot-manifest-1.0.json`

```
{
  "$schema": "https://schemas.botframework.com/schemas/skills/skill-manifest-2.0.0.json",
  "$id": "DialogSkillBot",
  "name": "Skill bot with dialogs",
  "version": "1.0",
  "description": "This is a sample skill definition for multiple activity types.",
  "publisherName": "Microsoft",
  "privacyUrl": "https://dialogskillbot.contoso.com/privacy.html",
  "copyright": "Copyright (c) Microsoft Corporation. All rights reserved.",
  "license": "",
  "iconUrl": "https://dialogskillbot.contoso.com/icon.png",
  "tags": [
    "sample",
    "travel",
    "weather",
    "luis"
  ],
  "endpoints": [
    {
      "name": "default",
      "protocol": "BotFrameworkV3",
      "description": "Default endpoint for the skill.",
      "endpointUrl": "https://dialogskillbot.contoso.com/api/messages",
      "msAppId": "00000000-0000-0000-0000-000000000000"
    }
  ],
  "activities": {
    "bookFlight": {
      "description": "Books a flight (multi turn).",
      "type": "event",
      "name": "BookFlight",
      "value": {
        "$ref": "#/definitions/bookingInfo"
      },
      "resultValue": {
        "$ref": "#/definitions/bookingInfo"
      }
    },
    "getWeather": {
      "description": "Retrieves and returns the weather for the user's location.",
      "type": "event",
      "name": "GetWeather",
      "value": {
        "$ref": "#/definitions/location"
      },
      "resultValue": {
        "$ref": "#/definitions/weatherReport"
      }
    },
    "passthroughMessage": {
      "type": "message",
      "value": {
        "$ref": "#/definitions/passthrough"
      }
    }
  }
}
```

```

    "description": "Receives the user's utterance and attempts to resolve it using the skill's LUIS
models.",
    "value": {
        "type": "object"
    }
},
"definitions": {
    "bookingInfo": {
        "type": "object",
        "required": [
            "origin"
        ],
        "properties": {
            "origin": {
                "type": "string",
                "description": "This is the origin city for the flight."
            },
            "destination": {
                "type": "string",
                "description": "This is the destination city for the flight."
            },
            "travelDate": {
                "type": "string",
                "description": "The date for the flight in YYYY-MM-DD format."
            }
        }
    },
    "weatherReport": {
        "type": "array",
        "description": "Array of forecasts for the next week.",
        "items": [
            {
                "type": "string"
            }
        ]
    },
    "location": {
        "type": "object",
        "description": "Location metadata.",
        "properties": {
            "latitude": {
                "type": "number",
                "title": "Latitude"
            },
            "longitude": {
                "type": "number",
                "title": "Longitude"
            },
            "postalCode": {
                "type": "string",
                "title": "Postal code"
            }
        }
    }
}
}

```

*Схема манифеста навыка* представляет собой файл в формате JSON с описанием схемы манифеста навыков. Последняя версия схемы — [v 2.1](#).

## Тестирование бота навыка

Вы можете проверить навык в эмуляторе с помощью потребителя навыков. Для этого одновременно запустите оба бота (навыка и потребителя навыка). Сведения о настройке навыка см. в статье об

использовании навыков с помощью диалога.

Скачайте и установите последнюю версию [Bot Framework Emulator](#).

1. Запустите бот навыка диалога и корневой бот диалога локально на компьютере. Дополнительные инструкции для [C#](#), [JavaScript](#) и [Python](#) см. в файле сведений соответствующего примера.
2. Примените эмулятор для тестирования бота.
  - Когда вы впервые присоединяетесь к беседе, бот отображает приветственное сообщение и спрашивает, какой навык вы хотите вызвать. В нашем примере бота навыков есть всего один навык.
  - Выберите **DialogSkillBot**.
3. Теперь бот предложит вам выбрать действие для этого навыка. Выберите **BookFlight**.
  - a. Навык начинает работу с действия book-flight, а вам нужно ответить на вопросы.
  - b. Когда навык завершит работу, корневой бот отобразит сведения о резервировании и снова предложит вам вызвать навык.
4. Снова выберите **DialogSkillBot** и **BookFlight**.
  - a. Ответьте на первый вопрос, а затем введите **cancel**, чтобы отменить действие.
  - b. Бот навыка завершит работу, не завершив действие, а потребитель снова предложит вам вызвать навык.

## Дополнительные сведения

- См. сведения в статье [Использование навыка с помощью диалога](#).

# Реализация потребителя навыка

27.03.2021 • 32 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Вы можете использовать навыки для расширения функциональности другого бота. *Навыком* здесь называется бот, который может выполнять ряд задач для другого бота и использует манифест для описания интерфейса. *Корневым ботом* называется бот, который взаимодействует с пользователем и может вызывать один или несколько навыков. Корневой бот — типичный вариант *потребителя навыка*.

- Потребитель навыков должен использовать проверку утверждений для управления тем, к каким навыкам имеет доступ.
- Потребитель навыка может использовать несколько навыков.
- Разработчики, у которых нет доступа к исходному коду навыка, могут использовать информацию из манифеста этого навыка для разработки потребителей.

В этой статье демонстрируется создание потребителя навыка, который использует эхо-навык для вывода на экран вводимых пользователем данных. Пример манифеста навыка и сведения о реализации эхо-навыка см. в статье [о реализации навыка](#).

Сведения об использовании диалога навыка для получения навыка см. в статье [Использование диалога для получения навыка](#).

## Предварительные требования

- Понимание [основ работы с ботами, принципа работы ботов с навыками и принципа реализации навыка](#).
- При необходимости — подписка Azure. Если у вас еще нет подписки Azure, создайте [бесплатную учетную запись](#), прежде чем начать работу.
- Копия примера [простого навыка для ботов](#) для языка [C#](#), [JavaScript](#) или [Python](#).

### NOTE

Начиная с версии 4.11 вам не требуется идентификатор приложения и пароль для проверки уровня потребителя на локальном компьютере в эмуляторе. Подписка Azure по-прежнему необходима для развертывания потребителя в Azure или для использования развернутого навыка.

## Об этом примере

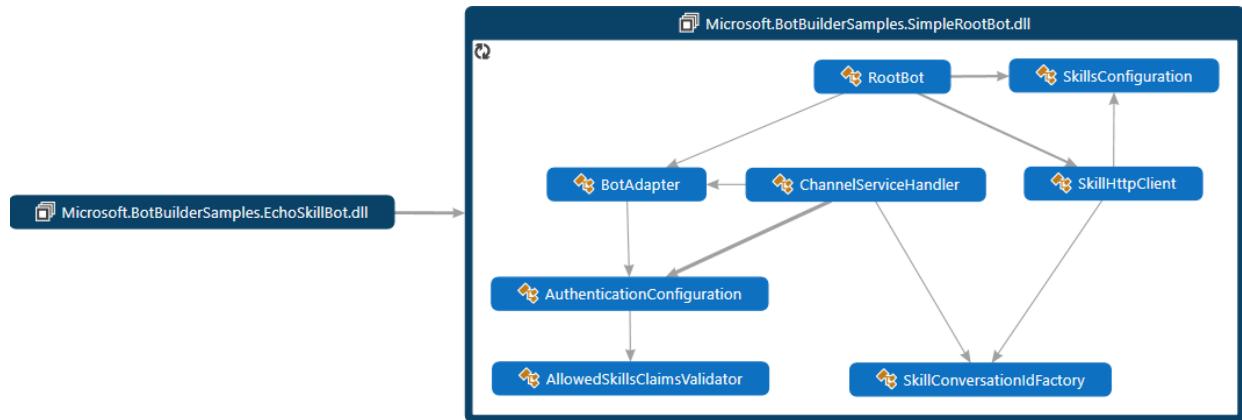
В пример [простого навыка для ботов](#) включены проекты двух ботов:

- *бот эхо-навыка*, который реализует этот навык;
- *простой корневой бот*, который реализует бот для использования этого навыка.

Эта статья посвящена корневому боту, в том числе логике поддержки в объектах бота и адаптера, а также объектам, которые используются для обмена действиями с помощью навыков. К ним относятся следующие объекты.

- Клиент навыка, который используется для отправки действий в навык.
- Обработчик навыков, который используется для получения действий от навыка.

- Фабрика идентификаторов бесед с навыками, которую клиент и обработчик навыков используют для взаимного преобразования ссылок между беседами пользователя с корневым ботом и корневого бота с навыком.
- [C#](#)
- [JavaScript](#)
- [Python](#)



Сведения о боте с эхо-навыком см. в статье [о реализации навыка](#).

## Ресурсы

Для развернутого программы-роботы проверка подлинности «Bot-to-Bot» требует, чтобы каждый участвующий робот получил допустимые идентификатор приложения и пароль. Тем не менее вы можете проверить навыки и квалификацию потребителей локально с помощью эмулятора без идентификатора приложения и пароля.

## Конфигурация приложений

- При необходимости добавьте идентификатор приложения и пароль корневого элемента Bot в файл конфигурации.
- Добавьте конечную точку узла навыка (служба или URL-адрес обратного вызова), на которую навыки должны отвечать на потребителей навыков.
- Добавьте запись для каждого навыка, который будет использоваться потребителем навыка. Каждая запись содержит следующие сведения:
  - идентификатор, который потребитель навыка использует для идентификации навыка;
  - При необходимости — идентификатор приложения навыка.
  - конечная точка обмена сообщениями для навыка.

### NOTE

Если потребитель или навык используют идентификатор приложения и пароль, оба они должны иметь значение.

- [C#](#)
- [JavaScript](#)
- [Python](#)

### SimpleRootBot\appsettings.json

При необходимости добавьте идентификатор и пароль корневого приложения Bot и добавьте

идентификатор приложения для программы-робота по навыку Echo в `BotFrameworkSkills` массив.

```
{
    "MicrosoftAppId": "",
    "MicrosoftAppPassword": "",
    "SkillHostEndpoint": "http://localhost:3978/api/skills/",
    "BotFrameworkSkills": [
        {
            "Id": "EchoSkillBot",
            "AppId": "TODO: Add here the App ID for the skill",
            "SkillEndpoint": "http://localhost:39783/api/messages"
        }
    ]
}
```

## Настройка навыков

Наш пример бота считывает сведения о каждом навыке из файла конфигурации в коллекцию объектов `skill`.

- [C#](#)
- [JavaScript](#)
- [Python](#)

### SimpleRootBot\SkillsConfiguration.cs

```
public class SkillsConfiguration
{
    public SkillsConfiguration(IConfiguration configuration)
    {
        var section = configuration?.GetSection("BotFrameworkSkills");
        var skills = section?.Get<BotFrameworkSkill[]>();
        if (skills != null)
        {
            foreach (var skill in skills)
            {
                Skills.Add(skill.Id, skill);
            }
        }

        var skillHostEndpoint = configuration?.GetValue<string>(nameof(SkillHostEndpoint));
        if (!string.IsNullOrWhiteSpace(skillHostEndpoint))
        {
            SkillHostEndpoint = new Uri(skillHostEndpoint);
        }
    }

    public Uri SkillHostEndpoint { get; }

    public Dictionary<string, BotFrameworkSkill> Skills { get; } = new Dictionary<string, BotFrameworkSkill>();
}
```

## Фабрика идентификаторов беседы

Этот элемент создает идентификатор беседы, который используется для работы с навыком, и может восстановить исходный идентификатор беседы с пользователем по идентификатору беседы с навыком.

Фабрика идентификаторов бесед для этого примера поддерживает простой сценарий со следующими характеристиками:

- корневой бот предназначен для использования одного конкретного навыка;
  - корневой бот поддерживает только одну активную беседу с навыком в конкретный момент времени.
- [C#](#)
  - [JavaScript](#)
  - [Python](#)

### SimpleRootBot\SkillConversationIdFactory.cs

```
public class SkillConversationIdFactory : SkillConversationIdFactoryBase
{
    private readonly ConcurrentDictionary<string, string> _conversationRefs = new
    ConcurrentDictionary<string, string>();

    public override Task<string> CreateSkillConversationIdAsync(SkillConversationIdFactoryOptions options,
    CancellationToken cancellationToken)
    {
        var skillConversationReference = new SkillConversationReference
        {
            ConversationReference = options.Activity.GetConversationReference(),
            OAuthScope = options.FromBotOAuthScope
        };
        var key = $"{options.FromBotId}-{options.BotFrameworkSkill.AppId}-
{skillConversationReference.ConversationReference.Conversation.Id}-
{skillConversationReference.ConversationReference.ChannelId}-skillconvo";
        _conversationRefs.GetOrAdd(key, JsonConvert.SerializeObject(skillConversationReference));
        return Task.FromResult(key);
    }

    public override Task<SkillConversationReference> GetSkillConversationReferenceAsync(string
    skillConversationId, CancellationToken cancellationToken)
    {
        var conversationReference = JsonConvert.DeserializeObject<SkillConversationReference>
        (_conversationRefs[skillConversationId]);
        return Task.FromResult(conversationReference);
    }

    public override Task DeleteConversationReferenceAsync(string skillConversationId, CancellationToken
    cancellationToken)
    {
        _conversationRefs.TryRemove(skillConversationId, out _);
        return Task.CompletedTask;
    }
}
```

В более сложных сценариях фабрика идентификаторов бесед должна поддерживать следующие действия:

- метод *создания идентификатора беседы с навыком* получает или создает идентификатор беседы с навыком;
- метод *получения ссылки на беседу* получает доступ к правильной беседе с пользователем.

## Клиент навыка и обработчик навыка

Потребитель навыка использует клиент навыка, чтобы пересыпать действия в навык. Для этого клиент навыка использует сведения о конфигурации навыков и фабрику идентификаторов бесед.

Потребитель навыка использует обработчик навыка для получения действий от навыка. Обработчик использует для этого фабрику идентификаторов бесед, конфигурацию проверки подлинности и поставщик учетных данных, а также имеет зависимости от адаптера корневого бота и обработчика действий.

- [C#](#)
- [JavaScript](#)
- [Python](#)

### SimpleRootBot\Startup.cs

```
services.AddHttpClient<SkillHttpClient>();
services.AddSingleton<ChannelServiceHandler, SkillHandler>();
```

Трафик HTTP от навыка поступает в конечную точку по URL-адресу службы, которую потребитель навыка сообщает этому навыку. Для передачи трафика обработчику навыка примените обработчик конечной точки для выбранного языка.

Обработчик навыка по умолчанию выполняет следующее:

- Если идентификатор приложения и пароль существуют, использует объект конфигурации проверки подлинности для выполнения проверки подлинности с помощью программы-робота и проверки утверждения.
- использует фабрику идентификаторов бесед для преобразования ссылки на беседу потребителя с навыком в ссылку на беседу пользователя с корневым ботом;
- создает упреждающее сообщение, которое позволяет потребителю навыка восстановить контекст реплики в беседе пользователя с корневым ботом и пересыпать действия пользователю.

## Логика обработчика действий

Обратите внимание, что логика потребителя навыка должна обеспечить следующее:

- запоминать наличие активных навыков и правильно передавать в них действия;
- обнаруживать запросы от пользователя, которые нужно передать в навык, и запускать соответствующий навык;
- обнаруживать действие `endofConversation`, поступающее от любого активного навыка, чтобы зафиксировать его завершение;
- если это уместно, по запросу пользователя или потребителя навыка останавливать навык, который еще не завершил работу;
- сохранять состояние перед вызовом навыка, так как любой ответ может поступить к другому экземпляру потребителя навыка (в том числе балансировку нагрузки и т. п.)

- [C#](#)
- [JavaScript](#)
- [Python](#)

### SimpleRootBot\Bots\RootBot.cs

Корневой бот имеет зависимости от состояния беседы, сведений о навыках, клиента навыка и общей конфигурации. В ASP.NET эти объекты реализуются путем внедрения зависимостей. Также корневой бот определяет метод доступа к свойству состояния беседы, чтобы отслеживать активные навыки.

```

public static readonly string ActiveSkillPropertyName = $"{typeof(RootBot).FullName}.ActiveSkillProperty";
private readonly IStatePropertyAccessor<BotFrameworkSkill> _activeSkillProperty;
private readonly string _botId;
private readonly ConversationState _conversationState;
private readonly SkillHttpClient _skillClient;
private readonly SkillsConfiguration _skillsConfig;
private readonly BotFrameworkSkill _targetSkill;

public RootBot(ConversationState conversationState, SkillsConfiguration skillsConfig, SkillHttpClient
skillClient, IConfiguration configuration)
{
    _conversationState = conversationState ?? throw new ArgumentNullException(nameof(conversationState));
    _skillsConfig = skillsConfig ?? throw new ArgumentNullException(nameof(skillsConfig));
    _skillClient = skillClient ?? throw new ArgumentNullException(nameof(skillClient));
    if (configuration == null)
    {
        throw new ArgumentNullException(nameof(configuration));
    }

    _botId = configuration.GetSection(MicrosoftAppCredentials.MicrosoftAppIdKey)?.Value;
    if (string.IsNullOrWhiteSpace(_botId))
    {
        throw new ArgumentException($"{MicrosoftAppCredentials.MicrosoftAppIdKey} is not set in
configuration");
    }

    // We use a single skill in this example.
    var targetSkillId = "EchoSkillBot";
    if (!_skillsConfig.Skills.TryGetValue(targetSkillId, out _targetSkill))
    {
        throw new ArgumentException($"Skill with ID \'{targetSkillId}\' not found in configuration");
    }

    // Create state property to track the active skill
    _activeSkillProperty = conversationState.CreateProperty<BotFrameworkSkill>(ActiveSkillPropertyName);
}

```

В нашем примере есть вспомогательный метод для перенаправления действий в навык. Он сохраняет состояние беседы перед вызовом навыка и проверяет, успешно ли выполнен HTTP-запрос.

```

private async Task SendToSkill(ITurnContext turnContext, BotFrameworkSkill targetSkill, CancellationToken
cancellationToken)
{
    // NOTE: Always SaveChanges() before calling a skill so that any activity generated by the skill
    // will have access to current accurate state.
    await _conversationState.SaveChangesAsync(turnContext, force: true, cancellationToken:
cancellationToken);

    // route the activity to the skill
    var response = await _skillClient.PostActivityAsync(_botId, targetSkill,
.skillsConfig.SkillHostEndpoint, turnContext.Activity, cancellationToken);

    // Check response status
    if (!(response.Status >= 200 && response.Status <= 299))
    {
        throw new HttpRequestException($"Error invoking the skill id: \'{targetSkill.Id}\' at \
{targetSkill.SkillEndpoint}\\" (status is {response.Status}). \r\n {response.Body}");
    }
}

```

Обратите внимание, что в корневом элементе бота предусмотрена логика для перенаправления действий в навык, запуска навыка по запросу пользователя и остановки навыка по завершении выполнения навыка.

```

protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    if (turnContext.Activity.Text.Contains("skill"))
    {
        await turnContext.SendActivityAsync(MessageFactory.Text("Got it, connecting you to the skill..."),
cancellationToken);

        // Save active skill in state
        await _activeSkillProperty.SetAsync(turnContext, _targetSkill, cancellationToken);

        // Send the activity to the skill
        await SendToSkill(turnContext, _targetSkill, cancellationToken);
        return;
    }

    // just respond
    await turnContext.SendActivityAsync(MessageFactory.Text("Me no nothin'. Say \"skill\" and I'll patch you
through"), cancellationToken);

    // Save conversation state
    await _conversationState.SaveChangesAsync(turnContext, force: true, cancellationToken:
cancellationToken);
}

```

```

protected override async Task OnEndOfConversationActivityAsync(ITurnContext<IEndOfConversationActivity>
turnContext, CancellationToken cancellationToken)
{
    // forget skill invocation
    await _activeSkillProperty.DeleteAsync(turnContext, cancellationToken);

    // Show status message, text and value returned by the skill
    var eocActivityMessage = $"Received {ActivityTypes.EndOfConversation}.\n\nCode:
{turnContext.Activity.Code}";
    if (!string.IsNullOrWhiteSpace(turnContext.Activity.Text))
    {
        eocActivityMessage += $"\n\nText: {turnContext.Activity.Text}";
    }

    if ((turnContext.Activity as Activity)?.Value != null)
    {
        eocActivityMessage += $"\n\nValue: {JsonConvert.SerializeObject((turnContext.Activity as
Activity)?.Value)}";
    }

    await turnContext.SendActivityAsync(MessageFactory.Text(eocActivityMessage), cancellationToken);

    // We are back at the root
    await turnContext.SendActivityAsync(MessageFactory.Text("Back in the root bot. Say \"skill\" and I'll
patch you through"), cancellationToken);

    // Save conversation state
    await _conversationState.SaveChangesAsync(turnContext, cancellationToken: cancellationToken);
}

```

## Глобальный обработчик ошибок с репликами

При возникновении ошибки адаптер очищает состояние беседы, чтобы сбросить параметры беседы с пользователем и избавиться от состояния ошибки.

Мы рекомендуем всегда отправлять сообщение *о завершении беседы* всем активным навыкам, прежде чем очищать состояние беседы в потребителе навыка. Это позволит навыку освободить все ресурсы, связанные с беседой между потребителем и навыком прежде, чем потребитель очистит эту беседу.

- [C#](#)
- [JavaScript](#)
- [Python](#)

### SimpleRootBot\AdapterWithErrorHandler.cs

В нашем примере логика обработки ошибок с репликами распределена между несколькими вспомогательными методами.

```
private async Task HandleTurnError(ITurnContext turnContext, Exception exception)
{
    // Log any leaked exception from the application.
    // NOTE: In production environment, you should consider logging this to
    // Azure Application Insights. Visit https://aka.ms/bottelemetry to see how
    // to add telemetry capture to your bot.
    _logger.LogError(exception, $"[OnTurnError] unhandled error : {exception.Message}");

    await SendErrorMessageAsync(turnContext, exception);
    await EndSkillConversationAsync(turnContext);
    await ClearConversationStateAsync(turnContext);
}

private async Task SendErrorMessageAsync(ITurnContext turnContext, Exception exception)
{
    try
    {
        // Send a message to the user
        var errorMessageText = "The bot encountered an error or bug.";
        var errorMessage = MessageFactory.Text(errorMessageText, errorMessageText,
InputHints.IgnoringInput);
        await turnContext.SendActivityAsync(errorMessage);

        errorMessageText = "To continue to run this bot, please fix the bot source code.";
        errorMessage = MessageFactory.Text(errorMessageText, errorMessageText, InputHints.ExpectingInput);
        await turnContext.SendActivityAsync(errorMessage);

        // Send a trace activity, which will be displayed in the Bot Framework Emulator
        await turnContext.TraceActivityAsync("OnTurnError Trace", exception.ToString(),
"https://www.botframework.com/schemas/error", "TurnError");
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, $"Exception caught in SendErrorMessageAsync : {ex}");
    }
}

private async Task EndSkillConversationAsync(ITurnContext turnContext)
{
    if (_skillClient == null || _skillsConfig == null)
    {
        return;
    }

    try
    {
        // Inform the active skill that the conversation is ended so that it has
        // a chance to clean up.
        // Note: ActiveSkillPropertyName is set by the RootBot while messages are being
        // forwarded to a Skill.
        var activeSkill = await _conversationState.CreateProperty<BotFrameworkSkill>
(RootBot.ActiveSkillPropertyName).GetAsync(turnContext, () => null);
        if (activeSkill != null)
        {
            var botId = _configuration.GetSection(MicrosoftAppCredentials.MicrosoftAppIdKey)?.Value;

            var endOfConversation = Activity.CreateEndOfConversationActivity();
    
```

```

        endOfConversation.Code = "RootSkillError";
        endOfConversation.ApplyConversationReference(turnContext.Activity.GetConversationReference(),
true);

        await _conversationState.SaveChangesAsync(turnContext, true);
        await _skillClient.PostActivityAsync(botId, activeSkill, _skillsConfig.SkillHostEndpoint,
(Activity)endOfConversation, CancellationToken.None);
    }
}
catch (Exception ex)
{
    _logger.LogError(ex, $"Exception caught on attempting to send EndOfConversation : {ex}");
}
}

private async Task ClearConversationStateAsync(ITurnContext turnContext)
{
    try
    {
        // Delete the conversationState for the current conversation to prevent the
        // bot from getting stuck in a error-loop caused by being in a bad state.
        // ConversationState should be thought of as similar to "cookie-state" in a Web pages.
        await _conversationState.DeleteAsync(turnContext);
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, $"Exception caught on attempting to Delete ConversationState : {ex}");
    }
}
}

```

## Конечная точка навыка

Бот определяет конечную точку, которая перенаправляет входящие действия навыка в обработчик навыка корневого бота.

- [C#](#)
- [JavaScript](#)
- [Python](#)

`SimpleRootBot\Controllers\SkillController.cs`

```

[ApiController]
[Route("api/skills")]
public class SkillController : ChannelServiceController
{
    public SkillController(ChannelServiceHandler handler)
        : base(handler)
    {
    }
}

```

## Регистрация службы

Добавьте объект конфигурации проверки подлинности со всеми необходимыми проверками утверждений, а также любые дополнительные объекты. В этом примере используется та же логика конфигурации проверки подлинности для проверки действий как пользователей, так и навыков.

- [C#](#)
- [JavaScript](#)
- [Python](#)

## SimpleRootBot\Startup.cs

```
// This method gets called by the runtime. Use this method to add services to the container.
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers().AddNewtonsoftJson();

    // Configure credentials
    services.AddSingleton<ICredentialProvider, ConfigurationCredentialProvider>();

    // Register the skills configuration class
    services.AddSingleton<SkillsConfiguration>();

    // Register AuthConfiguration to enable custom claim validation.
    services.AddSingleton(sp => new AuthenticationConfiguration { ClaimsValidator = new
        AllowedSkillsClaimsValidator(sp.GetService<SkillsConfiguration>()) });

    // Register the Bot Framework Adapter with error handling enabled.
    // Note: some classes use the base BotAdapter so we add an extra registration that pulls the same
    instance.
    services.AddSingleton<BotFrameworkHttpAdapter, AdapterWithErrorHandler>();
    services.AddSingleton<BotAdapter>(sp => sp.GetService<BotFrameworkHttpAdapter>());

    // Register the skills client and skills request handler.
    services.AddSingleton<SkillConversationIdFactoryBase, SkillConversationIdFactory>();
    services.AddHttpClient<SkillHttpClient>();
    services.AddSingleton<ChannelServiceHandler, SkillHandler>();

    // Register the storage we'll be using for User and Conversation state. (Memory is great for testing
    purposes.)
    services.AddSingleton<IStorage, MemoryStorage>();

    // Register Conversation state (used by the Dialog system itself).
    services.AddSingleton<ConversationState>();

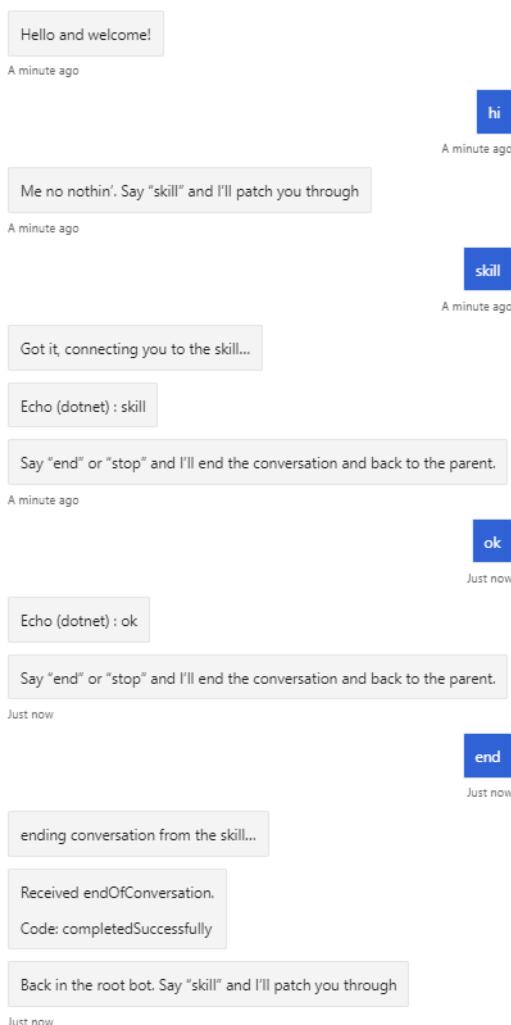
    // Register the bot as a transient. In this case the ASP Controller is expecting an IBot.
    services.AddTransient<IBot, RootBot>();
}
```

## Тестирование корневого бота

Вы можете протестировать потребитель навыка в эмуляторе, как любой обычный бот, но при этом навык и потребитель навыка должны одновременно работать в режиме ботов. Сведения о настройке навыка см. в статье [о реализации навыка](#).

Скачайте и установите последнюю версию [Bot Framework Emulator](#).

1. Запустите бот эхо-навыка и простой корневой бот на локальном компьютере. Дополнительные инструкции для [C#](#), [JavaScript](#) и [Python](#) см. в файле сведений соответствующего примера.
2. Примените эмулятор для тестирования бота, как показано ниже. Обратите внимание, что при отправке навыку сообщения `end` ИЛИ `stop` он отправляет корневому боту ответное сообщение, дополненное действием `endOfConversation`. Свойство `code` действия `endOfConversation` указывает, что навык успешно завершен.



Click on a log item in the panel below to inspect activity.

You can also inspect the JSON responses from your LUIS and QnA Maker services by selecting a "tr" [Learn More](#).

```
[11:19:54] <- message Hello and welcome!
[11:19:54] POST 200 conversations.:conversationId.activities.:activityId
[11:19:54] POST 200 directline.conversationUpdate
[11:19:54] POST 201 directline.conversations
[11:19:54] Emulator listening on http://[::]:49763
[11:19:54] ngrok not configured (only needed when connecting to remotely hosted)
[11:19:54] Connecting to bots hosted remotely
[11:19:54] Edit ngrok settings
[11:19:58] -> message hi
[11:19:58] <- message Me no nothin'. Say "skill" and I'll patch you thro...
[11:19:58] POST 200 conversations.:conversationId.activities.:activityId
[11:19:58] POST 200 directline.conversations.:conversationId.activities
[11:20:02] -> message skill
[11:20:02] <- message Got it, connecting you to the skill...
[11:20:02] POST 200 conversations.:conversationId.activities.:activityId
[11:20:02] <- message Echo (dotnet) : skill
[11:20:02] POST 200 conversations.:conversationId.activities.:activityId
[11:20:02] <- message Say "end" or "stop" and I'll end the conversation ...
[11:20:02] POST 200 conversations.:conversationId.activities.:activityId
[11:20:02] POST 200 directline.conversations.:conversationId.activities
[11:20:06] -> message ok
[11:20:06] <- message Echo (dotnet) : ok
[11:20:06] POST 200 conversations.:conversationId.activities.:activityId
[11:20:06] <- message Say "end" or "stop" and I'll end the conversation ...
[11:20:06] POST 200 conversations.:conversationId.activities.:activityId
[11:20:06] POST 200 directline.conversations.:conversationId.activities
[11:20:06] -> message end
[11:20:09] <- message ending conversation from the skill...
[11:20:09] POST 200 conversations.:conversationId.activities.:activityId
[11:20:09] <- message Received endOfConversation. Code: completedSuccess...
[11:20:09] POST 200 conversations.:conversationId.activities.:activityId
[11:20:09] <- message Back in the root bot. Say "skill" and I'll patch y...
[11:20:09] POST 200 conversations.:conversationId.activities.:activityId
[11:20:09] POST 200 directline.conversations.:conversationId.activities
```

## Дополнительные сведения

Ниже приведены некоторые аспекты, которые нужно учитывать при реализации более сложного корневого бота.

### Возможность отменить выполнение многоэтапного навыка

Корневой бот должен проверять сообщение пользователя, прежде чем перенаправить его активному навыку. Если пользователь хочет отменить текущий процесс, корневой бот отправляет в навык действие `endOfConversation`, а не само сообщение пользователя.

### Обмен данными между корневым ботом и навыком

Чтобы отправить параметры навыку, потребитель навыка может задать свойство `value` для сообщений, отправляемых в навык. Чтобы получить возвращаемые значения от навыка, потребитель навыка должен проверять свойство `value`, когда получает действие `endOfConversation` от навыка.

### Использование нескольких навыков

- Если навык уже активен, корневой бот должен определить активный навык и перенаправить сообщение пользователя в нужный навык.
- Если активных навыков нет, корневой бот должен определить навык для запуска (при его наличии), используя сведения о состоянии бота и сообщении пользователя.
- Если вы хотите, чтобы пользователь мог переключаться между несколькими параллельно выполняющимися навыками, корневой бот должен определять, с каким из активных навыков намерен взаимодействовать пользователь, прежде чем перенаправлять сообщение от пользователя.

## **Использование режима доставки «ожидание ответов»**

Для использования режима доставки *ожидание ответов*:

- Клонировать действие из контекста "выключить".
- Задайте для свойства *режим доставки* нового действия значение "експектреплиес" перед отправкой действия из корневого элемента Bot в навык.
- Считывает *ожидаемые ответы* из текста *ответа вызова*, возвращенного ответом запроса.
- Обработайте каждое действие либо в корневом роботе, либо путем отправки в канал, который инициировал исходный запрос.

Ожидание ответов может быть полезным в ситуациях, когда робот, отвечающий на действие, должен быть тем же экземпляром робота, который получил действие.

# Использование навыка с помощью диалога

27.03.2021 • 27 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье показано, как использовать *диалог навыка* в потребителе навыка. Диалог навыка отправляет действия из родительского робота в бот навыка и возвращает пользователю ответы от этого навыка. Бот навыка, к которому обращается этот потребитель, может обрабатывать действия сообщений и событий. Пример манифеста навыка и сведения о реализации этого навыка см. в статье о [применении диалогов в навыке](#).

Пример использования бота навыка без диалогов см. в статье о [реализации потребителя навыка](#).

## Предварительные требования

- Понимание [принципов работы ботов](#), [принципов работы ботов навыков](#) и [принципов реализации потребителя навыка](#).
- При необходимости — подписка Azure. Если у вас еще нет подписки Azure, создайте [бесплатную учетную запись](#), прежде чем начать работу.
- Копия примера `skills skillDialog` для [C#](#), [JavaScript](#) или [Python](#).

### NOTE

Начиная с версии 4.11 вам не требуется идентификатор приложения и пароль для проверки уровня потребителя на локальном компьютере в эмуляторе. Подписка Azure по-прежнему необходима для развертывания потребителя в Azure или для использования развернутого навыка.

## Об этом примере

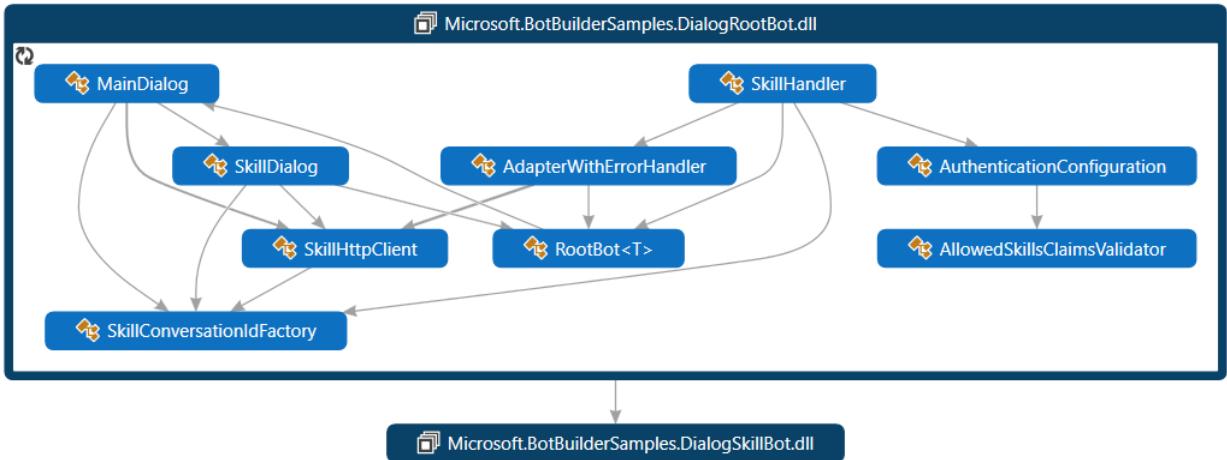
В пример `skills skillDialog` включены проекты двух ботов:

- *Корневой бот диалога*, который использует *диалог навыка* для применения навыка.
- *Бот навыка диалога*, который использует диалог для обработки действий, поступающих от потребителей навыков.

В этой статье основное внимание уделяется использованию класса *диалога навыка* в корневом боте для управления навыком, отправки действий сообщений и событий, а также отмены навыка.

Сведения о других аспектах создания потребителя навыка см. в статье о [реализации потребителя навыка](#).

- [C#](#)
- [JavaScript](#)
- [Python](#)



Сведения о боте навыка диалога см. в [этой статье](#).

## Ресурсы

Для развернутого программы-роботы проверка подлинности «Bot-to-Bot» требует, чтобы каждый участвующий робот получил допустимые идентификатор приложения и пароль. Тем не менее вы можете проверить навыки и квалификацию потребителей локально с помощью эмулятора без идентификатора приложения и пароля.

## Конфигурация приложений

- При необходимости добавьте идентификатор приложения и пароль корневого элемента Bot в файл конфигурации.
- Добавьте конечную точку узла навыка (служба или URL-адрес обратного вызова), на которую навыки должны отвечать на потребителей навыков.
- Добавьте запись для каждого навыка, который будет использоваться потребителем навыка. Каждая запись содержит следующие сведения:
  - идентификатор, который потребитель навыка использует для идентификации навыка;
  - При необходимости — идентификатор приложения навыка.
  - конечная точка обмена сообщениями для навыка.

### NOTE

Если потребитель или навык используют идентификатор приложения и пароль, оба они должны иметь значение.

- [C#](#)
- [JavaScript](#)
- [Python](#)

### DialogRootBot\appsettings.json

При необходимости добавьте идентификатор и пароль корневого приложения Bot и добавьте идентификатор приложения для программы-робота по навыку Echo в `BotFrameworkSkills` массив.

```
{
  "MicrosoftAppId": "TODO: Add here the App ID for the bot",
  "MicrosoftAppPassword": "TODO: Add here the password for the bot",
  "SkillHostEndpoint": "http://localhost:3978/api/skills/",
  "BotFrameworkSkills": [
    {
      "Id": "DialogSkillBot",
      "AppId": "TODO: Add here the App ID for the skill",
      "SkillEndpoint": "http://localhost:39783/api/messages"
    }
  ]
}
```

## Логика диалога

Основной диалог бота подключает *диалог навыка* для каждого навыка, потребляемого этим ботом. Диалог навыка управляет для вас навыком через специальные объекты, такие как *клиент навыка* и *фабрика идентификаторов бесед навыков*. В основном диалоге также показано, как отменить навык (через диалог навыка) на основе вводимых пользователем данных.

Навык, используемый этим ботом, поддерживает несколько функций. Он может забронировать авиабилет и получить прогноз погоды для города. Кроме того, если он получает сообщение за пределами одного из этих контекстов и при этом настроен распознаватель LUIS, он пытается интерпретировать намерение пользователя.

Манифест навыка (для [C#](#), [JavaScript](#), [Python](#)) содержит описание действий, которые может выполнять навык, его входных и выходных параметров, а также конечных точек навыка. Обратите внимание, что навык умеет обрабатывать события BookFlight и GetWeather. Он также обрабатывает сообщения.

Основной диалог содержит код для следующих действий:

- [инициализация основного диалога](#);
- [выбор навыка](#);
- [выбор действия навыка](#);
- [запуск навыка](#);
- [оценка результата выполнения навыка](#);
- [отмена навыка пользователем](#).

Основной диалог наследуется от класса *компонентного диалога*. Дополнительные сведения о компонентных диалогах см. в статье об [управлении сложностью диалогов](#).

### Инициализация основного диалога

Основной диалог содержит простые диалоги (для управления потоком беседы за пределами навыка) и диалоги навыков (для управления навыками). Каскадный диалог содержит перечисленные ниже шаги, каждый из которых подробно описывается в следующих разделах.

1. Предложение навыка на выбор пользователю. (Корневой бот использует один навык.)
2. Предложение действий для этого навыка на выбор пользователю. (Бот навыка определяет три действия.)
3. Запуск выбранного навыка с исходным действием на основе выбранного действия.
4. Отображение результата после выполнения навыка, если применимо. И, наконец, перезапуск каскадного диалога.

- [C#](#)
- [JavaScript](#)

- [Python](#)

## DialogRootBot\Dialogs\MainDialog.cs

Класс `MainDialog` является производным от `ComponentDialog`. Кроме сведений о состоянии, диалогу требуется идентификатор приложения корневого бота и ссылки на фабрику идентификаторов диалога навыка, HTTP-клиент навыка и объекты конфигурации навыков.

Конструктор диалога проверяет входные параметры, добавляет диалоги навыков, запрос и каскадные диалоги для управления потоком беседы за пределами навыка, а затем создает метод доступа свойства для отслеживания активного навыка, если применимо.

Конструктор вызывает вспомогательный метод `AddSkillDialogs`, чтобы создать `SkillDialog` для каждого навыка, включенного в файл конфигурации, по мере считывания из файла конфигурации в объект `SkillsConfiguration`.

```
// Helper method that creates and adds SkillDialog instances for the configured skills.  
private void AddSkillDialogs(ConversationState conversationState, SkillConversationIdFactoryBase  
conversationIdFactory, SkillHttpClient skillClient, SkillsConfiguration skillsConfig, string botId)  
{  
    foreach (var skillInfo in _skillsConfig.Skills.Values)  
    {  
        // Create the dialog options.  
        var skillDialogOptions = new SkillDialogOptions  
        {  
            BotId = botId,  
            ConversationIdFactory = conversationIdFactory,  
            SkillClient = skillClient,  
            SkillHostEndpoint = skillsConfig.SkillHostEndpoint,  
            ConversationState = conversationState,  
            Skill = skillInfo  
        };  
  
        // Add a SkillDialog for the selected skill.  
        AddDialog(new SkillDialog(skillDialogOptions, skillInfo.Id));  
    }  
}
```

## Выбор навыка

На первом шаге в основном диалоге пользователю предлагается выбрать, какой навык будет вызван, чтобы с помощью запроса на выбор `SkillPrompt` получить ответ. (Наш пример бота определяет только один навык.)

- [C#](#)
- [JavaScript](#)
- [Python](#)

## DialogRootBot\Dialogs\MainDialog.cs

```

// Render a prompt to select the skill to call.
private async Task<DialogTurnResult> SelectSkillStepAsync(WaterfallStepContext stepContext,
CancellationToken cancellationToken)
{
    // Create the PromptOptions from the skill configuration which contain the list of configured skills.
    var messageText = stepContext.Options?.ToString() ?? "What skill would you like to call?";
    var repromptMessageText = "That was not a valid choice, please select a valid skill.";
    var options = new PromptOptions
    {
        Prompt = MessageFactory.Text(messageText, messageText, InputHints.ExpectingInput),
        RetryPrompt = MessageFactory.Text(repromptMessageText, repromptMessageText,
InputHints.ExpectingInput),
        Choices = _skillsConfig.Skills.Select(skill => new Choice(skill.Value.Id)).ToList()
    };

    // Prompt the user to select a skill.
    return await stepContext.PromptAsync("SkillPrompt", options, cancellationToken);
}

```

## Выбор действия навыка

На следующем шаге основной диалог выполняет следующее:

- Сохраняет сведения о навыке, выбранном пользователем.
- Пользователю предлагается выбрать, какой навык будет вызван, чтобы с помощью запроса на выбор SkillActionPrompt получить ответ.
  - Он использует вспомогательный метод для получения списка действий, из которых осуществляется выбор.
  - Проверяющий элемент управления, связанный с этим запросом, по умолчанию отправит в навык сообщение, если введенные пользователем данные не совпадут с одним из доступных вариантов.

Варианты, предложенные в нашем боте, помогают проверить действия, определенные для этого навыка. В большинстве случаев эти вариантычитываются из манифеста навыка и предоставляются пользователю на основе этого списка.

- [C#](#)
- [JavaScript](#)
- [Python](#)

`DialogRootBot\Dialogs\MainDialog.cs`

```

// Render a prompt to select the action for the skill.
private async Task<DialogTurnResult> SelectSkillActionStepAsync(WaterfallStepContext stepContext,
CancellationToken cancellationToken)
{
    // Get the skill info based on the selected skill.
    var selectedSkillId = ((FoundChoice)stepContext.Result).Value;
    var selectedSkill = _skillsConfig.Skills.FirstOrDefault(s => s.Value.Id == selectedSkillId).Value;

    // Remember the skill selected by the user.
    stepContext.Values[_selectedSkillKey] = selectedSkill;

    // Create the PromptOptions with the actions supported by the selected skill.
    var messageText = $"Select an action # to send to **{selectedSkill.Id}** or just type in a message and
it will be forwarded to the skill";
    var options = new PromptOptions
    {
        Prompt = MessageFactory.Text(messageText, messageText, InputHints.ExpectingInput),
        Choices = GetSkillActions(selectedSkill)
    };

    // Prompt the user to select a skill action.
    return await stepContext.PromptAsync("SkillActionPrompt", options, cancellationToken);
}

```

```

// Helper method to create Choice elements for the actions supported by the skill.
private IList<Choice> GetSkillActions(BotFrameworkSkill skill)
{
    // Note: the bot would probably render this by reading the skill manifest.
    // We are just using hardcoded skill actions here for simplicity.

    var choices = new List<Choice>();
    switch (skill.Id)
    {
        case "DialogSkillBot":
            choices.Add(new Choice(SkillActionBookFlight));
            choices.Add(new Choice(SkillActionBookFlightWithInputParameters));
            choices.Add(new Choice(SkillActionGetWeather));
            break;
    }

    return choices;
}

```

```

// This validator defaults to Message if the user doesn't select an existing option.
private Task<bool> SkillActionPromptValidator(PromptValidatorContext<FoundChoice> promptContext,
CancellationToken cancellationToken)
{
    if (!promptContext.Recognized.Succeeded)
    {
        // Assume the user wants to send a message if an item in the list is not selected.
        promptContext.Recognized.Value = new FoundChoice { Value = SkillActionMessage };
    }

    return Task.FromResult(true);
}

```

## Запуск навыка

На следующем шаге основной диалог выполняет следующее:

1. Извлекает сведения о навыке и действии навыка, выбранных пользователем.
2. Применяет вспомогательный метод для создания действия, которое будет изначально отправлено в

навык.

3. Создает параметры диалога, с которыми будет запущен этот диалог навыка. Сюда входит и исходное действие для отправки.
4. Сохраняет состояние перед вызовом навыка. (Это необходимо, так как ответ навыка может поступить в другой экземпляр потребителя навыка.)
5. Запускает диалог навыка, передавая идентификатор вызываемого навыка и параметры для этого вызова.

- [C#](#)
- [JavaScript](#)
- [Python](#)

#### DialogRootBot\Dialogs\MainDialog.cs

```
// Starts the SkillDialog based on the user's selections.
private async Task<DialogTurnResult> CallSkillActionStepAsync(WaterfallStepContext stepContext,
CancellationToken cancellationToken)
{
    var selectedSkill = (BotFrameworkSkill)stepContext.Values[_selectedSkillKey];

    Activity skillActivity;
    switch (selectedSkill.Id)
    {
        case "DialogSkillBot":
            skillActivity = CreateDialogSkillBotActivity(((FoundChoice)stepContext.Result).Value,
stepContext.Context);
            break;

        // We can add other case statements here if we support more than one skill.
        default:
            throw new Exception($"Unknown target skill id: {selectedSkill.Id}.");
    }

    // Create the BeginSkillDialogOptions and assign the activity to send.
    var skillDialogArgs = new BeginSkillDialogOptions { Activity = skillActivity };

    // Save active skill in state.
    await _activeSkillProperty.SetAsync(stepContext.Context, selectedSkill, cancellationToken);

    // Start the skillDialog instance with the arguments.
    return await stepContext.BeginDialogAsync(selectedSkill.Id, skillDialogArgs, cancellationToken);
}
```

#### Оценка результата выполнения навыка

На последнем шаге основной диалог выполняет следующее:

1. Если навык вернул значение, отображает этот результат пользователю.
2. Удаляет активный навык из состояния диалога.
3. Удаляет активное свойство навыка из состояния беседы.
4. Перезапускает сам себя.

- [C#](#)
- [JavaScript](#)
- [Python](#)

#### DialogRootBot\Dialogs\MainDialog.cs

```

// The SkillDialog has ended, render the results (if any) and restart MainDialog.
private async Task<DialogTurnResult> FinalStepAsync(WaterfallStepContext stepContext, CancellationToken cancellationToken)
{
    var activeSkill = await _activeSkillProperty.GetAsync(stepContext.Context, () => null, cancellationToken);

    // Check if the skill returned any results and display them.
    if (stepContext.Result != null)
    {
        var message = $"Skill \"{activeSkill.Id}\" invocation complete.";
        message += $" Result: {JsonConvert.SerializeObject(stepContext.Result)}";
        await stepContext.Context.SendActivityAsync(MessageFactory.Text(message, message, inputHint:
InputHints.IgnoringInput), cancellationToken: cancellationToken);
    }

    // Clear the skill selected by the user.
    stepContext.Values[_selectedSkillKey] = null;

    // Clear active skill in state.
    await _activeSkillProperty.DeleteAsync(stepContext.Context, cancellationToken);

    // Restart the main dialog with a different message the second time around.
    return await stepContext.ReplaceDialogAsync(InitialDialogId, $"Done with \"{activeSkill.Id}\". \n\n What
skill would you like to call?", cancellationToken);
}

```

## Отмена навыка пользователем

Основной диалог переопределяет стандартное поведение метода *on continue dialog*, чтобы пользователь мог отменить текущий навык, если применимо. Этот метод позволяет выполнить следующее:

- Если существует активный навык и пользователь отправляет сообщение `abort`, отмените все диалоги и поместите основной диалог в очередь на перезапуск с самого начала.
- После этого вызовите реализацию метода *on continue dialog*, чтобы завершить обработку текущего шага.
- [C#](#)
- [JavaScript](#)
- [Python](#)

## DialogRootBot\Dialogs\MainDialog.cs

```

protected override async Task<DialogTurnResult> OnContinueDialogAsync(DialogContext innerDc,
CancellationToken cancellationToken = default)
{
    // This is an example on how to cancel a SkillDialog that is currently in progress from the parent bot.
    var activeSkill = await _activeSkillProperty.GetAsync(innerDc.Context, () => null, cancellationToken);
    var activity = innerDc.Context.Activity;
    if (activeSkill != null && activity.Type == ActivityTypes.Message && activity.Text.Equals("abort",
 StringComparison.CurrentCultureIgnoreCase))
    {
        // Cancel all dialogs when the user says abort.
        // The SkillDialog automatically sends an EndOfConversation message to the skill to let the
        // skill know that it needs to end its current dialogs, too.
        await innerDc.CancelAllDialogsAsync(cancellationToken);
        return await innerDc.ReplaceDialogAsync(InitialDialogId, "Canceled! \n\n What skill would you like
to call?", cancellationToken);
    }

    return await base.OnContinueDialogAsync(innerDc, cancellationToken);
}

```

## Логика обработчика действий

Так как логика навыка для каждого шага обрабатывается основным диалогом, обработчик действия выглядит примерно так же, как в остальных примерах диалогов.

- [C#](#)
- [JavaScript](#)
- [Python](#)

### DialogRootBot\Bots\RootBot.cs

```
public class RootBot<T> : ActivityHandler
    where T : Dialog

private readonly ConversationState _conversationState;
private readonly Dialog _mainDialog;

public RootBot(ConversationState conversationState, T mainDialog)
{
    _conversationState = conversationState;
    _mainDialog = mainDialog;
}

public override async Task OnTurnAsync(ITurnContext turnContext, CancellationToken cancellationToken =
default)
{
    if (turnContext.Activity.Type != ActivityTypes.ConversationUpdate)
    {
        // Run the Dialog with the Activity.
        await _mainDialog.RunAsync(turnContext, _conversationState.CreateProperty<DialogState>
("DialogState"), cancellationToken);
    }
    else
    {
        // Let the base class handle the activity.
        await base.OnTurnAsync(turnContext, cancellationToken);
    }

    // Save any state changes that might have occurred during the turn.
    await _conversationState.SaveChangesAsync(turnContext, false, cancellationToken);
}
```

## Регистрация службы

Для использования диалога навыка нужны те же службы, что и для самого потребителя навыка. Требуемые службы обсуждаются в статье о [реализации потребителя навыка](#).

## Тестирование корневого бота

Вы можете протестировать потребитель навыка в эмуляторе, как любой обычный бот, но при этом навык и потребитель навыка должны одновременно работать в режиме ботов. Сведения о настройке навыка см. в статье об [использовании диалогов в навыке](#).

Скачайте и установите последнюю версию [Bot Framework Emulator](#).

1. Запустите бот навыка диалога и корневой бот диалога локально на компьютере. Дополнительные инструкции для [C#](#), [JavaScript](#) и [Python](#) см. в файле сведений соответствующего примера.

2. Примените эмулятор для тестирования бота.
  - Когда вы впервые присоединяетесь к беседе, бот отображает приветственное сообщение и спрашивает, какой навык вы хотите вызвать. В нашем примере бота навыков есть всего один навык.
  - Выберите **DialogSkillBot**.
3. Теперь бот предложит вам выбрать действие для этого навыка. Выберите **BookFlight**.
  - a. Ответьте на вопросы.
  - b. Когда навык завершит работу, корневой бот отобразит сведения о резервировании и снова предложит вам вызвать навык.
4. Снова выберите **DialogSkillBot** и **BookFlight**.
  - a. Ответьте на первый вопрос, а затем введите `abort`, чтобы прервать навык.
  - b. Корневой бот отменит текущий навык и предложит вам вызвать новый.

## Дополнительные сведения

См. сведения в статье о [реализации потребителя навыка](#).

# Добавление телеметрии в бот

27.03.2021 • 24 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В версии 4.2 пакета SDK для Bot Framework теперь можно вести журнал телеметрии. Так приложения ботов смогут отправлять данные о событиях в службы телеметрии, например [Application Insights](#). Данные телеметрии включают полезные сведения о боте (например о том, какие функции используются чаще всего), помогают обнаруживать нежелательное поведение и предоставляют сведения о доступности, производительности и использовании.

**Примечание.** В версии 4.6 стандартный метод реализации телеметрии в ботах был обновлен для обеспечения правильной записи телеметрии с пользовательским адаптером. Эта статья была обновлена с учетом этого обновления метода. Изменения сохраняют обратную совместимость, и боты с прежним методом будут и далее правильно вести журнал телеметрии.

В этой статье вы узнаете, как реализовать телеметрию в Bot с помощью Application Insights. Будет рассмотрено следующее:

- Код, необходимый для подключения телеметрических данных в коде робота и соединения с Application Insights.
- Включение телеметрии в [диалоговых окнах](#)Bot.
- настройка телеметрии для сбора данных об использовании из других служб, например [LUIS](#) и [QnA Maker](#);
- Визуализация данных телеметрии в Application Insights.

## Предварительные требования

- [C#](#)
- [JavaScript](#)
- [Пример кода CoreBot](#).
- [Пример кода Application Insights](#).
- Подписка на [Microsoft Azure](#).
- [Ключ Application Insights](#).
- Опыт работы с [Application Insights](#).
- [git](#)

### NOTE

Пример кода Application Insights основан на [примере кода CoreBot](#). В этой статье показано, как изменить пример кода CoreBot для включения телеметрии. Если вы используете Visual Studio в качестве примера, вы получите Application Insights пример кода на момент завершения.

## Включение телеметрии в боте

- [C#](#)

- [JavaScript](#)

Эта статья начинается из [примера приложения коробот](#) и добавляет код, необходимый для интеграции телеметрии в любую программу-робот. Так Application Insights сможет отслеживать запросы.

**IMPORTANT**

Если вы не настроили учетную запись Application Insights и не создали [ключ Application Insights](#), [сделайте это](#) перед продолжением.

1. Откройте [пример приложения коробот](#) в Visual Studio.
2. Добавление пакета NuGet `Microsoft.Bot.Builder.Integration.ApplicationInsights.Core`. Подробные сведения об использовании NuGet см. в руководстве по [установке пакетов и управлении ими в Visual Studio](#).
3. Добавьте следующие инструкции в `Startup.cs`:

```
using Microsoft.ApplicationInsights.Extensibility;
using Microsoft.Bot.Builder.ApplicationInsights;
using Microsoft.Bot.Builder.Integration.ApplicationInsights.Core;
using Microsoft.Bot.Builder.Integration.AspNet.Core;
```

Примечание. Выполняя эти инструкции, чтобы обновить пример кода CoreBot, вы заметите, что инструкция `using` для `Microsoft.Bot.Builder.Integration.AspNet.Core` уже используется в коде CoreBot.

4. Включите следующий код в метод `ConfigureServices()` в `Startup.cs`. Так службы телеметрии станут доступными для бота путем [внедрения зависимостей](#):

```
// This method gets called by the runtime. Use this method to add services to the container.
public void ConfigureServices(IServiceCollection services)
{
    ...
    // Create the Bot Framework Adapter with error handling enabled.
    services.AddSingleton<IBotFrameworkHttpAdapter, AdapterWithErrorHandler>();

    // Add Application Insights services into service collection
    services.AddApplicationInsightsTelemetry();

    // Create the telemetry client.
    services.AddSingleton<IBotTelemetryClient, BotTelemetryClient>();

    // Add telemetry initializer that will set the correlation context for all telemetry items.
    services.AddSingleton<ITelemetryInitializer, OperationCorrelationTelemetryInitializer>();

    // Add telemetry initializer that sets the user ID and session ID (in addition to other bot-specific properties such as activity ID)
    services.AddSingleton<ITelemetryInitializer, TelemetryBotIdInitializer>();

    // Create the telemetry middleware to initialize telemetry gathering
    services.AddSingleton<TelemetryInitializerMiddleware>();

    // Create the telemetry middleware (used by the telemetry initializer) to track conversation events
    services.AddSingleton<TelemetryLoggerMiddleware>();
    ...
}
```

Примечание. Если вы выполните следующие действия, обновив пример кода Коробот, вы увидите,

ЧТО `services.AddSingleton<IBotFrameworkHttpAdapter, AdapterWithErrorHandler>();` уже существует.

5. Настройте адаптер на использование кода ПО промежуточного слоя, который был добавлен в метод `ConfigureServices()`. Это делается в `AdapterWithErrorHandler.cs` с параметром `TelemetryInitializerMiddleware` телеметрии из сертификата в списке параметров конструктора и `Use(telemetryInitializerMiddleware);` инструкцией в конструкторе, как показано ниже:

```
public AdapterWithErrorHandler(IConfiguration configuration, ILogger<BotFrameworkHttpAdapter>
logger, TelemetryInitializerMiddleware telemetryInitializerMiddleware, ConversationState
conversationState = null)
    : base(configuration, logger)
{
    ...
    Use(telemetryInitializerMiddleware);
}
```

6. Кроме того, необходимо добавить `Microsoft.Bot.Builder.Integration.ApplicationInsights.Core` в список инструкций `using` в `AdapterWithErrorHandler.cs`.
7. Добавьте ключ инструментирования Application Insights в файл `appsettings.json`. `appsettings.json` Файл содержит метаданные о внешних службах, которые использует Bot во время работы. Например, здесь хранятся сведения о подключении и метаданные для служб CosmosDB, Application Insights и LUIS (распознавание речи). Формат добавляемого в файл `appsettings.json` ключа должен быть таким:

```
{
    "MicrosoftAppId": "",
    "MicrosoftAppPassword": "",
    "ApplicationInsights": {
        "InstrumentationKey": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
    }
}
```

Примечание. Сведения о *ключе инструментирования Application* см. в статье о [ключах Application Insights](#).

На этом этапе вы уже включили телеметрию с помощью Application Insights. Вы можете запустить Bot локально с помощью эмулятора Bot, а затем переходить к Application Insights, чтобы узнать, что записывается в журнал, например время отклика, общая работоспособность приложения и общие сведения о выполнении.

### Включение телеметрии в диалогах бота

При добавлении нового диалога в любой класс `ComponentDialog` он наследует `Microsoft.Bot.Builder.IBotTelemetryClient` из родительского диалога. Например, в примере приложения `CoreBot` все диалоги добавляются в класс `MainDialog`, который имеет тип `ComponentDialog`. После установки свойства `TelemetryClient` в `MainDialog` все добавленные в него диалоги будут автоматически наследовать от него значение `telemetryClient`, и вам не придется явным образом указывать его при добавлении диалогов.

Чтобы обновить код `CoreBot`, сделайте следующее:

1. Обновите список параметров конструктора в `MainDialog.cs`, включив в него параметр `IBotTelemetryClient`, а затем в `MainDialog` установите для свойства `TelemetryClient` это же значение, как показано в следующем фрагменте кода:

```
public MainDialog(IConfiguration configuration, ILogger<MainDialog> logger, IBotTelemetryClient telemetryClient)
    : base(nameof(MainDialog))
{
    // Set the telemetry client for this and all child dialogs.
    this.TelemetryClient = telemetryClient;
    ...
}
```

#### TIP

Если у вас возникли проблемы при изменении примера кода CoreBot, см. [пример кода Application Insights](#).

Мы добавили телеметрию в диалоги бота. При его запуске вы увидите сведения, регистрируемые в журнале Application Insights. Но если вы используете такие интегрированные технологии, как LUIS и QnA Maker, вам также нужно добавить `TelemetryClient` в этот код.

## Включение и отключение ведения журнала событий действий и персональных данных

- [C#](#)
- [JavaScript](#)

### Включение или отключение ведения журнала действий

По умолчанию `TelemetryInitializerMiddleware` будет использовать `TelemetryLoggerMiddleware` для регистрации данных телеметрии, когда бот отправляет или получает действия. Ведение журнала действий создает пользовательские журналы событий в ресурсе Application Insights. При желании вы можете отключить ведение журнала событий действий, установив значение `False` для `logActivityTelemetry` в `TelemetryInitializerMiddleware` при его регистрации в `Startup.cs`.

```
public void ConfigureServices(IServiceCollection services)
{
    ...
    // Add the telemetry initializer middleware
    services.AddSingleton<TelemetryInitializerMiddleware>(sp =>
    {
        var loggerMiddleware = sp.GetService<TelemetryLoggerMiddleware>();
        return new TelemetryInitializerMiddleware(loggerMiddleware, logActivityTelemetry: false);
    });
    ...
}
```

### Включение или отключение сохранения персональных данных в журнал

По умолчанию, если включено ведение журнала действий, некоторые свойства входящих или исходящих действий исключаются из журнала, так как они могут содержать персональные данные, например имя пользователя и текст действия. Вы можете включить сохранение этих свойств в журнал, внеся следующие изменения в `Startup.cs` при регистрации `TelemetryLoggerMiddleware`.

```
public void ConfigureServices(IServiceCollection services)
{
    ...
    // Add the telemetry initializer middleware
    services.AddSingleton<TelemetryLoggerMiddleware>(sp =>
    {
        var telemetryClient = sp.GetService<IBotTelemetryClient>();
        return new TelemetryLoggerMiddleware(telemetryClient, logPersonalInformation: true);
    });
    ...
}
```

Теперь нам нужно включить функцию телеметрии в диалоги. Так вы сможете получать дополнительные сведения о запущенных диалогах, а также собирать статистику по каждому из них.

## Настройка телеметрии для сбора данных об использовании из других служб, например LUIS и QnA Maker

- [C#](#)
- [JavaScript](#)

Теперь вы реализуете функции телеметрии в службе LUIS. Служба LUIS имеет встроенную функцию ведения журнала телеметрии, поэтому настроить получение данных телеметрии из LUIS очень просто. Если вы хотите включить телеметрию в боте с поддержкой QnA Maker, выполните инструкции из [этой статьи](#).

1. Параметр `IBotTelemetryClient telemetryClient` является обязательным в конструкторе

`FlightBookingRecognizer` в `FlightBookingRecognizer.cs`:

```
public FlightBookingRecognizer(IConfiguration configuration, IBotTelemetryClient telemetryClient)
```

2. Далее потребуется включить `telemetryClient` при создании `LuisRecognizer` в конструкторе

`FlightBookingRecognizer`. Это можно сделать, добавив в `telemetryClient` качестве нового `луисреконизероптион`.

```
if (luisIsConfigured)
{
    var luisApplication = new LuisApplication(
        configuration["LuisAppId"],
        configuration["LuisAPIKey"],
        "https://" + configuration["LuisAPIHostName"]);

    // Set the recognizer options depending on which endpoint version you want to use.
    // More details can be found in https://docs.microsoft.com/azure/cognitive-services/luis/luis-
    migration-api-v3
    var recognizerOptions = new LuisRecognizerOptionsV3(luisApplication)
    {
        TelemetryClient = telemetryClient,
    };
    _recognizer = new LuisRecognizer(recognizerOptions);
}
```

Теперь у вас есть функциональный бот, который записывает данные телеметрии в Application Insights. Запустить бота локально можно с помощью [Bot Framework Emulator](#). Вы не увидите изменений в поведении бота, но данные будут регистрироваться в Application Insights. Отправьте боту несколько сообщений. В следующем разделе описано, как просматривать результаты телеметрии в Application Insights.

Сведения о тестировании и отладке бота см. в таких статьях:

- [Отладка бота](#)
- [Рекомендации по тестированию и отладке](#)
- [Отладка с помощью эмулятора](#)

## Отображение данных телеметрии в Application Insights

Application Insights отслеживает доступность, производительность и использование бота независимо от размещения (в облачной или локальной среде). Она использует мощную платформу анализа данных в Azure Monitor, чтобы предоставлять подробные аналитические сведения о работе приложения и диагностировать ошибки еще до того, как пользователи сообщат о них. Данные телеметрии, собираемые Application Insights, можно просматривать несколькими способами: с помощью запросов и панели мониторинга.

### Получение данных телеметрии в Application Insights с помощью запросов Kusto

В этом разделе описано, как использовать запросы журналов в Application Insights с помощью двух запросов, а также предоставлены ссылки на другую документацию с дополнительными сведениями.

Чтобы запросить данные:

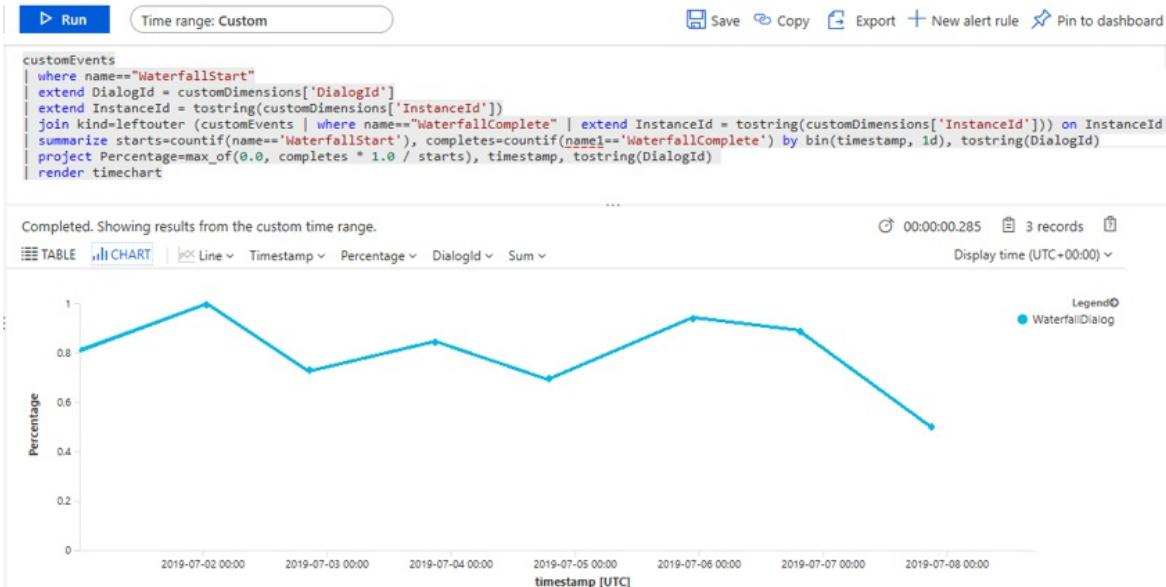
1. Перейдите на [портал Azure](#).
2. Перейдите к Application Insights. Для этого щелкните **Монитор > Приложения** и выполните поиск.
3. В Application Insights щелкните **Журналы (Analytics)** на панели навигации.

The screenshot shows the Azure Application Insights Analytics blade. At the top, there's a search bar labeled "Search (Ctrl+Shift+F)" and a "Logs (Analytics)" button highlighted with a red box. Below the search bar is a navigation menu with "Overview" selected, followed by "Activity log" and "Access control (IAM)". On the right side, there's a message: "Search and Logs (Analytics) will be removed from the top commands bar. They can be accessed from the left menu bar." Below this message, it shows the "Resource group (change) : CoreBot" and "Location : West US".

4. Откроется окно "Запрос". Введите следующий запрос и нажмите *Запуск*.

```
customEvents
| where name=="WaterfallStart"
| extend DialogId = customDimensions['DialogId']
| extend InstanceId = tostring(customDimensions['InstanceId'])
| join kind=leftouter (customEvents | where name=="WaterfallComplete" | extend InstanceId =
tostring(customDimensions['InstanceId'])) on InstanceId
| summarize starts=countif(name=='WaterfallStart'), completes=countif(name1=='WaterfallComplete') by
bin(timestamp, 1d), tostring(DialogId)
| project Percentage=max_of(0.0, completes * 1.0 / starts), timestamp, tostring(DialogId)
| render timechart
```

5. Отобразится процент каскадных диалоговых окон, которые выполнялись до завершения.

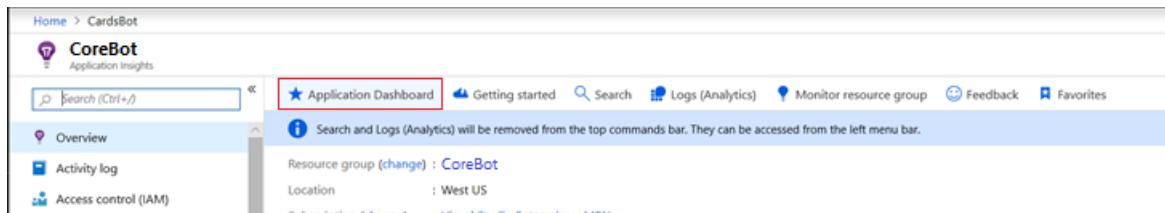


#### TIP

Любой запрос можно закрепить на панели мониторинга Application Insights, нажав на кнопку в верхней правой части колонки **Журналы (Analytics)**. Просто выберите нужную панель мониторинга, и вы сможете выбрать этот запрос при ее следующем открытии.

## Панель мониторинга Application Insights

Когда вы создаете ресурс Application Insights в Azure, автоматически создается и связанная панель мониторинга. Эту панель мониторинга можно просмотреть, нажав кнопку **Панель мониторинга приложений** в верхней части колонки Application Insights.



Просмотреть данные можно также на портале Azure. Щелкните слева **Панель мониторинга** и выберите нужную панель мониторинга в раскрывающемся списке.

Здесь вы увидите некоторые стандартные сведения о производительности бота и дополнительные запросы, которые вы закрепили на панели мониторинга.

## Дополнительные сведения

- [Добавление данных телеметрии в бота QnAMaker](#)
- [Что такое Azure Application Insights?](#)
- [Поиск в Application Insights](#)
- [Создание настраиваемых панелей мониторинга ключевых показателей эффективности с помощью Azure Application Insights](#)

# Добавление данных телеметрии в бота QnAMaker

27.03.2021 • 15 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В версии 4.2 пакета SDK для Bot Framework теперь можно вести журнал телеметрии. Так приложения ботов смогут отправлять данные о событиях в службы телеметрии, например [Application Insights](#). Данные телеметрии включают полезные сведения о боте (например о том, какие функции используются чаще всего), помогают обнаруживать нежелательное поведение и предоставляют сведения о доступности, производительности и использовании.

В пакет SDK для Bot Framework добавлены два новых компонента для ведения журнала телеметрии в ботах с поддержкой QnA Maker: `TelemetryLoggerMiddleware` и класс `QnAMaker`. `TelemetryLoggerMiddleware` представляет собой компонент ПО промежуточного слоя, который сохраняет в журнал данные о каждом полученном, отправленном, обновленном или удаленном сообщении, а класс `QnAMaker` поддерживает настраиваемое ведение журнала, расширяющее возможности телеметрии.

В этой статье вы изучите следующее:

- Код для подключения телеметрии к боту.
- Код для включения встроенных функций ведения журнала и отчетности QnA, которые используют стандартные свойства событий.
- Процедуру изменения или расширения стандартных свойств событий, предоставляемых пакетом SDK, для широкого спектра задач отчетности.

## Предварительные требования

- [Пример кода для QnA Maker](#).
- Подписка на [Microsoft Azure](#).
- [Ключ Application Insights](#).
- Вам будет проще, если вы уже знакомы со службой [QnA Maker](#).
- Учетная запись [QnA Maker](#).
- Опубликованная база знаний QnA Maker. Если у вас ее нет, выполните инструкции в статье [Руководство по созданию базы знаний на портале QnA Maker](#), чтобы создать базу знаний QnA Maker с вопросами и ответами.

### NOTE

В этой статье используется [пример кода для QnA Maker](#) и даны пошаговые инструкции по внедрению в него телеметрии.

## Подключение телеметрии к боту QnA Maker

Мы начнем с [примера приложения для QnA Maker](#) и добавим в него код, позволяющий интегрировать

тelemетрию в бота с помощью службы QnA. Так Application Insights сможет отслеживать запросы.

1. Откройте [пример кода для QnA Maker](#) в Visual Studio.
2. Добавление пакета NuGet `Microsoft.Bot.Builder.Integration.ApplicationInsights.Core`. Подробные сведения об использовании NuGet см. в руководстве по [установке пакетов и управлении ими в Visual Studio](#).
3. Добавьте следующие инструкции в `Startup.cs`:

```
using Microsoft.ApplicationInsights.Extensibility;
using Microsoft.Bot.Builder.ApplicationInsights;
using Microsoft.Bot.Builder.Integration.ApplicationInsights.Core;
```

#### NOTE

Выполняя эти инструкции, чтобы обновить пример кода для QnA Maker, вы заметите, что инструкция `using` для `Microsoft.Bot.Builder.Integration.AspNet.Core` уже включена в код для QnA Maker.

4. Добавьте в метод `ConfigureServices()` файла `Startup.cs` следующий код. Теперь службы телеметрии станут доступными для бота через [внедрение зависимостей](#):

```
// This method gets called by the runtime. Use this method to add services to the container.
public void ConfigureServices(IServiceCollection services)
{
    ...
    // Create the Bot Framework Adapter with error handling enabled.
    services.AddSingleton<IBotFrameworkHttpAdapter, AdapterWithErrorHandler>();

    // Add Application Insights services into service collection
    services.AddApplicationInsightsTelemetry();

    // Add the standard telemetry client
    services.AddSingleton<IBotTelemetryClient, BotTelemetryClient>();

    // Create the telemetry middleware to track conversation events
    services.AddSingleton<TelemetryLoggerMiddleware>();

    // Add the telemetry initializer middleware
    services.AddSingleton<IMiddleware, TelemetryInitializerMiddleware>();

    // Add telemetry initializer that will set the correlation context for all telemetry items
    services.AddSingleton<ITelemetryInitializer, OperationCorrelationTelemetryInitializer>();

    // Add telemetry initializer that sets the user ID and session ID (in addition to other bot-specific properties, such as activity ID)
    services.AddSingleton<ITelemetryInitializer, TelemetryBotIdInitializer>();
    ...
}
```

#### NOTE

Выполняя эти инструкции, чтобы обновить пример кода для QnA Maker, вы заметите, что `services.AddSingleton<IBotFrameworkHttpAdapter, AdapterWithErrorHandler>();` уже существует.

5. Настройте адаптер на использование кода ПО промежуточного слоя, который был добавлен в метод `ConfigureServices()`. Откройте `AdapterWithErrorHandler.cs` и добавьте `IMiddleware middleware` в список параметров конструктора. Добавьте инструкцию `Use(middleware);` последней строкой в

конструктор:

```
public AdapterWithErrorHandler(ICredentialProvider credentialProvider,
    ILogger<BotFrameworkHttpAdapter> logger, IMiddleware middleware, ConversationState conversationState
    = null)
    : base(credentialProvider)
{
    ...
    Use(middleware);
}
```

6. Добавьте ключ инструментирования Application Insights в файл `appsettings.json`. Файл `appsettings.json` содержит метаданные о внешних службах, которые бот использует в своей работе. Например, здесь хранятся сведения о подключении и метаданные для служб CosmosDB, Application Insights и QnA Maker. Формат добавляемого в файл `appsettings.json` ключа должен быть таким:

```
{
    "MicrosoftAppId": "",
    "MicrosoftAppPassword": "",
    "QnAKnowledgebaseId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
    "QnAEndpointKey": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
    "QnAEndpointHostName": "https://xxxxxxxx.azurewebsites.net/qnamaker",
    "ApplicationInsights": {
        "InstrumentationKey": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
    }
}
```

#### NOTE

- Сведения о *ключе инструментирования Application* см. в статье о [ключах Application Insights](#).
- У вас уже должна быть [учетная запись QnA Maker](#). при необходимости можно найти сведения о получении идентификатора базы знаний QnA, ключа конечной точки и имени узла в статье [Управление ключами](#).

На этом этапе вы уже включили телеметрию с помощью Application Insights. Вы можете запустить Bot локально с помощью эмулятора Bot, а затем переходить к Application Insights, чтобы увидеть, что именно регистрируется, например время отклика, общая работоспособность приложения и общие сведения о выполнении.

#### TIP

Включение и отключение ведения журнала событий действий и персональных данных описано в [этой статье](#).

Теперь нам нужно включить функцию телеметрии в службу QnA Maker.

## Настройка телеметрии для сбора данных о потреблении из службы QnA Maker

Служба QnA Maker имеет встроенную функцию ведения журнала телеметрии, поэтому настроить получение данных телеметрии из QnA Maker очень просто. Для начала мы узнаем, как включить телеметрию в код QnA Maker, чтобы использовать встроенную функцию ведения журнала телеметрии, а затем попробуем изменить и добавить свойства существующих данных о событиях для поддержки более

широкого спектра задач отчетности.

## Включение стандартной функции ведения журнала QnA

1. Создайте закрытое поле только для чтения с типом `IBotTelemetryClient` в классе `QnABot` в

`QnABot.cs`:

```
public class QnABot : ActivityHandler
{
    private readonly IBotTelemetryClient _telemetryClient;
    ...
}
```

2. Добавьте параметр `IBotTelemetryClient` в конструктор класса `QnABot` в файле `QnABot.cs` и сохраните его значение в закрытое поле, созданное на предыдущем шаге:

```
public QnABot(IConfiguration configuration, ILogger<QnABot> logger, IHttpClientFactory
httpClientFactory, IBotTelemetryClient telemetryClient)
{
    ...
    _telemetryClient = telemetryClient;
}
```

3. Параметр `telemetryClient` является обязательным при создании экземпляра объекта `QnAMaker` в

`QnABot.cs`:

```
var qnaMaker = new QnAMaker(new QnAMakerEndpoint
{
    KnowledgeBaseId = _configuration["QnAKnowledgebaseId"],
    EndpointKey = _configuration["QnAEndpointKey"],
    Host = _configuration["QnAEndpointHostName"]
},
null,
httpClient,
_telemetryClient);
```

### TIP

Убедитесь, что имена свойств, используемых в `_configuration` записях, соответствуют именам свойств, использованным в `AppSettings.json` для файла, и значения этих свойств получены путем нажатия кнопки *Просмотреть код* на странице [Мои базы знаний](#) на портале QnA Maker.

The QnABot.cs file:	The AppSettings.json file:	Settings from: <a href="https://www.qnamaker.ai/Home/MyServices">https://www.qnamaker.ai/Home/MyServices</a>
<pre>var qnaMaker = new QnAMaker(new QnAMakerEndpoint {     KnowledgeBaseId = _configuration["QnAKnowledgebaseId"],     EndpointKey = _configuration["QnAEndpointKey"],     Host = _configuration["QnAEndpointHostName"] }, null, httpClient, _telemetryClient);</pre>	<pre>{     "MicrosoftAppId": "",     "MicrosoftAppPassword": "",     "QnAKnowledgebaseId": "",     "QnAEndpointKey": "",     "QnAEndpointHostName": "" }</pre>	<p>Postman   Curl</p> <p>POST /knowledgebase/{knowledgebaseId}/generateAnswer Authorization: EndpointKey {endpointKey} Host: https://qnamaker-test.azuredashboards.net/qnamaker/ Content-Type: application/json {"question":&lt;Your question&gt;}</p>

### Просмотр данных телеметрии, полученных из записей QnA Maker по умолчанию

Результаты использования QnA Maker Bot можно просмотреть в Application Insights после запуска программы Bot в эмуляторе Bot, выполнив следующие действия.

1. Перейдите на [портал Azure](#).
2. Перейдите к Application Insights, щелкнув **Мониторинг > Приложения**.

3. В Application Insights щелкните **Журналы (аналитика)** на панели навигации, как показано ниже.

The screenshot shows the Azure Application Insights interface. At the top, there's a navigation bar with 'QnABot' and 'Application Insights'. Below it is a search bar labeled 'Search (Ctrl+ /)'. The top navigation bar has several items: 'Application Dashboard', 'Getting started', 'Search', 'Logs (Analytics)', and 'Monitor resource group'. The 'Logs (Analytics)' item is highlighted with a red box. A tooltip message says: 'Search and Logs (Analytics) will be removed from the top commands bar. They can be accessed from the left menu bar.' On the left, there's a sidebar with 'Overview', 'Activity log', and 'Access control (IAM)'. In the center, there's a message about a resource group change: 'Resource group (change) : QnABot' and 'Location : West US'. Below the sidebar, there's a code editor window containing Kusto query language:

```
customEvents  
| where name == 'QnaMessage'  
| extend answer = tostring(customDimensions.answer)  
| summarize count() by answer
```

4. Введите следующий запрос Kusto и щелкните **Запуск**.

```
customEvents  
| where name == 'QnaMessage'  
| extend answer = tostring(customDimensions.answer)  
| summarize count() by answer
```

5. Не закрывайте эту страницу в браузере. Мы вернемся к ней после добавления нового пользовательского свойства.

#### TIP

Если вы не знакомы с языком запросов Kusto, который используется для написания запросов по журналам в Azure Monitor, но знаете язык запросов SQL, вам будет полезна [памятка по применению SQL для запросов по журналам Azure Monitor](#).

### Изменение или расширение стандартных свойств событий

Если вам нужны свойства, которые отсутствуют в классе `QnAMaker`, у вас есть два способа решить эту задачу. Оба они требуют создания собственного класса, производного от класса `QnAMaker`. Первый способ описан в разделе [Добавление свойств](#) ниже, где свойства добавляются в существующее событие `QnAMessage`. Второй метод позволяет создавать новые события и добавлять в них свойства, как описано в разделе [Добавление новых событий с пользовательскими свойствами](#).

#### NOTE

Событие `QnAMessage` входит в состав пакета SDK для Bot Framework и предоставляет все свойства событий, которые сохраняются в Application Insights.

#### Добавление свойств

Наследование от класса `QnAMaker` демонстрируется в примере ниже. Например, вы можете добавить свойство `MyImportantProperty` в событие `QnAMessage`. Событие `QnAMessage` записывается в журнал при каждом вызове `GetAnswers` службы в QnA.

Изучив процесс добавления пользовательских свойств, мы создадим новое пользовательское событие и свяжем с ним свойства, а затем запустим бота в локальной среде с помощью эмулятора Bot Framework и посмотрим, что передается в Application Insights, с помощью языка запросов Kusto.

1. Создайте новый класс с именем `MyQnAMaker` в пространстве имен `Microsoft.BotBuilderSamples`, унаследовав его от класса `QnAMaker`, и сохраните как `MyQnAMaker.cs`. Для наследования от класса `QnAMaker` необходимо добавить инструкцию `using Microsoft.Bot.Builder.AI.QnA`. Теперь код должен выглядеть следующим образом:

```
using Microsoft.Bot.Builder.AI.QnA;

namespace Microsoft.BotBuilderSamples
{
    public class MyQnAMaker : QnAMaker
    {
        }
}
```

2. Добавьте конструктор класса в `MyQnAMaker`. Обратите внимание, что потребуются два дополнительных оператора `using` для параметров конструкторов `System.Net.Http` и `Microsoft.Bot.Builder`.

```
...
using Microsoft.Bot.Builder.AI.QnA;
using System.Net.Http;
using Microsoft.Bot.Builder;

namespace Microsoft.BotBuilderSamples
{
    public class MyQnAMaker : QnAMaker
    {
        public MyQnAMaker(
            QnAMakerEndpoint endpoint,
            QnAMakerOptions options = null,
            HttpClient httpClient = null,
            IBotTelemetryClient telemetryClient = null,
            bool logPersonalInformation = false)
            : base(endpoint, options, httpClient, telemetryClient, logPersonalInformation)
        {
        }

        }
    }
}
```

3. Добавьте новое свойство в событие `QnAMessage` сразу после конструктора и включите инструкции `System.Collections.Generic`, `System.Threading` и `System.Threading.Tasks`:

```

using Microsoft.Bot.Builder.AI.QnA;
using System.Net.Http;
using Microsoft.Bot.Builder;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;

namespace Microsoft.BotBuilderSamples
{
    public class MyQnAMaker : QnAMaker
    {
        ...

        protected override async Task OnQnaResultsAsync(
            QueryResult[] queryResults,
            Microsoft.Bot.Builder.ITurnContext turnContext,
            Dictionary<string, string> telemetryProperties = null,
            Dictionary<string, double> telemetryMetrics = null,
            CancellationToken cancellationToken = default(CancellationToken))
        {
            var eventData = await FillQnAEventAsync(
                queryResults,
                turnContext,
                telemetryProperties,
                telemetryMetrics,
                cancellationToken)
                .ConfigureAwait(false);

            // Add new property
            eventData.Properties.Add("MyImportantProperty", "myImportantValue");

            // Log QnAMessage event
            TelemetryClient.TrackEvent(
                QnATelemetryConstants.QnaMsgEvent,
                eventData.Properties,
                eventData.Metrics
            );
        }
    }
}

```

4. Включите в бота код использования нового класса, и теперь вместо объекта `qnaMaker` вы будете создавать в `QnABot.cs` объект `MyQnAMaker`:

```

var qnaMaker = new MyQnAMaker(new QnAMakerEndpoint
{
    KnowledgeBaseId = _configuration["QnAKnowledgebaseId"],
    EndpointKey = _configuration["QnAEndpointKey"],
    Host = _configuration["QnAEndpointHostName"]
},
null,
httpClient,
_telemetryClient);

```

**Просмотр данных телеметрии, сохраненных через новое свойство `MyImportantProperty`**

После запуска программы-робота в эмуляторе можно просмотреть результаты в Application Insights, выполнив следующие действия.

1. Вернитесь в браузер, где открыто представление *Журналы (аналитика)*.
2. Введите следующий запрос Kusto и щелкните *Запуск*. Он возвращает количество выполнений для нового свойства:

```
customEvents
| where name == 'QnaMessage'
| extend MyImportantProperty = tostring(customDimensions.MyImportantProperty)
| summarize count() by MyImportantProperty
```

3. Чтобы отобразить подробные сведения вместо счетчика, удалите последнюю строку и повторно выполните запрос:

```
customEvents
| where name == 'QnaMessage'
| extend MyImportantProperty = tostring(customDimensions.MyImportantProperty)
```

### Добавление новых событий с пользовательскими свойствами

Если вам нужно сохранять данные в другое событие, отличное от `QnaMessage`, вы можете создать пользовательское событие с нужными свойствами. Для этого мы добавим в конец класса `MyQnAMaker` следующий код:

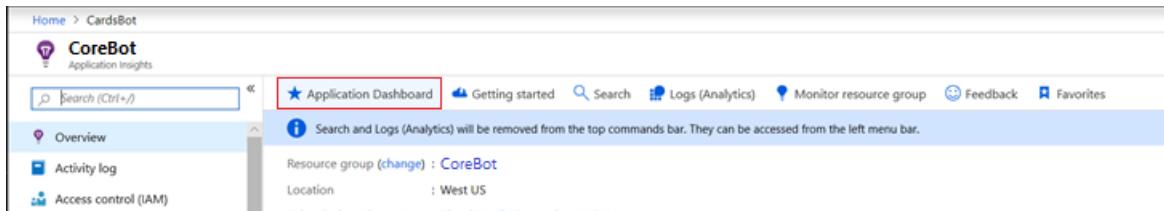
```
public class MyQnAMaker : QnAMaker
{
    ...
    // Create second event.
    var secondEventProperties = new Dictionary<string, string>();

    // Create new property for the second event.
    secondEventProperties.Add(
        "MyImportantProperty2",
        "myImportantValue2");

    // Log secondEventProperties event
    TelemetryClient.TrackEvent(
        "MySecondEvent",
        secondEventProperties);
}
```

## Панель мониторинга Application Insights

Когда вы создаете ресурс Application Insights в Azure, автоматически создается и связанная панель мониторинга. Эту панель мониторинга можно просмотреть, нажав кнопку **Панель мониторинга приложений** в верхней части колонки Application Insights.



Просмотреть данные можно также на портале Azure. Щелкните слева **Панель мониторинга** и выберите нужную панель мониторинга в раскрывающемся списке.

Здесь вы увидите некоторые стандартные сведения о производительности бота и дополнительные запросы, которые вы закрепили на панели мониторинга.

## Дополнительные сведения

- Добавление данных телеметрии в бот
- Что такое Azure Application Insights?
- Поиск в Application Insights
- Создание настраиваемых панелей мониторинга ключевых показателей эффективности с помощью Azure Application Insights

# Анализ данных телеметрии бота

27.03.2021 • 25 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

## Анализ поведения бота

Анализировать поведение бота можно с помощью указанной ниже коллекции запросов. Эта коллекция позволяет создавать пользовательские запросы в [Azure Monitor Log Analytics](#) и (или) панели мониторинга и визуализации [Power BI](#).

## Предварительные требования

Будет полезным иметь хотя бы общее представление о следующих понятиях:

- [запросы Kusto](#);
- создание запросов к журналам Azure Monitor с помощью [Log Analytics](#) на портале Azure;
- основные принципы [запросов к журналам](#) в Azure Monitor.

## Панели мониторинга

Панели мониторинга Azure позволяют с удобством просматривать и предоставлять другим сведения, полученные на основе ваших запросов. Вы можете создавать настраиваемые панели мониторинга, чтобы отслеживать работу ботов, сопоставляя запросы с плитками на панели мониторинга. Дополнительные сведения о панелях мониторинга и сопоставлении запросов с ними см. в статье о [создании панелей мониторинга данных Log Analytics и предоставлении к ним общего доступа](#). Далее в этой статье представлены примеры запросов, которые будут полезны для мониторинга поведения бота.

## Примеры запросов Kusto

### NOTE

Мы рекомендуем создать сводки по разным измерениям, таким как периоды, каналы и языковые стандарты, для всех запросов в этой статье.

### Число пользователей за период

В этом примере создается график по данным о том, сколько уникальных пользователей обращалось к вашему боту за последние 14 дней. Период можно легко изменить, присвоив другие значения переменным `queryStartDate`, `queryEndDate` и `interval`.

### IMPORTANT

Правильное число уникальных пользователей в этом запросе вы получите только в том случае, если они проходили проверку подлинности. Также результаты могут изменяться в зависимости от возможностей канала.

```
// number of users per period
let queryStartDate = ago(14d);
let queryEndDate = now();
let groupByInterval = 1d;
customEvents
| where timestamp > queryStartDate
| where timestamp < queryEndDate
| summarize uc=dcount(user_Id) by bin(timestamp, groupByInterval)
| render timechart
```

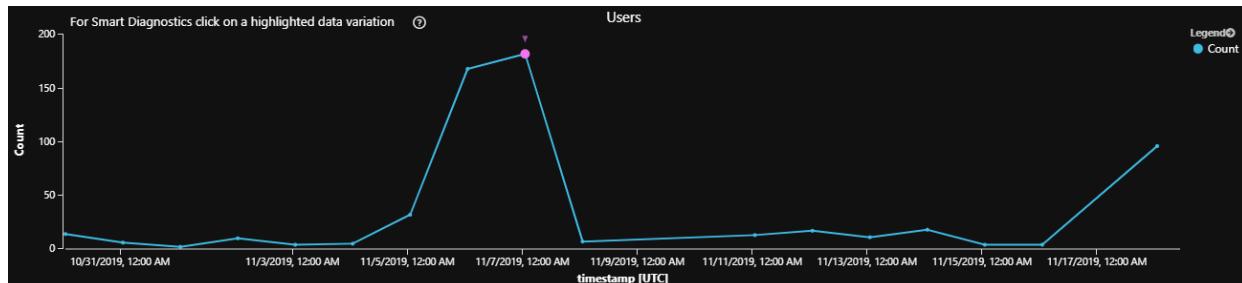
### TIP

Оператор `summarize` в Kusto позволяет создать таблицу со статистическими сведениями по содержимому входной таблицы.

Скалярная функция `Bin` в Kusto, когда она используется в сочетании с `summarize operator`, позволяет сгруппировать результаты запроса по указанному значению. В приведенном выше примере группировка выполнена по дням. Кроме этого, Kusto принимает значение `h` = часы, `m` = минуты, `s` = секунды, `ms` = миллисекунды, `microsecond` = микросекунды.

Оператор `Render` позволяет легко визуализировать диаграммы, например график `timechart`, в котором ось `x` представляет значение `DateTime`, а для оси `y` можно указать любой другой числовой столбец. Он автоматически поддерживает эффективное распределение значений по оси `x`, даже если данные есть не за все периоды. Если оператор `render` не указан, по умолчанию используется `table`.

### Выборка результатов запроса на период для пользователей



### Активность по периодам

В этом примере показано, как оценить объем действий по нужному измерению, например количество бесед, диалогов или сообщений в день за последние 14 дней. Период можно легко изменить, присвоив другие значения переменным `querystartdate`, `queryEndDate` и `interval`. Требуемое измерение определяется `extend` предложением в следующем примере, `metric` может быть задано значение `Instanceid`, `dialogid` или `ActivityId`.

Присвойте параметру `metric` значение того измерения, которое вы намерены отображать.

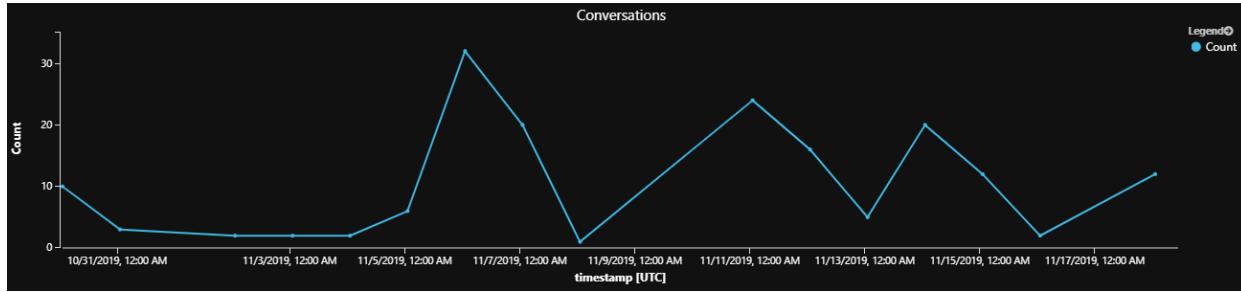
- `Instanceid` позволяет измерить количество **бесед**;
- `DialogId` позволяет измерить количество **диалогов**;
- `ActivityId` позволяет измерить количество **сообщений**.

```
// Measures the number of activity's (conversations, dialogs, messages) per period.
let queryStartDate = ago(14d);
let queryEndDate = now();
let groupByInterval = 1d;
customEvents
| where timestamp > queryStartDate
| where timestamp < queryEndDate
| extend InstanceId = tostring(customDimensions['InstanceId'])
| extend DialogId = tostring(customDimensions['DialogId'])
| extend ActivityId = tostring(customDimensions['activityId'])
| where DialogId != '' and InstanceId != '' and user_Id != ''
| extend metric = InstanceId // DialogId or ActivityId
| summarize Count=dcount(metric) by bin(timestamp, groupByInterval)
| order by Count desc nulls last
| render timechart
```

### TIP

Оператор Kusto [Extend](#) позволяет создавать вычисляемые столбцы и добавлять их к результатирующему набору.

#### Пример результатов запроса "действия в периодах"



#### Активность по пользователям по периодам

В этом примере показано, как подсчитать количество действий на одного пользователя за период. Это удобно, когда мы детализируем запрос *активности по периодам* до следующего уровня: активности по пользователям по периодам. К действиям относятся диалоги, беседы или сообщения. Запрос помогает оценить взаимодействие пользователей с ботом и выявить потенциальные проблемы, например:

- дни с большим количеством действий одного пользователя могут означать атаку или тестирование;
- дни с низким уровнем взаимодействия могут указывать на проблемы с работоспособностью службы.

### TIP

Удалив строку *by user\_Id*, вы получите общий объем действий бота, который можно сводить по времени и диалогам, сообщениям или беседам.

```

// number of users per period per dialogs
let queryStartDate = ago(14d);
let queryEndDate = now();
let interval = 6h;
customEvents
| where timestamp > queryStartDate
| where timestamp < queryEndDate
| extend InstanceId = tostring(customDimensions['InstanceId'])
| extend DialogId = tostring(customDimensions['DialogId'])
| extend ActivityId = tostring(customDimensions['activityId'])
| where DialogId != '' and InstanceId != '' and user_Id != ''
| extend metric = ActivityId // InstanceId // DialogId // or InstanceId for conversation count
| summarize Count=dcount(metric) by user_Id, bin(timestamp, groupByInterval)
| order by Count desc nulls last

```

#### Пример действия для результатов запроса на пользователя в течение периода

USER_ID	TIMESTAMP	COUNT
User-8107ffd2	2019-09-03T00:00:00Z	14
User-75f2cc8f	2019-08-30T00:00:00Z	13
User-75f2cc8d	2019-09-03T00:00:00Z	13
User-3060aada	2019-09-03T00:00:00Z	10

#### Завершение диалогов

Когда вы настроите клиент телеметрии для некоторого диалога, этот диалог и его дочерние элементы будут передавать некоторые данные телеметрии по умолчанию, например *started* и *completed*. Этот пример можно использовать для сравнения количества завершенных (*completed*) и начатых (*started*) диалогов. Если число начатых диалогов превышает число завершенных, значит, некоторые из пользователей не завершают настроенную последовательность. Это может стать отправной точкой для выявления и устранения ошибок в логике диалога. Также это можно использовать для выявления более популярных и редко используемых диалогов.

```

// % Completed Waterfall Dialog: shows completes relative to starts
let queryStartDate = ago(14d);
let queryEndDate = now();
customEvents
| where timestamp > queryStartDate
| where timestamp < queryEndDate
| where name=="WaterfallStart"
| extend DialogId = customDimensions['DialogId']
| extend InstanceId = tostring(customDimensions['InstanceId'])
| join kind=leftouter (
    customEvents
    | where name=="WaterfallComplete"
    | extend InstanceId = tostring(customDimensions['InstanceId']))
    ) on InstanceId
| summarize started=countif(name=='WaterfallStart'), completed=countif(name1=='WaterfallComplete') by
tostring(DialogId)
| where started > 100 // filter for sample
// Show starts vs. completes
| project tostring(DialogId), started, completed
| order by started desc, completed asc nulls last
| render barchart with (kind=unstacked, xcolumn=DialogId, ycolumns=completed, started, ysplit=axes)

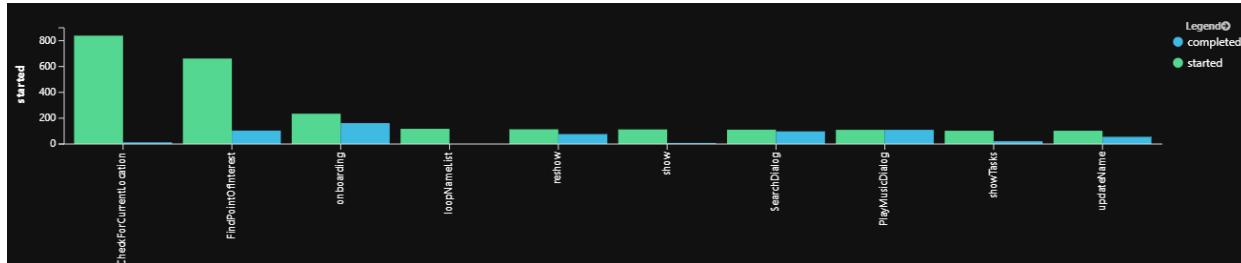
```

## TIP

Оператор `join` в Kusto позволяет объединить строки двух таблиц для формирования новой таблицы путем сопоставления значений указанных столбцов каждой таблицы.

Оператор `project` используется для выбора полей, которые должны отображаться в выходных данных. Как и `extend operator`, который отвечает за добавление нового поля, оператор `project operator` позволяет выбрать поле из набора существующих или добавить новое.

## Пример диалогового окна результатов выполнения запроса



## Незавершение диалога

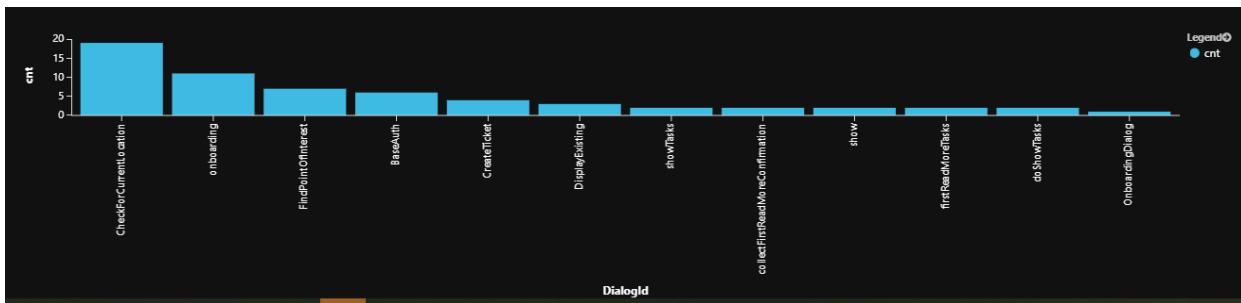
Этот пример можно использовать для подсчета запущенных в течение указанного периода потоков диалогов, которые не были завершены из-за отмены или прекращения действий. С его помощью можно просмотреть незавершенные диалоги и выяснить, была ли отмена активным действием из-за сомнений или связана с потерей внимания либо интереса пользователя.

```
// show incomplete dialogs
let queryStartDate = ago(14d);
let queryEndDate = now();
customEvents
| where timestamp > queryStartDate
| where timestamp < queryEndDate
| where name == "WaterfallStart"
| extend DialogId = customDimensions['DialogId']
| extend instanceId = tostring(customDimensions['InstanceId'])
| join kind=leftanti (
    customEvents
    | where name == "WaterfallComplete"
    | extend instanceId = tostring(customDimensions['InstanceId']))
) on instanceId
| summarize cnt=count() by  tostring(DialogId)
| order by cnt
| render barchart
```

## TIP

Оператор `order` в Kusto (аналогичен `sort operator`) сортирует строки входной таблицы в порядке значений одного или нескольких столбцов. Примечание. Если вам нужно исключить значения NULL из результатов любого запроса, их можно отфильтровать в операторе `WHERE` (например, добавив строку "`and isnotnull(Timestamp)`" или вернуть значения NULL в начале или в конце списка (добавив `nulls first` или `nulls last` в конец инструкции `order`).

## Образец диалогового окна "результаты запроса на невыполнение"



## Детализация последовательности диалога

### Просмотр начала, шагов и завершения для каскадных диалогов в беседе

Этот пример отображает последовательность шагов диалога, сгруппированных по беседе (instanceId).

Это полезно при выявлении шагов, которые приводят к прерыванию диалога.

В поле выполнить этот запрос `DialogId` введите нужное значение вместо <SampleDialogId>

```
// Drill down: Show waterfall start/step/complete for specific dialog
let queryStartDate = ago(14d);
let queryEndDate = now();
let DialogActivity=(dgid:string) {
customEvents
| where timestamp > queryStartDate
| where timestamp < queryEndDate
| extend DialogId = customDimensions['DialogId']
| extend StepName = customDimensions['StepName']
| extend InstanceId = customDimensions['InstanceId']
| where DialogId == dgid
| project timestamp, name, StepName, InstanceId
| order by tostring(InstanceId), timestamp asc
};
// For example see SampleDialogId behavior
DialogActivity("<SampleDialogId>")
```

### TIP

Запрос был написан с использованием [определенной запросом функции](#). Это функция, которая определяется пользователем с помощью оператора `let` и используется в пределах одного запроса. Этот пример можно переписать и без `query-defined function`, вот так:

```
let queryStartDate = ago(14d);
let queryEndDate = now();
customEvents
| where timestamp > queryStartDate
| where timestamp < queryEndDate
| extend DialogId = customDimensions['DialogId']
| extend StepName = customDimensions['StepName']
| extend InstanceId = customDimensions['InstanceId']
| where DialogId == "<SampleDialogId>"
| project timestamp, name, StepName, InstanceId
| order by tostring(InstanceId), timestamp asc
```

### Результаты выполнения примера запроса

TIMESTAMP	NAME	STEPNAME	INSTANCEID
2019-08-23T20:04...	WaterfallStart	null	...79c0f03d8701
2019-08-23T20:04...	WaterfallStep	GetPointOfInterestLocatio nS	...79c0f03d8701

TIMESTAMP	NAME	STEPNAME	INSTANCEID
2019-08-23T20:04...	WaterfallStep	ProcessPointOfInterestSelection	...79c0f03d8701
2019-08-23T20:04...	WaterfallStep	GetRoutesToDestination	...79c0f03d8701
2019-08-23T20:05...	WaterfallStep	ResponseToStartRoutePrompt	...79c0f03d8701
2019-08-23T20:05...	Ватерфаллкомплект <sup>1</sup>	null	...79c0f03d8701
2019-08-28T23:35...	WaterfallStart	null	...6ac8b3211b99
2019-08-28T23:35...	Ватерфаллстеп <sup>2</sup>	GetPointOfInterestLocations	...6ac8b3211b99
2019-08-28T19:41...	WaterfallStart	null	...8137d76a5cbb
2019-08-28T19:41...	Ватерфаллстеп <sup>2</sup>	GetPointOfInterestLocations	...8137d76a5cbb
2019-08-28T19:41...	WaterfallStart	null	...8137d76a5cbb

1 завершено

2 Отброшено

*Интерпретация полученных данных. Похоже, что пользователи часто прерывают беседу на шаге GetPointOfInterestLocations.*

#### NOTE

Каскадные диалоги выполняют последовательность (начало, несколько шагов, завершение). Если для последовательности отображается начало, но нет завершения, значит, диалог был прерван из-за отмены пользователем или прекращения действий. В этих подробных результатах анализа вы заметите такое поведение (сравнивая количество завершенных и прерванных шагов).

#### Агрегированные результаты для шагов каскадного диалога (начало, шаг, завершение, отмена)

Этот пример демонстрирует статистические итоги: общее количество запусков последовательности диалога, общее количество каскадных шагов, количество успешно выполненных и количество отмен.

Разность значения WaterfallStart и суммы значений WaterfallComplete и WaterfallCancel обозначает общее число отмененных диалогов.

```

// Drill down: Aggregate view of waterfall start/step/complete/cancel steps totals for specific dialog
let queryStartDate = ago(14d);
let queryEndDate = now();
let DialogSteps=(dlgId:string) {
customEvents
| where timestamp > queryStartDate
| where timestamp < queryEndDate
| extend DialogId = customDimensions['DialogId']
| where DialogId == dlgId
| project name
| summarize count() by name
};

// For example see SampleDialogId behavior
DialogSteps("<SampleDialogId>")

```

Образец каскада — результаты статистических запросов

NAME	COUNT
WaterfallStart	21
WaterfallStep	47
WaterfallComplete	11
WaterfallCancel	1

*Интерпретация: 21 вызовов последовательности диалоговых окон, только 11 завершено, 9 отменено, а другая была отменена пользователем.*

### Средняя продолжительность диалога

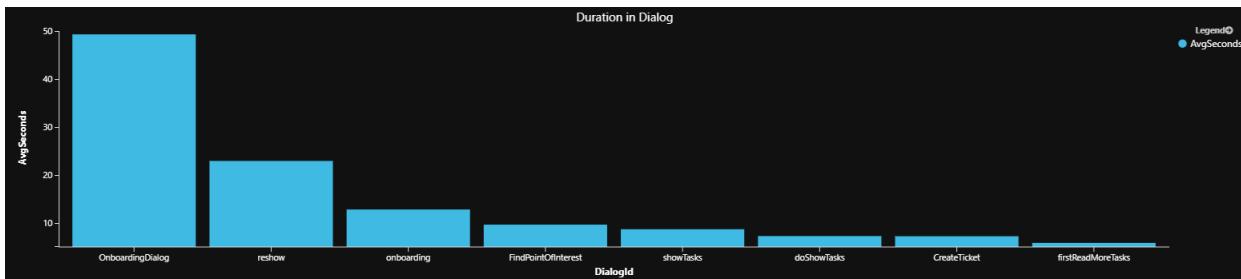
В этом примере оценивается среднее время, затрачиваемое пользователями на определенный диалог. Длительное время нахождения в диалоге может означать, что можно его упростить.

```

// Average dialog duration
let queryStartDate = ago(14d);
let queryEndDate = now();
customEvents
| where timestamp > queryStartDate
| where timestamp < queryEndDate
| where name=="WaterfallStart"
| extend DialogId = customDimensions['DialogId']
| extend instanceId = tostring(customDimensions['InstanceId'])
| join kind=leftouter (customEvents | where name=="WaterfallCancel" | extend instanceId =
tostring(customDimensions['InstanceId'])) on instanceId
| join kind=leftouter (customEvents | where name=="WaterfallComplete" | extend instanceId =
tostring(customDimensions['InstanceId'])) on instanceId
| extend duration = case(not(isnull(timestamp1)), timestamp1 - timestamp,
not(isnull(timestamp2)), timestamp2 - timestamp, 0s) // Abandoned are not counted. Alternate: now()-timestamp)
| extend seconds = round(duration / 1s)
| summarize AvgSeconds=avg(seconds) by tostring(DialogId)
| order by AvgSeconds desc nulls last
| render barchart with (title="Duration in Dialog")

```

Выборка результатов запроса средней длительности



### Среднее число шагов в диалоге

В этом примере отображается "длина" для каждого выполненного диалога, которая определяется по среднему, минимальному и максимальному количеству шагов, а также стандартному отклонению. Это помогает анализировать качество диалогов. Пример:

- Диалоги с большим количеством шагов следует проверить на возможность упрощения.
- Большая разница между минимальным, максимальным и (или) средним числом шагов может означать, что пользователи путаются или теряются при выполнении задач. Возможно, для таких задач стоит сократить длину путей или снизить сложность диалога.
- Диалоги с большим стандартным отклонением могут содержать слишком сложные пути или ошибки в логике (частая отмена или прекращение).
- Малое число шагов в диалоге может означать, что они не завершены. Сравнение долей завершенных и прекращенных диалогов поможет понять, так ли это.

```
// min/max/std/avg steps per dialog
let queryStartDate = ago(14d);
let queryEndDate = now();
customEvents
| where timestamp > queryStartDate
| where timestamp < queryEndDate
| extend DialogId = tostring(customDimensions['DialogId'])
| extend StepName = tostring(customDimensions['StepName'])
| extend InstanceId = tostring(customDimensions['InstanceId'])
| where name == "WaterfallStart" or name == "WaterfallStep" or name == "WaterfallComplete"
| order by InstanceId, timestamp asc
| project timestamp, DialogId, name, InstanceId, StepName
| summarize cnt=count() by InstanceId, DialogId
| summarize avg=avg(cnt), minsteps=min(cnt), maxsteps=max(cnt), std=stdev(cnt) by DialogId
| extend avgsteps = round(avg, 1)
| extend avgshortbysteps=maxsteps-avgsteps
| extend avgshortbypercent=round((1.0 - avgsteps/maxsteps)*100.0, 1)
| project DialogId, avgsteps, minsteps, maxsteps, std, avgshortbysteps, avgshortbypercent
| order by std desc nulls last
```

### Выборка среднего числа шагов запроса

ID ДИАЛОГА	СРЕДНЕЕ ЧИСЛО ШАГОВ	МИНИМАЛЬНОЕ ЧИСЛО ШАГОВ	МАКСИМАЛЬНОЕ ЧИСЛО ШАГОВ	STD	СРЕДНЕЕ СОКРАЩЕНИЕ ПО ШАГАМ	СРЕДНЯЯ СОКРАЩЕНИЕ В ПРОЦЕНТАХ
FindArticlesDialog	6.2	2	7	2,04	0,8	11,4 %
CreateTicket	4.3	2	5	1.5	0,7	14 %
CheckForCurrentLocation	3.9	2	5	1,41 %	1,1	22 %
BaseAuth	3.3	2	4	1,03 %	0,7	17,5 %

ИД ДИАЛОГА	СРЕДНЕЕ ЧИСЛО ШАГОВ	МИНИМАЛЬНОЕ ЧИСЛО ШАГОВ	МАКСИМАЛЬНОЕ ЧИСЛО ШАГОВ	STD	СРЕДНЕЕ СОКРАЩЕНИЕ ПО ШАГАМ	СРЕДНЯЯ СОКРАЩЕНИЕ В ПРОЦЕНТАХ
Подключение	2.7	2	4	0,94	1,3	32,5 %

Интерпретация полученных данных. Например, у FindArticlesDialog большая разница между минимальным и максимальным числом шагов, а значит, его нужно проверить и, возможно, переработать и (или) оптимизировать.

### Активность канала по метрике действия

В этом примере оценивается количество действий, которые бот получает от каждого канала за определенный период. Для этого он подсчитывает любую из следующих метрик: входящие сообщений, пользователи, беседы или диалоги. Это может быть полезно для анализа работоспособности служб и (или) оценки популярности каналов.

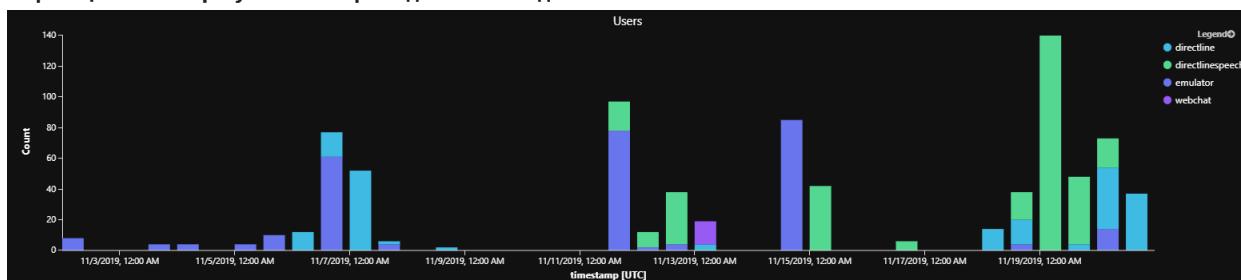
```
// number of metric: messages, users, conversations, dialogs by channel
let queryStartDate = ago(14d);
let queryEndDate = now();
let groupByInterval = 1d;
customEvents
| where timestamp > queryStartDate
| where timestamp < queryEndDate
| extend InstanceId = tostring(customDimensions['InstanceId'])
| extend DialogId = tostring(customDimensions['DialogId'])
| extend ActivityId = tostring(customDimensions['activityId'])
| extend ChannelId = tostring(customDimensions['channelId'])
| where DialogId != '' and InstanceId != '' and user_Id != ''
| extend metric = user_Id // InstanceId or ActivityId or user_Id
| summarize Count=count(metric) by ChannelId, bin(timestamp, groupByInterval)
| order by Count desc nulls last
| render barchart with (title="Users", kind=stacked) // or Incoming Messages or Conversations or Users
```

#### TIP

Вы можете попробовать следующие варианты:

- Выполните запрос без сегментирования меток времени: `bin(timestamp, groupByInterval)`.
- Также сравните `dcount` (для отдельных пользователей) и `count` (для всех действий с событиями пользователя). Этот запрос удобен и для повторных пользователей.

### Образец канала — результаты запроса действий по действиям



Интерпретация полученных данных. Ранее широко использовалось тестирование в эмуляторе, но с момента публикации службы в канале DirectLineSpeech он стал самым популярным.

### Общее количество целей по популярности

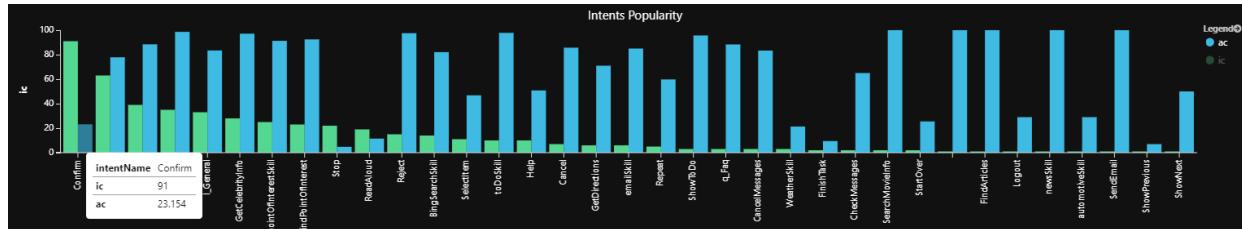
Этот пример применяется к ботам с поддержкой LUIS. Он отображает сводку по всем [намерениям](#) с

сортировкой по популярности и оценками уверенности в определении этих намерений.

- На практике это представление лучше просматривать отдельно для каждой метрики.
- Более популярные пути намерений можно оптимизировать для улучшения взаимодействия с пользователем.
- Низкие средние показатели могут означать плохое определение намерений или низкую востребованность.

```
// show total intents
let queryStartDate = ago(14d);
let queryEndDate = now();
customEvents
| where timestamp > queryStartDate
| where timestamp < queryEndDate
| where name startswith "LuisResult"
| extend intentName = tostring(customDimensions['intent'])
| extend intentScore = todouble(customDimensions['intentScore'])
| summarize ic=count(), ac=avg(intentScore)*100 by intentName
| project intentName, ic, ac
| order by ic desc nulls last
| render barchart with (kind=unstacked, xcolumn=intentName, ycolumns=ic,ac, title="Intents Popularity")
```

#### Образцы результатов запроса по популярности



Интерпретация полученных данных. Например, самое популярное намерение (подтверждение) обнаруживается со средней точностью лишь 23 %.

#### TIP

Линейчатые диаграммы (barchart) — один из нескольких десятков вариантов, доступных для запросов Kusto. Вы также можете использовать anomalychart (диаграмма аномалий), areachart (диаграмма зона), columnchart (диаграмма столбцов), linechart (график), scatterchart (точечная диаграмма). Дополнительные сведения см. в документации по [оператору render](#).

## Схема инструментирования аналитики бота

В следующих таблицах показаны наиболее распространенные поля, в которые бот может записывать данные телеметрии.

### Общая огибающая

Типичные поля анализа журналов при инструментировании Application Insights.

ПОЛЕ	ОПИСАНИЕ	ПРИМЕРЫ ЗНАЧЕНИЙ
------	----------	------------------

ПОЛЕ	ОПИСАНИЕ	ПРИМЕРЫ ЗНАЧЕНИЙ
name	тип сообщений;	BotMessageSend, BotMessageReceived, LuisResult, WaterfallStep, WaterfallStart, SkillWebSocketProcessRequestLatency, SkillWebSocketOpenCloseLatency, WaterfallComplete, QnaMessage, WaterfallCancel, SkillWebSocketTurnLatency, AuthPromptValidatorAsyncFailure
customDimensions	Пакет средств разработки аналитики бота	activityId = <id>, ActivityType = Message, channelId = Emulator, fromId = <id>, FromName = пользователь, locale = en-US, RecipientId = <id>, RecipientName = Bot, Text = найти кафе
TIMESTAMP	Время события	2019-09-05T18:32:45.287082Z
instance_Id	Идентификатор беседы	f7b2c416-a680-4b2c-b4cc- 79c0f03d8711
operation_Id	ИД шага	084b2856947e3844a5a18a8476d99a aa
user_Id	Уникальный идентификатор пользователя в канале	emulator7c259c8e-2f47...
client_IP	IP-адрес клиента	127.0.0.1 (может отсутствовать в связи с блокировкой конфиденциальности)
client_City	Город клиента	Редмонд (если обнаружен, но может отсутствовать)

### Пользовательские измерения

Большинство данных о действиях для конкретного бота хранится в поле *customDimensions*.

ПОЛЕ	ОПИСАНИЕ	ПРИМЕРЫ ЗНАЧЕНИЙ
activityId	Идентификатор сообщения	<id>: 8da6d750-d00b-11e9-80e0- c14234b3bc2a
activityType	Тип сообщения	message, conversationUpdate, event, invoke
channelId	Идентификатор канала	emulator, directline, msteams, webchat
fromId	Идентификатор источника	<id>
fromName	Имя пользователя, полученное от клиента	John Bonham, Keith Moon, Steve Smith, Steve Gadd

ПОЛЕ	ОПИСАНИЕ	ПРИМЕРЫ ЗНАЧЕНИЙ
локаль	Исходный язык клиента	en-us, zh-cn, en-GB, de-de, zh-CN
recipientId	Идентификатор получателя	<id>
recipientName	Имя получателя	John Bonham, Keith Moon, Steve Smith, Steve Gadd
text	Текст сообщения	find a coffee shop

### Пользовательские измерения: LUIS

При инструментировании LUIS данные сохраняются в следующих полях пользовательских измерений.

ПОЛЕ	ОПИСАНИЕ	ПРИМЕРЫ ЗНАЧЕНИЙ
intent	Обнаружено намерение LUIS	pointOfInterestSkill
intentScore	Оценка распознавания LUIS	0,98
Сущности	Обнаруженные сущности LUIS	FoodOfGrocery = [["coffee"]], KEYWORD= ["coffee shop"]
Вопрос	Обнаруженный вопрос LUIS	find a coffee shop
sentimentLabel	Обнаруженная тональность LUIS	Позитивная

### Пользовательские измерения: QnA Maker

При инструментировании QnAMaker данные сохраняются в следующих полях пользовательских измерений.

#### TIP

Чтобы включить ведение журнала персональных данных, таких как вопросы и ответы, параметру *журнала личных сведений* должно быть присвоено значение true в конструкторе класса *QnA Maker*.

ПОЛЕ	ОПИСАНИЕ	ПРИМЕРЫ ЗНАЧЕНИЙ
question	Обнаруженный вопрос QnA	what can you do?
Ответ	Ответ QnA	You have questions, I may have answers.
articleFound	QnA	Да
questionId	Идентификатор вопроса QnA	488
knowledgeBaseld	Идентификатор базы знаний QnA	2a4936f3-b2c8-44ff-b21f-67bc413b9727

ПОЛЕ	ОПИСАНИЕ	ПРИМЕРЫ ЗНАЧЕНИЙ
matchedQuestion	Массив подходящих вопросов	[ "Can you explain to me what your role is?", "Can you tell me a bit about yourself?", "Can you tell me about you?", "could you help me", "hmmm so what can you do?", "how can you help me", "How can you help me?", "How can you help?", "so how can i use you in my projects?", "Talk to me about your capability", "What are you capable of?", ... ]

## См. также:

- Инструкции по написанию запросов к журналам Azure Monitor см. в [этой статье](#).
- Прочтите статью о [визуализации данных из Azure Monitor](#).
- Узнайте, как [добавить телеметрию в бота](#).
- Ознакомьтесь с дополнительными сведениями о [запросах по журналам в Azure Monitor](#).
- Полный список [событий Application Insightsной](#) платформы Bot
- Узнайте, как [создавать панели мониторинга данных Log Analytics и предоставлять к ним общий доступ](#).

# Использование канала Direct Line Speech в боте

27.03.2021 • 5 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Direct Line Speech использует новую возможность потоковой передачи на основе WebSocket, которую предоставляет Bot Framework, чтобы организовать обмен сообщениями между каналом Direct Line Speech и ботом. Настроив канал Direct Line Speech на портале Azure, следует обновить бота для прослушивания и приема этих подключений WebSocket. Здесь мы приводим инструкции, как это сделать.

## Шаг 1. Обновление пакета SDK до последней версии

Для использования Direct Line Speech требуется последняя версия пакета SDK для Bot Builder.

## Шаг 2. Обновите код бота для .NET Core, если в нем вместо BotController используются AddBot и UseBotFramework.

Если вы создавали бота с помощью пакета SDK Bot Builder v4 в версии, предшествующей 4.3.2, в нем, скорее всего, нет BotController, а вместо него в файле Startup.cs применяются методы AddBot() и UseBotFramework() для предоставления конечной точки POST, в которой бот получает сообщения. Чтобы предоставить новую конечную точку потоковой передачи, следует добавить BotController и удалить методы AddBot() и UseBotFramework(). Здесь представлены инструкции по внесению таких изменений. Если вы уже внесли эти изменения, перейдите к следующему шагу.

Добавьте в проект бота новый контроллер MVC, добавив файл с именем BotController.cs. Добавьте в этот файл код контроллера:

```
[Route("api/messages")]

[ApiController]

public class BotController : ControllerBase
{
    private readonly IBotFrameworkHttpAdapter _adapter;
    private readonly IBot _bot;
    public BotController(IBotFrameworkHttpAdapter adapter, IBot bot)
    {
        _adapter = adapter;

        _bot = bot;
    }

    [HttpPost,HttpGet]
    public async Task ProcessMessageAsync()
    {
        await _adapter.ProcessAsync(Request, Response, _bot);
    }
}
```

В файле Startup.cs найдите метод Configure. Удалите строку `UseBotFramework()` и убедитесь, что в нем есть следующие строки для `UseWebSockets`:

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    ...
    app.UseDefaultFiles();
    app.UseStaticFiles();
    app.UseWebSockets();
    app.UseMvc();
    ...
}
```

В этом же файле Startup.cs найдите метод ConfigureServices. Удалите строку `AddBot()` и убедитесь, что в нем есть строки для добавления `IBot` и `BotFrameworkHttpAdapter`:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
    services.AddSingleton<ICredentialProvider, ConfigurationCredentialProvider>();
    services.AddSingleton<IChannelProvider, ConfigurationChannelProvider>();

    // Create the Bot Framework Adapter.
    services.AddSingleton<IBotFrameworkHttpAdapter, BotFrameworkHttpAdapter>();

    // Create the bot as a transient. In this case the ASP Controller is expecting an IBot.
    services.AddTransient<IBot, EchoBot>();
}
```

Остальная часть кода бота остается неизменной.

## Шаг 3. Убедитесь, что включены WebSocket

При создании нового бота на портале Azure с помощью одного из шаблонов, например EchoBot, полученный бот будет содержать контроллер MVC для ASP.NET, который предоставляет одну конечную точку GET и POST и поддерживает WebSocket. В этих инструкциях описано, как добавить в бот эти элементы, если вы его обновляете или создавали не из шаблона бота.

Откройте `BotController.cs` в папке `Controllers` для своего решения.

Найдите в классе метод `PostAsync` и измените для него маркер `[HttpPost]` на `[HttpPost,HttpGet]`:

```
[HttpPost, HttpGet]
public async Task PostAsync()
{
    await _adapter.ProcessAsync(Request, Response, _bot);
}
```

Сохраните и закройте `BotController.cs`.

Откройте файл `Startup.cs` из корневого каталога решения.

В файле `Startup.cs` перейдите в нижнюю часть метода `Configure`. Перед вызовом `app.UseMvc()` добавьте вызов `app.UseWebSockets()`. Порядок вызовов этих инструкций `use` имеет значение. Теперь конец метода должен выглядеть примерно так:

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    ...
    app.UseDefaultFiles();
    app.UseStaticFiles();
    app.UseWebSockets();
    app.UseMvc();
    ...
}
```

Остальная часть кода бота остается неизменной.

## Шаг 4. При необходимости заполните поле Speak (Сказать) в разделе действий, чтобы изменить проговариваемые для пользователя сообщения.

По умолчанию проговариваются все сообщения, отправляемые пользователю через Direct Line Speech.

По желанию вы можете настроить режим озвучки сообщений, установив флажок Speak (Сказать) для любого действия, отправляемого из бота.

```
public IActivity Speak(string message)
{
    var activity = MessageFactory.Text(message);
    string body = @"<speak version='1.0' xmlns='https://www.w3.org/2001/10/synthesis' xml:lang='en-US'>

        <voice name='Microsoft Server Speech Text to Speech Voice (en-US, JessaRUS)'>" +
        $"{message}" + "</voice></speak>";

    activity.Speak = body;
    return activity;
}
```

Следующий фрагмент кода демонстрирует, как использовать предыдущую функцию Speak.

```
protected override async Task OnMessageActivityAsync(ITurnContext< IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    await turnContext.SendActivityAsync(Speak($"Echo: {turnContext.Activity.Text}"), cancellationToken);
}
```

## Дополнительные сведения

- Полный пример создания и использования бота с поддержкой голосовых функций см. в [этом руководстве](#).
- Дополнительные сведения о работе с действиями см. в статьях[принципы работы программы-роботы](#) и[Отправка и получение текстовых сообщений](#).

# Как выполнять модульное тестирование ботов

27.03.2021 • 30 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье описаны следующие задачи:

- создание модульных тестов для ботов;
- использование оператора assert для сопоставления действий, которые возвращает шаг диалога, с ожидаемыми значениями;
- использование оператора assert для проверки результатов, которые возвращает диалог;
- создание разных типов тестов, управляемых данными;
- создание макетов объектов для разных зависимостей диалогов (распознавателей LUIS и т. д.).

## Предварительные требования

- [C#](#)
- [JavaScript](#)

Пример [CoreBot Tests](#) ссылается на пакет [Microsoft.Bot.Builder.Testing](#), [XUnit](#) и [Moq](#) для создания модульных тестов.

## Тестирование диалогов

В примере CoreBot выполняется модульное тестирование диалогов с помощью класса [DialogTestClient](#) — средства изолированного тестирования за пределами бота, при использовании которого не нужно развертывать код в веб-службе.

С помощью этого класса можно создавать модульные тесты, которые проверяют ответы в диалогах на основе смены шагов. Модульные тесты с использованием класса [DialogTestClient](#) должны поддерживать другие диалоги, созданные с помощью библиотеки диалогов Bot Builder.

В следующем примере показаны тесты с использованием [DialogTestClient](#):

- [C#](#)
- [JavaScript](#)

```

var sut = new BookingDialog();
var testClient = new DialogTestClient(Channels.Msteams, sut);

var reply = await testClient.SendActivityAsync<IMessageActivity>("hi");
Assert.Equal("Where would you like to travel to?", reply.Text);

reply = await testClient.SendActivityAsync<IMessageActivity>("Seattle");
Assert.Equal("Where are you traveling from?", reply.Text);

reply = await testClient.SendActivityAsync<IMessageActivity>("New York");
Assert.Equal("When would you like to travel?", reply.Text);

reply = await testClient.SendActivityAsync<IMessageActivity>("tomorrow");
Assert.Equal("OK, I will book a flight from Seattle to New York for tomorrow, Is this Correct?", reply.Text);

reply = await testClient.SendActivityAsync<IMessageActivity>("yes");
Assert.Equal("Sure thing, wait while I finalize your reservation...", reply.Text);

reply = testClient.GetNextReply<IMessageActivity>();
Assert.Equal("All set, I have booked your flight to Seattle for tomorrow", reply.Text);

```

Класс `DialogTestClient` определяется в пространстве имен `Microsoft.Bot.Builder.Testing`. Он включен в пакет NuGet [Microsoft.Bot.Builder.Testing](#).

## DialogTestClient

Первый параметр `DialogTestClient` — это целевой канал. Это позволяет протестировать другую логику отрисовки на основе целевого канала для программы-робота (команды, временной резерв и т. д.). Если вы не уверены, какой целевой канал выбрать, вы можете использовать идентификаторы канала `Emulator` или `Test`. Но помните, что некоторые компоненты могут вести себя по-разному в зависимости от текущего канала, например `ConfirmPrompt` отображает варианты "Да/Нет" по-разному для каналов `Test` и `Emulator`. Этот параметр также можно использовать для проверки условной логики визуализации в диалоге на основе идентификатора канала.

Второй параметр — это экземпляр тестируемого диалога. (Примечание. Сокращение `sut`, используемое в фрагментах кода, означает "тестируемая система".)

Конструктор `DialogTestClient` предоставляет дополнительные параметры для дополнительной настройки поведения клиента или передачи параметров в тестируемый диалог, если это необходимо. Вы можете передать данные инициализации для диалога, добавить настраиваемое ПО промежуточного слоя, а также использовать `TestAdapter` и экземпляр `ConversationState`.

## Отправка и получение сообщений

- [C#](#)
- [JavaScript](#)

Метод `SendActivityAsync<IActivity>` отправляет текстовый фрагмент или `IActivity` в диалог и возвращает первое полученное сообщение. Параметр `<t>` используется для возврата строго типизированного экземпляра ответа, чтобы вы могли утвердить его без приведения.

```

var reply = await testClient.SendActivityAsync<IMessageActivity>("hi");
Assert.Equal("Where would you like to travel to?", reply.Text);

```

В некоторых сценариях бот может отправить несколько сообщений в ответ на одно действие. В таких случаях `DialogTestClient` будет помещать ответы в очередь и использовать метод `GetNextReply<IActivity>` для извлечения следующего сообщения из очереди.

```
reply = testClient.GetNextReply<IMessageActivity>();
Assert.Equal("All set, I have booked your flight to Seattle for tomorrow", reply.Text);
```

Если в очереди ответов отсутствуют сообщения, `GetNextReply<IActivity>` возвращает значение NULL.

## Утверждение действий

- [C#](#)
- [JavaScript](#)

Код в примере CoreBot утверждает только свойство `Text` полученных действий. Для более сложных ботов может потребоваться утверждать другие свойства, например `Speak`, `InputHint`, `ChannelData` и т. д.

```
Assert.Equal("Sure thing, wait while I finalize your reservation...", reply.Text);
Assert.Equal("One moment please...", reply.Speak);
Assert.Equal(InputHints.IgnoringInput, reply.InputHint);
```

Это можно сделать, проверив каждое свойство, как показано выше. Вы можете написать собственные вспомогательные служебные программы для утверждения действий или использовать другие платформы, например [FluentAssertions](#), для написания пользовательских утверждений и упрощения кода теста.

## Передача параметров в диалоги

Конструктор `DialogTestClient` использует `initialDialogOptions` для передачи параметров в диалог. Например, `MainDialog` в этом примере инициализирует объект `BookingDetails` из результатов LUIS с сущностями, которые он разрешает из речевого фрагмента пользователя, а затем передает этот объект для вызова `BookingDialog`.

Это можно реализовать в тесте следующим образом:

- [C#](#)
- [JavaScript](#)

```
var inputDialogParams = new BookingDetails()
{
    Destination = "Seattle",
    TravelDate = $"{DateTime.UtcNow.AddDays(1):yyyy-MM-dd}"
};

var sut = new BookingDialog();
var testClient = new DialogTestClient(Channels.Msteams, sut, inputDialogParams);
```

`BookingDialog` получает этот параметр и обращается к нему в teste так же, как при вызове из `MainDialog`.

```
private async Task<DialogTurnResult> DestinationStepAsync(WaterfallStepContext stepContext,
CancellationToken cancellationToken)
{
    var bookingDetails = (BookingDetails)stepContext.Options;
    ...
}
```

## Результаты утверждения шага диалога

- [C#](#)
- [JavaScript](#)

Некоторые диалоги, например `BookingDialog` или `DateResolverDialog`, возвращают значение в вызывающий диалог. Объект `DialogTestClient` предоставляет свойство `DialogTurnResult` для анализа и утверждения результатов, возвращаемых диалогом.

Пример:

```
var sut = new BookingDialog();
var testClient = new DialogTestClient(Channels.Msteams, sut);

var reply = await testClient.SendActivityAsync<IMessageActivity>("hi");
Assert.Equal("Where would you like to travel to?", reply.Text);

...

var bookingResults = (BookingDetails)testClient.DialogTurnResult.Result;
Assert.Equal("New York", bookingResults?.Origin);
Assert.Equal("Seattle", bookingResults?.Destination);
Assert.Equal("2019-06-21", bookingResults?.TravelDate);
```

Свойство `DialogTurnResult` также можно использовать для проверки и утверждения промежуточных результатов, возвращаемых шагами в каскаде.

### Анализ результатов теста

Иногда бывает нужно прочитать расшифровку модульного теста, чтобы проанализировать выполнение теста без его отладки.

- C#
- JavaScript

Пакет `Microsoft.Bot.Builder.Testing` включает `XUnitDialogTestLogger` для записи в консоль сообщений, отправляемых и получаемых диалогом.

Чтобы использовать это ПО промежуточного слоя, тест должен использовать конструктор, который получает объект `ITestOutputHelper`, предоставляемый средством запуска тестов XUnit, и создать `XUnitDialogTestLogger` для передачи в `DialogTestClient` с использованием параметра `middlewares`.

```
public class BookingDialogTests
{
    private readonly IMiddleware[] _middlewares;

    public BookingDialogTests(ITestOutputHelper output)
        : base(output)
    {
        _middlewares = new[] { new XUnitDialogTestLogger(output) };
    }

    [Fact]
    public async Task SomeBookingDialogTest()
    {
        // Arrange
        var sut = new BookingDialog();
        var testClient = new DialogTestClient(Channels.Msteams, sut, middlewares: _middlewares);

        ...
    }
}
```

Ниже показано, как после настройки `XUnitDialogTestLogger` выводит записанные данные:

Test Name: CoreBot.Tests.Dialogs.BookingDialogTests.DialogFlowUseCases(testData: TestDataObject { TestObject = {"Name": "Full flow", "InitialBookingDetails": "..."} })

Test Outcome:  Passed

#### Standard Output

Test Case: Full flow

User: hi  
-> ts: 01:34:10

Bot: Text=Where would you like to travel to?  
Speak=Where would you like to travel to?  
InputHint=expectingInput  
-> ts: 01:34:10 elapsed: 0 ms

User: Seattle  
-> ts: 01:34:10

Bot: Text=Where are you traveling from?  
Speak=Where are you traveling from?  
InputHint=expectingInput  
-> ts: 01:34:10 elapsed: 2 ms

User: New York  
-> ts: 01:34:10

Bot: Text=When would you like to travel?  
Speak=When would you like to travel?  
InputHint=expectingInput  
-> ts: 01:34:10 elapsed: 0 ms

User: tomorrow  
-> ts: 01:34:10

Bot: Text=Please confirm, I have you traveling to: Seattle from: New York on: 2019-06-22. Is this correct? (1) Yes or (2) No  
Speak=Please confirm, I have you traveling to: Seattle from: New York on: 2019-06-22. Is this correct?  
InputHint=expectingInput  
-> ts: 01:34:10 elapsed: 3 ms

User: yes  
-> ts: 01:34:10

Подробные сведения о записи и отправке выходных данных теста в консоль при использовании xUnit см. [в документации по xUnit](#).

Эти выходные данные также регистрируются на сервере сборки во время сборок с непрерывной интеграцией. Они помогают анализировать сбои сборки.

## Тесты, управляемые данными

В большинстве случаев логика диалога не меняется, и разные пути выполнения в диалоге основаны на речевых фрагментах пользователя. Чтобы не писать модульные тесты для каждого варианта в диалоге, проще использовать управляемые данными (параметризованные) тесты.

Так, пример теста в разделе с общими сведениями показывает, как проверить один поток выполнения. Но если пользователь не выполнит подтверждение, используется другая дата или существуют другие условия?

Управляемые данными тесты позволяют протестировать все эти изменения без необходимости переписывать тесты.

- [C#](#)
- [JavaScript](#)

В примере CoreBot мы используем тесты [Theory](#) из XUnit для параметризации тестов.

### Тесты с использованием `InlineData`

Следующий тест проверяет, что диалог отменяется, когда пользователь вводит cancel (Отмена).

```
[Fact]
public async Task ShouldBeAbleToCancel()
{
    var sut = new TestCancelAndHelpDialog();
    var testClient = new DialogTestClient(Channels.Test, sut);

    var reply = await testClient.SendActivityAsync<IMessageActivity>("Hi");
    Assert.Equal("Hi there", reply.Text);
    Assert.Equal(DialogTurnStatus.Waiting, testClient.DialogTurnResult.Status);

    reply = await testClient.SendActivityAsync<IMessageActivity>("cancel");
    Assert.Equal(" Cancelling...", reply.Text);
}
```

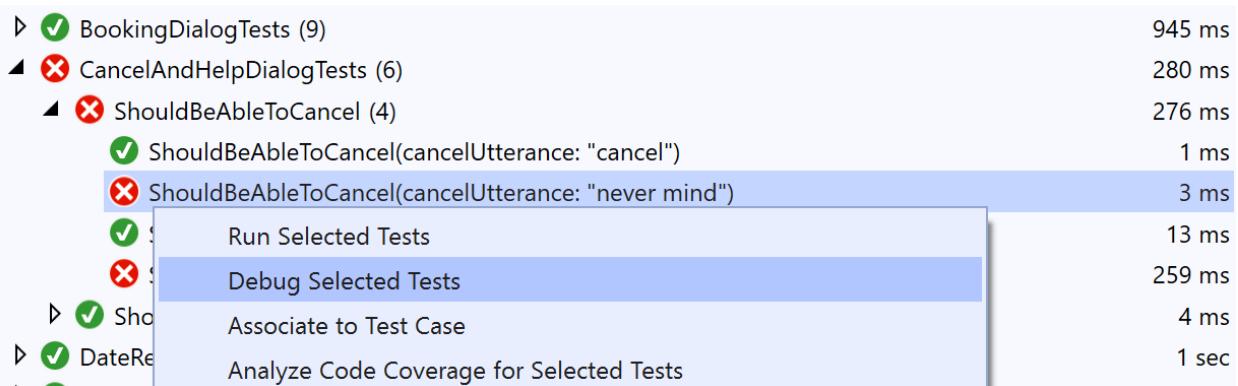
Но для отмены диалога пользователь может ввести `quit` (Выйти), `never mind` (Не важно) или `stop it` (Остановить). Чтобы не писать новый тестовый случай для каждого возможного слова, создайте один метод теста `Theory`, который принимает параметры через список значений `InlineData` и определяет параметры для каждого тестового случая:

```
[Theory]
[InlineData("cancel")]
[InlineData("quit")]
[InlineData("never mind")]
[InlineData("stop it")]
public async Task ShouldBeAbleToCancel(string cancelUtterance)
{
    var sut = new TestCancelAndHelpDialog();
    var testClient = new DialogTestClient(Channels.Test, sut, middlewares: _middlewares);

    var reply = await testClient.SendActivityAsync<IMessageActivity>("Hi");
    Assert.Equal("Hi there", reply.Text);
    Assert.Equal(DialogTurnStatus.Waiting, testClient.DialogTurnResult.Status);

    reply = await testClient.SendActivityAsync<IMessageActivity>(cancelUtterance);
    Assert.Equal(" Cancelling...", reply.Text);
}
```

Новый тест будет выполнен четыре раза с разными параметрами, и каждый вариант будет отображен как дочерний элемент в наборе тестов `ShouldBeAbleToCancel` в обозревателе тестов Visual Studio. Если один из тестов завершится с ошибкой, как показано ниже, не нужно перезапускать весь набор тестов. Просто щелкните правой кнопкой мыши и выполните отладку скрипта, который завершился сбоем.



## Тестирование с использованием `MemberData` и сложных типов

`InlineData` удобно использовать для небольших тестов, управляемых данными, которые получают параметры со значениями простых типов (строка, целое число и т. д.).

`BookingDialog` получает объект `BookingDetails` и возвращает новый объект `BookingDetails`.

Непараметризованная версия теста для этого диалогового окна будет выглядеть следующим образом:

```

[Fact]
public async Task DialogFlow()
{
    // Initial parameters
    var initialBookingDetails = new BookingDetails
    {
        Origin = "Seattle",
        Destination = null,
        TravelDate = null,
    };

    // Expected booking details
    var expectedBookingDetails = new BookingDetails
    {
        Origin = "Seattle",
        Destination = "New York",
        TravelDate = "2019-06-25",
    };

    var sut = new BookingDialog();
    var testClient = new DialogTestClient(Channels.Test, sut, initialBookingDetails);

    // Act/Assert
    var reply = await testClient.SendActivityAsync<IMessageActivity>("hi");
    ...

    var bookingResults = (BookingDetails)testClient.DialogTurnResult.Result;
    Assert.Equal(expectedBookingDetails.Origin, bookingResults?.Origin);
    Assert.Equal(expectedBookingDetails.Destination, bookingResults?.Destination);
    Assert.Equal(expectedBookingDetails.TravelDate, bookingResults?.TravelDate);
}

```

Чтобы параметризировать этот тест, мы создали класс `BookingDialogTestCase`, содержащий данные наших тестовых случаев. Он содержит начальный объект `BookingDetails`, ожидаемый объект `BookingDetails` и массив строк с пользовательскими речевыми фрагментами и ожидаемыми ответами из диалога для каждого шага.

```

public class BookingDialogTestCase
{
    public BookingDetails InitialBookingDetails { get; set; }

    public string[,] UtterancesAndReplies { get; set; }

    public BookingDetails ExpectedBookingDetails { get; set; }
}

```

Мы также создали вспомогательный класс `BookingDialogTestsDataGenerator`, который предоставляет метод `IEnumerable<object[]> BookingFlows()` для получения коллекции тестовых случаев, которые будут использоваться в teste.

Чтобы каждый тестовый случай отображался в виде отдельного элемента в обозревателе тестов Visual Studio, средство запуска тестов XUnit требует, чтобы сложные типы, например `BookingDialogTestCase`, реализовали `IXunitSerializable`. Для упрощения Bot.Builder.Testing предоставляет класс `TestDataObject`, который реализует этот интерфейс и который может использоваться для упаковки данных тестовых случаев без реализации `IXunitSerializable`.

Ниже приведен фрагмент кода `IEnumerable<object[]> BookingFlows()`, который показывает использование двух классов:

```

public static class BookingDialogTestsDataGenerator
{
    public static IEnumerable<object[]> BookingFlows()
    {
        // Create the first test case object
        var testCaseData = new BookingDialogTestCase
        {
            InitialBookingDetails = new BookingDetails(),
            UtterancesAndReplies = new[,]
            {
                { "hi", "Where would you like to travel to?" },
                { "Seattle", "Where are you traveling from?" },
                { "New York", "When would you like to travel?" },
                { "tomorrow", $"Please confirm, I have you traveling to: Seattle from: New York on: {DateTime.Now.AddDays(1):yyyy-MM-dd}. Is this correct? (1) Yes or (2) No" },
                { "yes", null },
            },
            ExpectedBookingDetails = new BookingDetails
            {
                Destination = "Seattle",
                Origin = "New York",
                TravelDate = $"{DateTime.Now.AddDays(1):yyyy-MM-dd}",
            },
        };
        // wrap the test case object into TestDataObject and return it.
        yield return new object[] { new TestDataObject(testCaseData) };

        // Create the second test case object
        testCaseData = new BookingDialogTestCase
        {
            InitialBookingDetails = new BookingDetails
            {
                Destination = "Seattle",
                Origin = "New York",
                TravelDate = null,
            },
            UtterancesAndReplies = new[,]
            {
                { "hi", "When would you like to travel?" },
                { "tomorrow", $"Please confirm, I have you traveling to: Seattle from: New York on: {DateTime.Now.AddDays(1):yyyy-MM-dd}. Is this correct? (1) Yes or (2) No" },
                { "yes", null },
            },
            ExpectedBookingDetails = new BookingDetails
            {
                Destination = "Seattle",
                Origin = "New York",
                TravelDate = $"{DateTime.Now.AddDays(1):yyyy-MM-dd}",
            },
        };
        // wrap the test case object into TestDataObject and return it.
        yield return new object[] { new TestDataObject(testCaseData) };
    }
}

```

Создав объект для хранения тестовых данных и предоставляющий коллекцию тестовых случаев класс,

мы воспользуемся атрибутом `MemberData` XUnit вместо `InlineData`, чтобы передать данные в тест.

Первый параметр для `MemberData` — это имя статической функции, которая возвращает коллекцию тестовых случаев, а второй параметр — это тип класса, который предоставляет этот метод.

```
[Theory]
[MemberData(nameof(BookingDialogTestsDataGenerator.BookingFlows), MemberType =
typeof(BookingDialogTestsDataGenerator))]
public async Task DialogFlowUseCases(TestDataObject testData)
{
    // Get the test data instance from TestDataObject
    var bookingTestData = testData.GetObject<BookingDialogTestCase>();
    var sut = new BookingDialog();
    var testClient = new DialogTestClient(Channels.Test, sut, bookingTestData.InitialBookingDetails);

    // Iterate over the utterances and replies array.
    for (var i = 0; i < bookingTestData.UtterancesAndReplies.GetLength(0); i++)
    {
        var reply = await testClient.SendActivityAsync<IMessageActivity>
(bookingTestData.UtterancesAndReplies[i, 0]);
        Assert.Equal(bookingTestData.UtterancesAndReplies[i, 1], reply?.Text);
    }

    // Assert the resulting BookingDetails object
    var bookingResults = (BookingDetails)testClient.DialogTurnResult.Result;
    Assert.Equal(bookingTestData.ExpectedBookingDetails?.Origin, bookingResults?.Origin);
    Assert.Equal(bookingTestData.ExpectedBookingDetails?.Destination, bookingResults?.Destination);
    Assert.Equal(bookingTestData.ExpectedBookingDetails?.TravelDate, bookingResults?.TravelDate);
}
}
```

Ниже приведен пример результатов для тестов `DialogFlowUseCases` в обозревателе тестов Visual Studio при их выполнении.

▲ ✓ DialogFlowUseCases (5)	575 ms
✓ DialogFlowUseCases(testData: TestDataObject { TestObject = "{\"Name\":\"All booking details given for today\",... }")	230 ms
✓ DialogFlowUseCases(testData: TestDataObject { TestObject = "{\"Name\":\"Destination and Origin given\",\"Initi...\"}")	3 ms
✓ DialogFlowUseCases(testData: TestDataObject { TestObject = "{\"Name\":\"Destination given\",\"InitialBookingDe...\"}")	332 ms
✓ DialogFlowUseCases(testData: TestDataObject { TestObject = "{\"Name\":\"Full flow with 'no' at confirmation\",...\"}")	6 ms
✓ DialogFlowUseCases(testData: TestDataObject { TestObject = "{\"Name\":\"Full flow\",\"InitialBookingDetails\":...\"}")	4 ms

## Использование макетов

Вы можете использовать макеты для элементов, которые вы сейчас не тестируете. В справочных целях этот уровень можно рассматривать как модульное и интеграционное тестирование.

Расмакетирование как можно больше элементов, что позволит лучше изолировать тестируемую часть. Макетами элементов может быть хранилище, адаптер, ПО промежуточного слоя, конвейер действия, каналы и все, что напрямую не является частью бота. Использование макетов также может предусматривать временное удаление определенных элементов, например ПО промежуточного слоя, являющегося частью тестируемого бота. Это нужно для изоляции каждого элемента. Тем не менее, если вы тестируете ПО промежуточного слоя, возможно, вы захотите макетировать бот вместо этого.

Макетирование элементов может принимать разнообразные формы — от замены элемента другим известным объектом до реализации простой функции Hello World. Это также может привести к простому удалению элемента, если это не требуется, или просто заставить его ничего не делать.

Макеты позволяют настроить зависимости диалога и убедиться, что они находятся в известном состоянии во время выполнения теста. При этом вам не нужно полагаться на внешние ресурсы, включая базы данных, модели LUIS или другие объекты.

Чтобы облегчить тестирование диалогов и сократить зависимости от внешних объектов, вам может потребоваться внедрить внешние зависимости в конструктор диалогов.

Например, вместо создания экземпляра `BookingDialog` в `MainDialog` сделайте следующее:

- C#
- JavaScript

```
public MainDialog()
    : base(nameof(MainDialog))
{
    ...
    AddDialog(new BookingDialog());
    ...
}
```

Мы передаем экземпляр `BookingDialog` в качестве параметра конструктора:

- C#
- JavaScript

```
public MainDialog(BookingDialog bookingDialog)
    : base(nameof(MainDialog))
{
    ...
    AddDialog(bookingDialog);
    ...
}
```

Так мы сможем заменить экземпляра `BookingDialog` макетом объекта и написать модульные тесты для `MainDialog`, не вызывая фактический класс `BookingDialog`.

- C#
- JavaScript

```
// Create the mock object
var mockDialog = new Mock<BookingDialog>();

// Use the mock object to instantiate MainDialog
var sut = new MainDialog(mockDialog.Object);

var testClient = new DialogTestClient(Channels.Test, sut);
```

## Макетирование диалогов

Как описано выше, `MainDialog` вызывает `BookingDialog` для получения объекта `BookingDetails`. Чтобы реализовать и настроить экземпляр макета `BookingDialog`, сделайте следующее:

- C#
- JavaScript

```

// Create the mock object for BookingDialog.
var mockDialog = new Mock<BookingDialog>();
mockDialog
    .Setup(x => x.BeginDialogAsync(It.IsAny<DialogContext>(), It.IsAny<object>(),
It.IsAny<CancellationToken>()))
    .Returns(async (DialogContext dialogContext, object options, CancellationToken cancellationToken) =>
{
    // Send a generic activity so we can assert that the dialog was invoked.
    await dialogContext.Context.SendActivityAsync($"{mockDialog.Name} mock invoked",
cancellationToken: cancellationToken);

    // Create the BookingDetails instance we want the mock object to return.
    var expectedBookingDialogResult = new BookingDetails()
    {
        Destination = "Seattle",
        Origin = "New York",
        TravelDate = $"{DateTime.UtcNow.AddDays(1):yyyy-MM-dd}"
    };

    // Return the BookingDetails we need without executing the dialog logic.
    return await dialogContext.EndDialogAsync(expectedBookingDialogResult, cancellationToken);
});

// Create the sut (System Under Test) using the mock booking dialog.
var sut = new MainDialog(mockDialog.Object);

```

В этом примере мы использовали [Moq](#) для создания макета диалога, а также методы `Setup` и `Returns` для настройки его поведения.

## Макетирование результатов LUIS

В простых сценариях можно реализовать результаты макета LUIS с помощью кода. Для этого сделайте следующее:

- [C#](#)
- [JavaScript](#)

```

var mockRecognizer = new Mock<IRecognizer>();
mockRecognizer
    .Setup(x => x.RecognizeAsync<FlightBooking>(It.IsAny<ITurnContext>(), It.IsAny<CancellationToken>()))
    .Returns(() =>
{
    var luisResult = new FlightBooking
    {
        Intents = new Dictionary<FlightBooking.Intent, IntentScore>
        {
            { FlightBooking.Intent.BookFlight, new IntentScore() { Score = 1 } },
        },
        Entities = new FlightBooking._Entities(),
    };
    return Task.FromResult(luisResult);
});

```

Но результаты LUIS могут быть сложными. В таком случае проще записать желаемый результат в JSON-файл, добавить его в проект в качестве ресурса и десериализовать в результат LUIS. Например:

- [C#](#)
- [JavaScript](#)

```
var mockRecognizer = new Mock<IRecognizer>();
mockRecognizer
    .Setup(x => x.RecognizeAsync<FlightBooking>(It.IsAny<ITurnContext>(), It.IsAny<CancellationToken>()))
    .Returns(() =>
{
    // Deserialize the LUIS result from embedded json file in the TestData folder.
    var bookingResult = GetEmbeddedTestData($"'{GetType().Namespace}'.TestData.FlightToMadrid.json");

    // Return the deserialized LUIS result.
    return Task.FromResult(bookingResult);
});
```

## Дополнительные сведения

- [Пример теста CoreBot Test \(C#\)](#)
- [Пример теста CoreBot Test \(JavaScript\)](#)
- [Тестирование ботов](#)

# Добавление действий трассировки в бот

27.10.2020 • 7 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

*Действием трассировки* называется действие, которое бот может отправить в Bot Framework Emulator. Вы можете использовать действия трассировки для интерактивной отладки бота, так как они позволяют просматривать сведения о боте при локальном выполнении.

Действия трассировки отправляются только в приложение Emulator, но не другим клиентам и не в каналы. Приложение отображает их в журнале, но не на главной панели разговора.

- Действия трассировки из контекста шага отправляются через *обработчики действий отправки*, которые зарегистрированы в контексте этого шага.
- Действия трассировки из контекста шага привязываются к входящему действию путем применения ссылки на беседу, если таковая имеется. Для упрахдающего сообщения *идентификатор адреса для ответа* будет содержать новый уникальный идентификатор.
- Независимо от метода отправки действие трассировки никогда не устанавливает флаг *responded* (получен ответ).

## Использование действия трассировки

Чтобы увидеть действие трассировки в приложении Emulator, создайте сценарий, в котором бот отправляет такое действие трассировки. Например, можно создать исключение и отправить действие трассировки из обработчика ошибок на шаге адаптера.

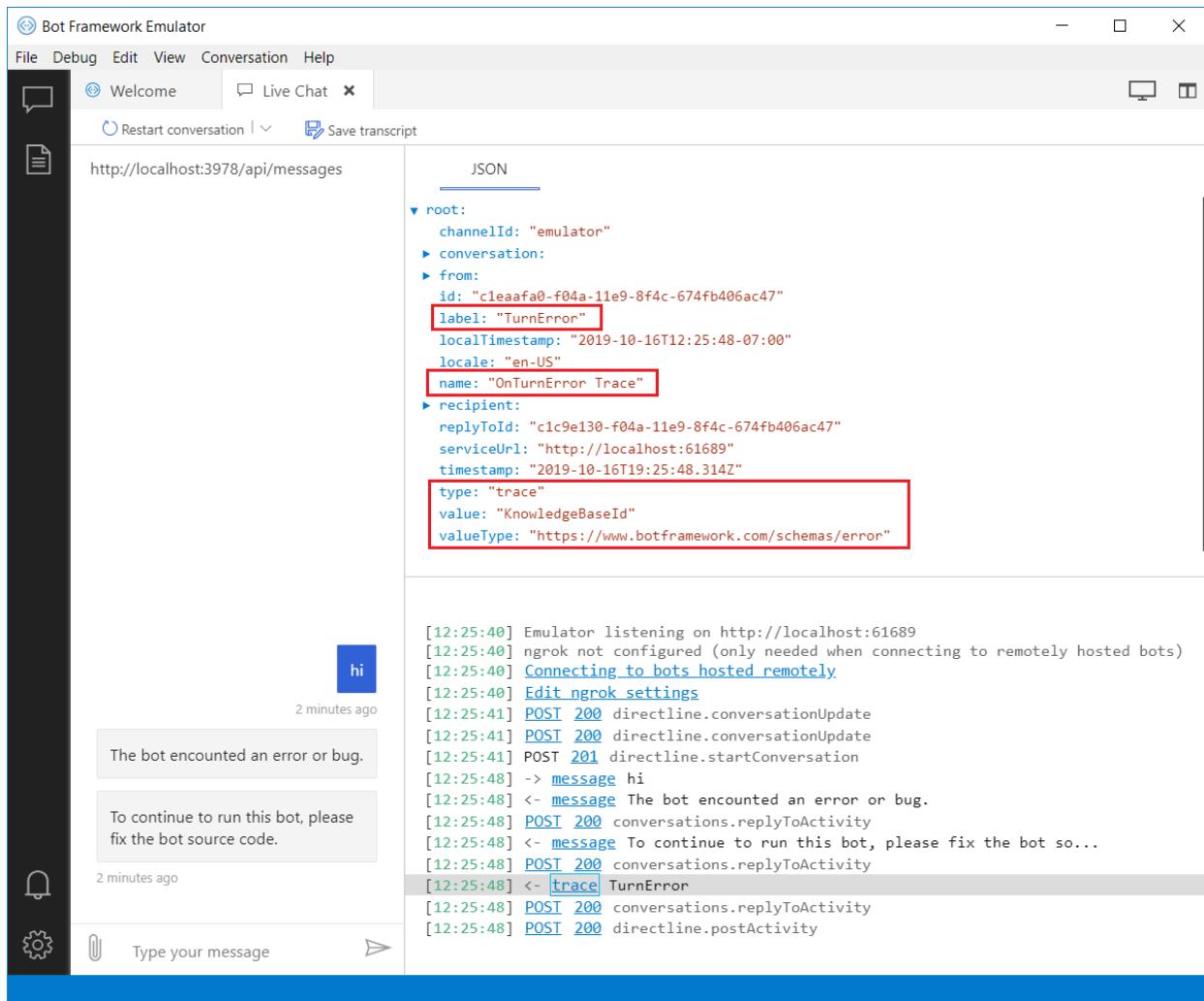
Чтобы отправить из бота действие трассировки, сделайте следующее.

1. Создайте новое действие.
  - Присвойте его свойству *type* (тип) значение "trace" (трассировка). Это обязательный шаг.
  - При желании задайте для трассировки подходящие значения свойств *name* (имя), *label* (метка), *value* (значение) и *value type* (тип значения).
2. Чтобы отправить действие трассировки, используйте *действие отправки* из объекта контекста шага.
  - Этот метод добавляет значения для незаполненных обязательных свойств действия, используя сведения о входящем действии. Сюда относятся свойства *channel ID* (идентификатор канала), *service URL* (URL-адрес службы), *from* (от кого) и *recipient* (получатель).

Чтобы просмотреть действие трассировки в приложении Emulator, выполните следующие действия.

1. Запустите бота на локальном компьютере.
2. Выполните тестирование в Emulator.
  - Напишите что-нибудь боту, чтобы сработали те шаги в сценарии, которые создают действие трассировки.
  - Когда бот создаст действие трассировки, оно отобразится в журнале Emulator.

Ниже приведен пример операции трассировки, которую вы увидите при запуске бота Core без настройки базы знаний QnA Maker, от которой зависит этот бот.



## Добавление действия трассировки в обработчик ошибок адаптера

Обработчик ошибок шага адаптера перехватывает все неперехваченные в другом месте исключения, которые создаются ботом во время выполнения. Это подходящее место для размещения действия трассировки, так как вы можете отправить понятное сообщение пользователю и отладочную информацию о возникшем исключении в Emulator.

В этом случае используется код из примера Core Bot. Ознакомьтесь с полным примером для [C#](#), [JavaScript](#) и [Python](#).

- [C#](#)
- [JavaScript](#)
- [Python](#)

Обработчик адаптера `OnTurnError` создает действие трассировки, которое собирает сведения об исключении и отправляет их в эмулятор.

`AdapterWithErrorHandler.cs`

```

public AdapterWithErrorHandler(IConfiguration configuration, ILogger<BotFrameworkHttpAdapter> logger,
ConversationState conversationState = null)
    : base(configuration, logger)
{
    OnTurnError = async (turnContext, exception) =>
    {
        // Log any leaked exception from the application.
        // NOTE: In production environment, you should consider logging this to
        // Azure Application Insights. Visit https://aka.ms/bottelemetry to see how
        // to add telemetry capture to your bot.
        logger.LogError(exception, $"[OnTurnError] unhandled error : {exception.Message}");

        // Send a message to the user
        var errorMessageText = "The bot encountered an error or bug.";
        var errorMessage = MessageFactory.Text(errorMessageText, errorMessageText,
InputHints.IgnoringInput);
        await turnContext.SendActivityAsync(errorMessage);

        errorMessageText = "To continue to run this bot, please fix the bot source code.";
        errorMessage = MessageFactory.Text(errorMessageText, errorMessageText, InputHints.ExpectingInput);
        await turnContext.SendActivityAsync(errorMessage);

        if (conversationState != null)
        {
            try
            {
                // Delete the conversationState for the current conversation to prevent the
                // bot from getting stuck in a error-loop caused by being in a bad state.
                // ConversationState should be thought of as similar to "cookie-state" in a Web pages.
                await conversationState.DeleteAsync(turnContext);
            }
            catch (Exception e)
            {
                logger.LogError(e, $"Exception caught on attempting to Delete ConversationState :
{e.Message}");
            }
        }

        // Send a trace activity, which will be displayed in the Bot Framework Emulator
        await turnContext.TraceActivityAsync("OnTurnError Trace", exception.Message,
"https://www.botframework.com/schemas/error", "TurnError");
    };
}

```

## Дополнительные ресурсы

- В статье [Отладка бота с помощью проверяющего ПО промежуточного слоя](#) описывается, как добавить ПО промежуточного слоя для создания действий трассировки.
- Для отладки развернутого робота можно применить Application Insights. Дополнительные сведения см. в статье о [добавлении телеметрии в бот](#).
- Дополнительные сведения о типах действий см. в статье о [схеме действий Bot Framework](#).

# Отладка Bot

27.03.2021 • 8 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье описывается отладка ботов с помощью интегрированной среды разработки (IDE), такой как Visual Studio или Visual Studio Code, и Bot Framework Emulator. Эти методы можно использовать для локальной отладки любого бота. Но для работы с этой статьей используется бот [C#, Javascript](#) или [Python](#), созданный в рамках краткого руководства.

## NOTE

В этой статье предполагается, что вы используете Bot Framework Emulator для отправки сообщений в бот и получения их от бота во время отладки. См. также об [отладке бота с помощью Bot Framework Emulator](#).

## Предварительные требования

- Скачайте и установите [Bot Framework Emulator](#).
- Скачайте и установите [Visual Studio Code](#) или [Visual Studio](#) (Community Edition или более поздней версии).
- [C#](#)
- [JavaScript](#)
- [Python](#)

### Установка точек останова в Visual Studio Code

В Visual Studio Code можно установить точки останова и запустить бот в режиме отладки для пошагового выполнения кода. Чтобы установить точки останова в VS Code, выполните следующие действия.

1. Запустите VS Code и откройте папку проекта бота.
2. При необходимости задайте точки останова. Вы можете сделать это, наведя указатель мыши на столбец слева от номеров строк. Появится красная точка. Если щелкнуть на нее, установятся точки останова. Если щелкнуть на нее снова, точки останова будут удалены.
3. В строке меню щелкните **Выполнить**, а затем — **Начать отладку**. Бот запустится в режиме отладки из терминала в Visual Studio Code.

```
3
4 using System.Collections.Generic;
5 using System.Threading;
6 using System.Threading.Tasks;
7 using Microsoft.Bot.Builder;
8 using Microsoft.Bot.Schema;
9
10 namespace Microsoft.BotBuilderSamples.Bots
11 {
12     public class EchoBot : ActivityHandler
13     {
14         protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext, CancellationToken cancellationToken)
15         {
16             var replyText = $"Echo: {turnContext.Activity.Text}";
17             await turnContext.SendActivityAsync(MessageFactory.Text(replyText, replyText));
18         }
19
20         protected override async Task OnMembersAddedAsync(IList<ChannelAccount> membersAdded, ITurnContext<IMemberAddedActivity> turnContext, CancellationToken cancellationToken)
21         {
22             var welcomeText = "Hello and welcome!";
23             foreach (var member in membersAdded)
24             {
25                 if (member.Id != turnContext.Activity.Recipient.Id)
26                 {
27                     await turnContext.SendActivityAsync(MessageFactory.Text(welcomeText, welcomeText));
28                 }
29             }
29 }
```

4. Запустите Bot Framework Emulator и подключитесь к боту, как описано в руководстве по [отлаживанию с помощью Bot Framework Emulator](#).
5. В эмуляторе отправьте сообщение Bot (например, отправьте сообщение «Hi»). Выполнение останавливается на строке, в которой установлена точка останова.

```
3
4 using System.Collections.Generic;
5 using System.Threading;
6 using System.Threading.Tasks;
7 using Microsoft.Bot.Builder;
8 using Microsoft.Bot.Schema;
9
10 namespace Microsoft.BotBuilderSamples.Bots
11 {
12     public class EchoBot : ActivityHandler
13     {
14         protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext, CancellationToken cancellationToken)
15         {
16             var replyText = $"Echo: {turnContext.Activity.Text}";
17             await turnContext.SendActivityAsync(MessageFactory.Text(replyText, replyText), cancellationToken);
18         }
19
20         protected override async Task OnMembersAddedAsync(IList<ChannelAccount> membersAdded, ITurnContext<IMemberAddedActivity> turnContext, CancellationToken cancellationToken)
21         {
22             var welcomeText = "Hello and welcome!";
22 }
```

### Установка точек останова в Visual Studio

В Visual Studio (VS) можно установить точки останова и запустить бот в режиме отладки для пошагового выполнения кода. Чтобы установить точки останова в VS, выполните следующие действия.

1. Перейдите к папке ботов и откройте файл SLN. Откроется решение в Visual Studio.
2. В строке меню щелкните **Построить**, затем **Построить решение**.
3. В **обозревателе решений** щелкните файл .cs и задайте требуемые точки останова. Этот файл определяет логику основного бота. В VS можно установить точки останова, наведя указатель мыши на столбец слева от номеров строк. Появится красная точка. Щелкнув ее, вы зададите точки останова. Если щелкнуть ее снова, точки останова будут удалены.
4. В меню щелкните **Отладка**, а затем — **Начать отладку**. На этом этапе бот функционирует локально.

```

    /// <summary>
    /// Every conversation turn for our Echo Bot will call this method.
    /// There are no dialogs used, since it's "single turn" processing, meaning a single
    /// request and response.
    /// </summary>
    /// <param name="turnContext">A <see cref="ITurnContext"/> containing all the data needed
    /// for processing this conversation turn. </param>
    /// <param name="cancellationToken">(Optional) A <see cref=" CancellationToken "/> that can be used by other objects
    /// or threads to receive notice of cancellation.</param>
    /// <returns>
    /// <see cref="Task"/> that represents the work queued to execute.</returns>
    /// <seesalso cref="BotStateSet"/>
    /// <seesalso cref="ConversationState"/>
    /// <seesalso cref="IMiddleware"/>
    public async Task OnTurnAsync(ITurnContext turnContext, CancellationToken cancellationToken = default(CancellationToken))
    {
        // Handle Message activity type, which is the main activity type for shown within a conversational interface.
        // Message activities may contain text, speech, interactive cards, and binary or unknown attachments.
        // see https://aka.ms/about-bot-activity-message to learn more about the message and other activity types
        if (turnContext.Activity.Type == ActivityTypes.Message)
        {
            // Get the conversation state from the turn context.
            var state = await _accessors.CounterState.GetAsync(turnContext, () => new CounterState());
            // Bump the turn count for this conversation.
            state.TurnCount++;

            // Set the property using the accessor.
            await _accessors.CounterState.SetAsync(turnContext, state);

            // Save the new turn count into the conversation state.
            await _accessors.ConversationState.SaveChangesAsync(turnContext);

            // Echo back to the user whatever they typed.
            var responseMessage = $"Turn {state.TurnCount}: You sent '{turnContext.Activity.Text}'\n";
            await turnContext.SendActivityAsync(responseMessage);
        }
        else
        {
            await turnContext.SendActivityAsync($"'{turnContext.Activity.Type} event detected'");
        }
    }
}

```

1. Запустите Bot Framework Emulator и подключитесь к боту, как описано выше.
2. В эмуляторе отправьте сообщение Bot (например, отправьте сообщение «Hi»). Выполнение останавливается на строке, в которой установлена точка останова.

```

    /// <summary>
    /// Every conversation turn for our Echo Bot will call this method.
    /// There are no dialogs used, since it's "single turn" processing, meaning a single
    /// request and response.
    /// </summary>
    /// <param name="turnContext">A <see cref="ITurnContext"/> containing all the data needed
    /// for processing this conversation turn. </param>
    /// <param name="cancellationToken">(Optional) A <see cref=" CancellationToken "/> that can be used by other objects
    /// or threads to receive notice of cancellation.</param>
    /// <returns>
    /// <see cref="Task"/> that represents the work queued to execute.</returns>
    /// <seesalso cref="BotStateSet"/>
    /// <seesalso cref="ConversationState"/>
    /// <seesalso cref="IMiddleware"/>
    public async Task OnTurnAsync(ITurnContext turnContext, CancellationToken cancellationToken = default(CancellationToken))
    {
        // Handle Message activity type, which is the main activity type for shown within a conversational interface.
        // Message activities may contain text, speech, interactive cards, and binary or unknown attachments.
        // see https://aka.ms/about-bot-activity-message to learn more about the message and other activity types
        if (turnContext.Activity.Type == ActivityTypes.Message)
        {
            // Get the conversation state from the turn context.
            var state = await _accessors.CounterState.GetAsync(turnContext, () => new CounterState());
            // Bump the turn count for this conversation.
            state.TurnCount++;

            // Set the property using the accessor.
            await _accessors.CounterState.SetAsync(turnContext, state);

            // Save the new turn count into the conversation state.
            await _accessors.ConversationState.SaveChangesAsync(turnContext);

            // Echo back to the user whatever they typed.
            var responseMessage = $"Turn {state.TurnCount}: You sent '{turnContext.Activity.Text}'\n";
            await turnContext.SendActivityAsync(responseMessage);
        }
        else
        {
            await turnContext.SendActivityAsync($"'{turnContext.Activity.Type} event detected'");
        }
    }
}

```

## Дополнительные ресурсы

- См. [статью об устранении общих проблем](#) и другие статьи об устранении неполадок в этом разделе.
- См. подробнее об [отладке ботов с помощью Emulator](#).

## Дальнейшие действия

[Отладка бота с помощью файлов записей разговоров](#)

# Отладка с помощью эмулятора

27.03.2021 • 12 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Эмулятор Bot Framework — это классическое приложение, позволяющее Bot-разработчикам тестировать и отлаживать программы-роботы, как локально, так и удаленно. С помощью эмулятора вы можете общаться с программой-роботом и проверять сообщения, которые отправляет и получает Bot. Эмулятор отображает сообщения в том виде, в котором они отображались в пользовательском интерфейсе веб-чата, и записывает запросы и ответы JSON при обмене сообщениями с помощью программы Bot. Перед развертыванием программы Bot в облаке запустите ее локально и протестируйте с помощью эмулятора. Вы можете протестировать программу-робота с помощью эмулятора, даже если вы еще не [создали](#) ее с помощью службы Azure Bot или не настроили ее для запуска на всех каналах.

## Предварительные требования

- Установите [Bot Framework Emulator](#).

## Локальный запуск бота

Подключаемого к Bot Framework Emulator бота нужно сперва запустить локально. Это можно сделать с помощью Visual Studio, Visual Studio Code или командной строки. Чтобы запустить бота с помощью командной строки, сделайте следующее:

- [C#](#)
- [JavaScript](#)
- [Python](#)
- В командной строке измените каталог на каталог проекта ботов.
- Запустите бота, выполнив следующую команду:

```
dotnet run
```

- Скопируйте номер порта в строке перед текстом *Application started. Press CTRL+C to shut down.*

```
Now listening on: https://localhost:3979
Now listening on: http://localhost:3978
Application started. Press Ctrl+C to shut down.
```

На этом этапе бот должен запуститься локально.

## Подключение к боту, работающему на локальном узле

### Настройка параметров прокси

При разработке за корпоративным прокси-сервером эмулятор будет использовать настроенные переменные среды `HTTP_PROXY` и `HTTPS_PROXY`, указывающие маршрут URL-адреса прокси-сервера для запросов HTTP и HTTPS соответственно.

При подключении к программе-роботу, работающей на `localhost`, эмулятор сначала попытается выполнить маршрутизацию через прокси, прежде чем подключиться к `localhost`. Как правило, прокси-

сервер блокирует подключение, если не указано, что его следует обходить `localhost`.

Чтобы обойти `HTTP_PROXY` `HTTPS_PROXY` Параметры и разрешить эмулятору подключаться к `localhost`, на локальном компьютере необходимо определить следующую переменную среды:

```
NO_PROXY=localhost
```

## Настройка эмулятора для проверки подлинности

Если для Bot требуется проверка подлинности, то при отображении диалогового окна входа необходимо настроить эмулятор, как показано ниже.

### Использование кода проверки входа в систему

1. Запустите эмулятор Bot Framework.
2. В эмуляторе щелкните значок шестеренки в левом нижнем углу или вкладку **Параметры эмулятора** в правом верхнем углу.
3. Установите флажок **Use a sign-in verification code for OAuthCards** (Использовать код проверки входа в систему для OAuthCards)
4. Установите флажок **Bypass ngrok for local address** (Обходить ngrok для локального адреса).
5. Нажмите кнопку **Сохранить**.

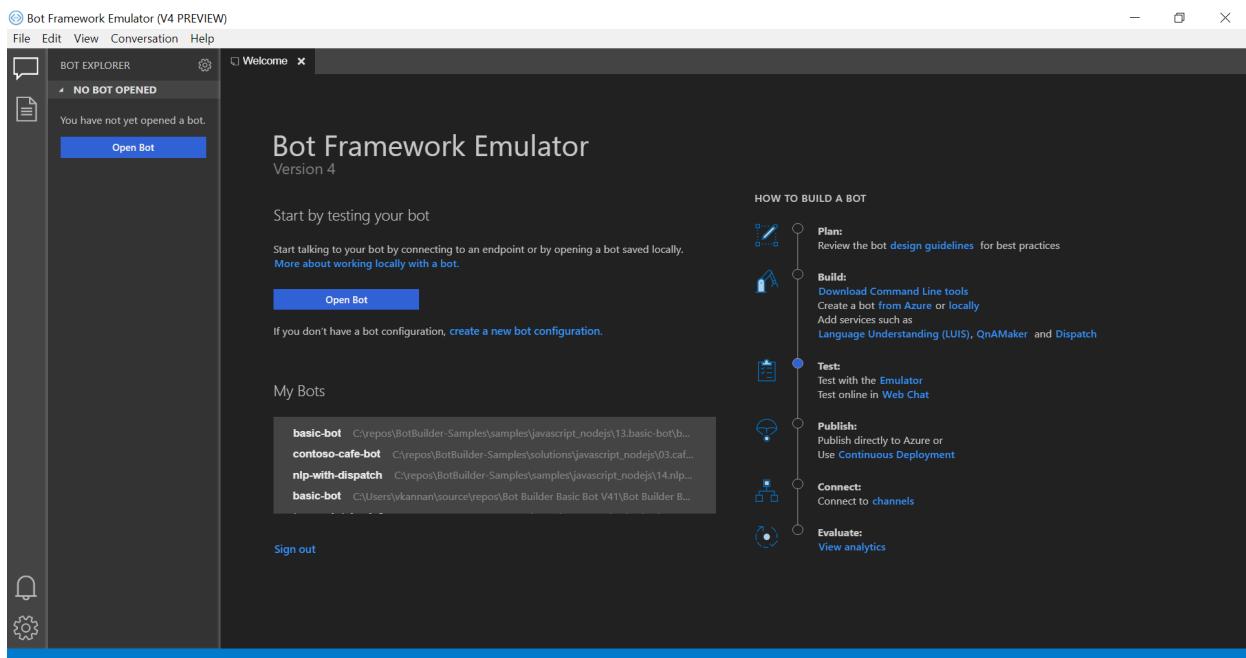
При нажатии кнопки входа в боте будет создан код проверки. Для выполнения проверки подлинности необходимо ввести код в поле входного разговора Bot. После этого можно выполнять разрешенные операции.

Кроме того, вы можете выполнить описанные ниже действия.

### Использование маркеров аутентификации

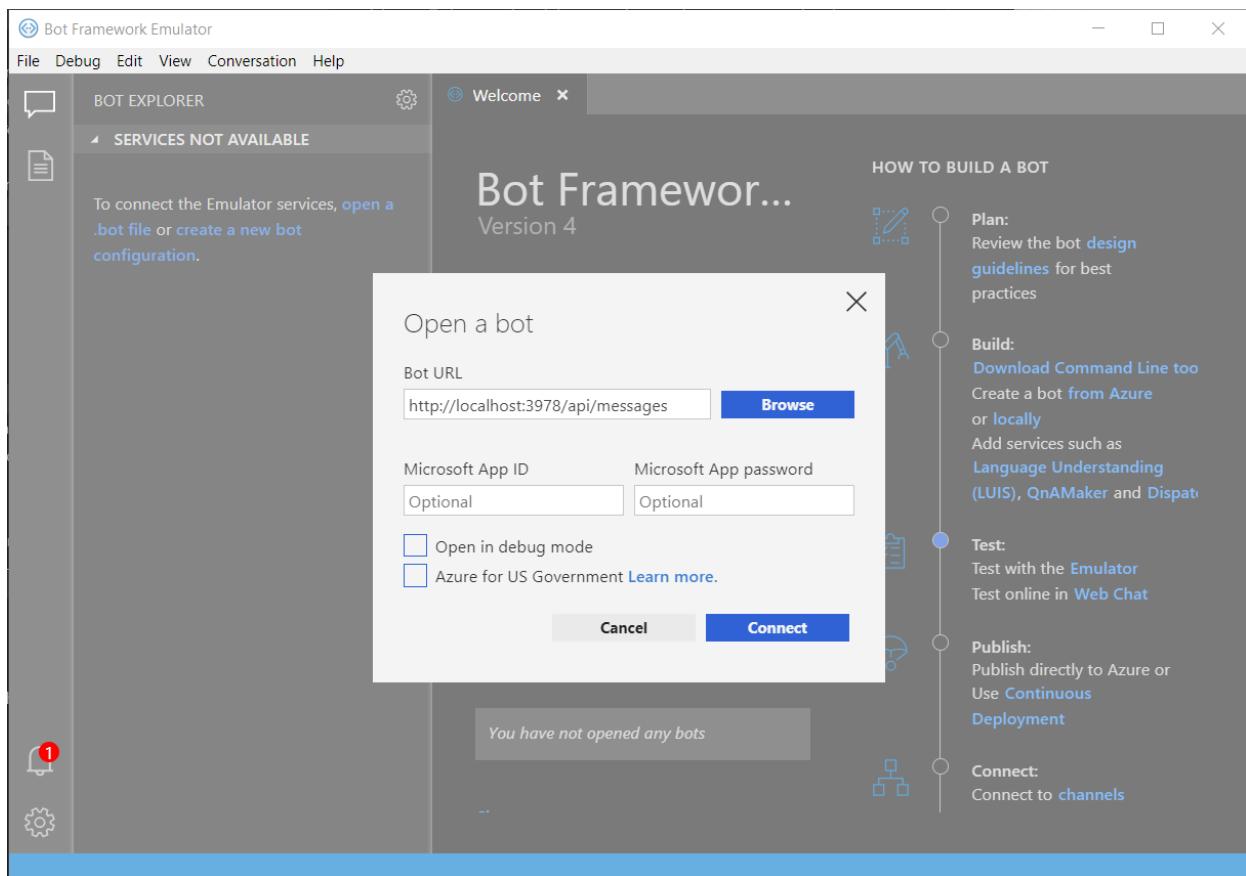
1. Запустите эмулятор Bot Framework.
2. В эмуляторе щелкните значок шестеренки в левом нижнем углу или вкладку **Параметры эмулятора** в правом верхнем углу.
3. Установите флажок **Use version1.0 authentication tokens** (Использовать маркеры аутентификации версии 1.0).
4. Введите локальный путь к средству ngrok. См. дополнительные сведения о [ngrok](#).
5. Установите флажок **Run ngrok when the Emulator starts up** (Запустить ngrok при запуске эмулятора).
6. Нажмите кнопку **Сохранить**.

Когда вы нажмете кнопку входа в боте, вам будет предложено ввести свои учетные данные. Будет создан маркер аутентификации. После этого можно выполнять разрешенные операции.



Чтобы подключиться к запущенному локально боту, щелкните **Открыть бота**. Добавьте скопированный ранее номер порта в следующий URL-адрес и вставьте полученный URL-адрес в поле "URL-адрес бота":

`http://localhost:номер_порта/api/messages`



Если бот выполняется с [учетными данными учетной записи Майクロсофт \(MSA\)](#), введите эти учетные данные.

### Использование учетных данных бота

Открыв бот, задайте **идентификатор приложения Майクロсофт и пароль приложения**

**Майкрософт**, если бот выполняется с учетными данными. Если вы создали бот с помощью службы Azure Bot, учетные данные доступны в Службе приложений бота в разделе **Параметры -> Конфигурация**. Если вы не знаете значения, можно удалить их из файла конфигурации локально запущенного бота, а

затем запустить бот в эмуляторе. Если программа-робот не работает с этими параметрами, вам не нужно запускать эмулятор с параметрами.

При создании приложения поставщика удостоверений AD учитывайте следующее:

- Если для поддерживаемых типов учетных записей задано значение "один клиент", то при использовании личной подписки вместо учетная запись Майкрософт эмулятор выдаст ошибку: *идентификатор приложения-робота или пароль Microsoft App не указаны правильно*.
- В этом случае для поддерживаемых типов учетных записей нужно указать *Учетные записи в любом каталоге организации (любой каталог Azure AD — мультитенантный) и персональные учетные записи Майкрософт (например, Xbox)*.

Дополнительные сведения см. в статьях [Создание приложения поставщика удостоверений Azure AD](#) и [Регистрация нового приложения с помощью портал Azure](#).

## Просмотр действий с сообщениями с помощью инспектора

Отправьте боту сообщение и получите от него ответ. Сообщение можно щелкнуть в окне беседы, чтобы просмотреть необработанные действия JSON с помощью **компоненты для проверки** в правой части окна. При выборе сообщение станет желтым, и объект действия JSON будет отображаться слева от окна чата. Данные JSON включают метаданные ключа, в том числе идентификатор канала, тип действия, идентификатор беседы, текстовое сообщение, URL-адрес конечной точки и т. д. Можно проверить действия, отправленные пользователем, а также действия, с помощью которых бот отвечает.

The screenshot shows the Microsoft Bot Emulator interface. On the left, a conversation window displays a message from a user ("Please enter your mode of transport.") and a bot response ("hi"). Below the messages are three buttons: "Car", "Bus", and "Bicycle". On the right, there are two panes: "INSPECTOR - JSON" which shows the raw JSON message, and "LOG" which shows the bot's internal log output.

```
{ "attachments": [], "channelId": "emulator", "conversation": { "id": "9680fab0-50a2-11e9-90e5-b95b5f203e63|livechat" }, "entities": [], "from": { "id": "c219a680-4f36-11e9-90e5-b95b5f203e63", "name": "Bot", "role": "bot" }, "id": "992036a0-50a2-11e9-862b-37a65f22ae7e", "inputHint": "acceptingInput", "localTimestamp": "2019-03-27T08:11:29-07:00", "locale": "", "recipient": { "id": "7c32a522-2552-4728-92cb-3a2fb7951e0b",
```

```
[08:11:25] POST 201 directline.startConversation
[08:11:25] Emulator listening on http://localhost:61023
[08:11:25] ngrok not configured (only needed when connecting to
remotely hosted bots)
[08:11:25] Connecting to bots hosted remotely
[08:11:25] Edit ngrok settings
[08:11:29] ->message_hi
```

### TIP

Вы можете отлаживать изменения состояния в подключенном к каналу боте, добавив в него [проверяющее ПО промежуточного слоя](#).

## Проверка служб

С помощью эмулятора v4 можно также проверить ответы JSON из LUIS и QnA. Для бота с подключенными языковыми службами в окне журнала в нижнем правом углу можно выбрать **трассировку**. Это новое средство также предоставляет функции для обновления языковых служб непосредственно из эмулятора.

```

LOG
[11:38:45] POST 200 conversations.replyToActivity
[11:38:45] <- message Hello and welcome to the Luis Sample bot.
[11:39:01] -> message run tests
[11:39:02] POST 200 conversations.replyToActivity
[11:39:02] <- trace Luis Trace
[11:39:02] POST 200 conversations.replyToActivity
[11:39:02] <- message The **top intent** was: **'RunTests'**, with score...
[11:39:02] POST 200 conversations.replyToActivity
[11:39:02] <- message Detail of intents scorings:
[11:39:02] POST 200 conversations.replyToActivity
[11:39:02] <- message * 'RunTests', score 0.9753478 * 'Build', score 0....
[11:39:02] POST 200 directline.postActivity

```

Если подключена служба LUIS, вы заметите, что в ссылке трассировки указана **трассировка Luis**. Выбрав ее, вы увидите необработанный ответ от службы LUIS, включающий намерения, сущности и соответствующие оценки. Также имеется возможность переназначить намерения для фраз пользователей.

```

LOG
[11:22:29] Emulator listening on http://localhost:49863
[11:22:29] ngrok listening on https://899d5e3b.ngrok.io
[11:22:29] ngrok traffic inspector: http://127.0.0.1:4040
[11:22:29] Will bypass ngrok for local addresses
[11:22:29] POST 201 directline.startConversation
[11:22:29] POST 200 conversations.replyToActivity
[11:22:29] <- message Hello and welcome to the QnA Maker Sample bot.
[11:22:36] -> message hi
[11:22:37] POST 200 conversations.replyToActivity
[11:22:37] <- trace QnAMaker Trace
[11:22:37] POST 200 conversations.replyToActivity
[11:22:37] <- message Hello!
[11:22:37] POST 200 directline.postActivity

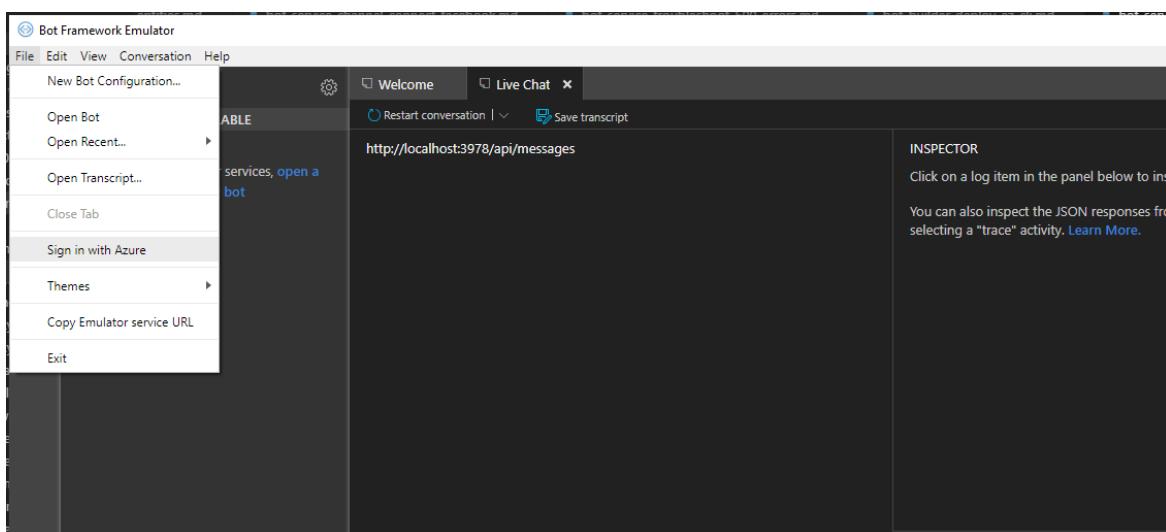
```

Если подключена служба QnA, журнал отобразит **трассировку QnA**. Выбрав ее, можно выполнить предварительный просмотр пары "вопрос-ответ", связанной с этим действием, наряду с оценкой достоверности. Здесь можно добавить выражения альтернативных вопросов для ответа.

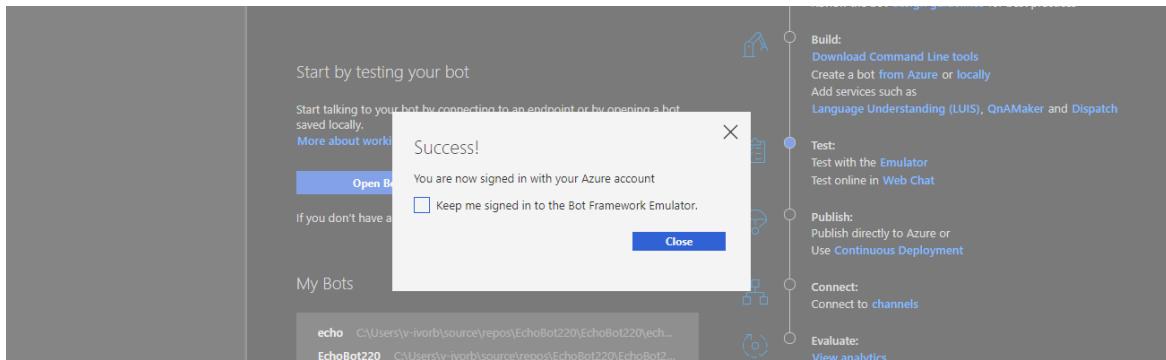
## Вход в Azure

Для входа в учетную запись Azure можно использовать Эмулятор. Это особенно полезно для добавления служб, от которых зависит ваш бот, и управления ими. Войдите в Azure, сделав следующее:

1. Щелкните файл —> войти с помощью Azure.



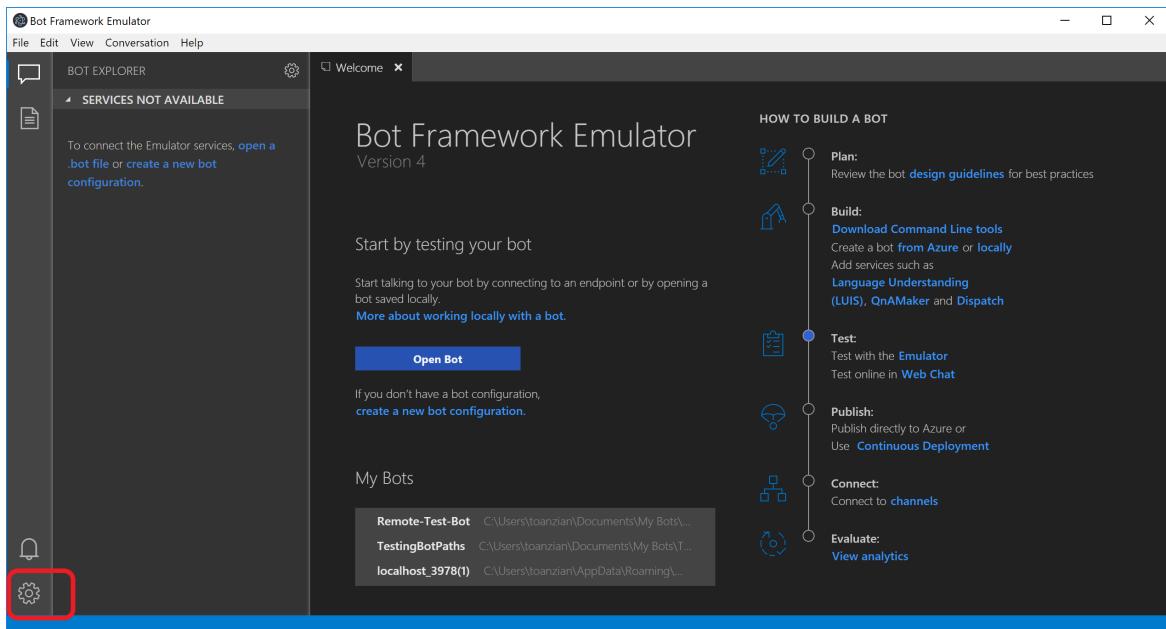
2. На экране приветствия щелкните Вход с помощью учетной записи Azure. При необходимости эмулятор может быть в курсе перезапуска приложений эмулятора.



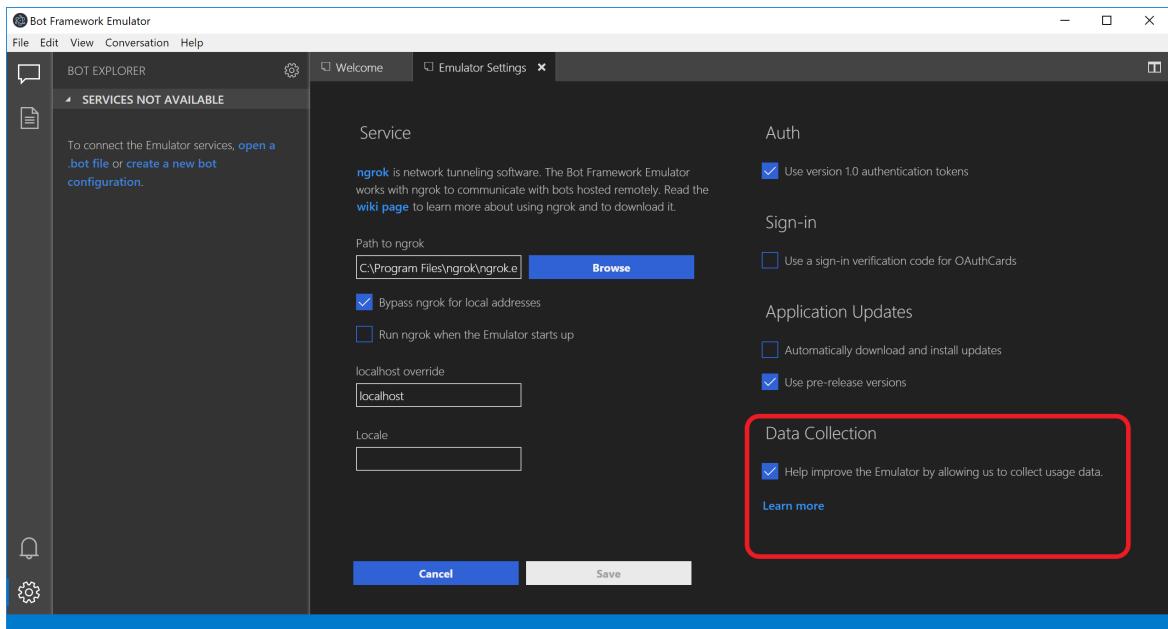
## Отключение сбора данных

Если вы решите, что Эмулятор не должен далее собирать данные об использовании, сбор данных можно отключить следующим образом.

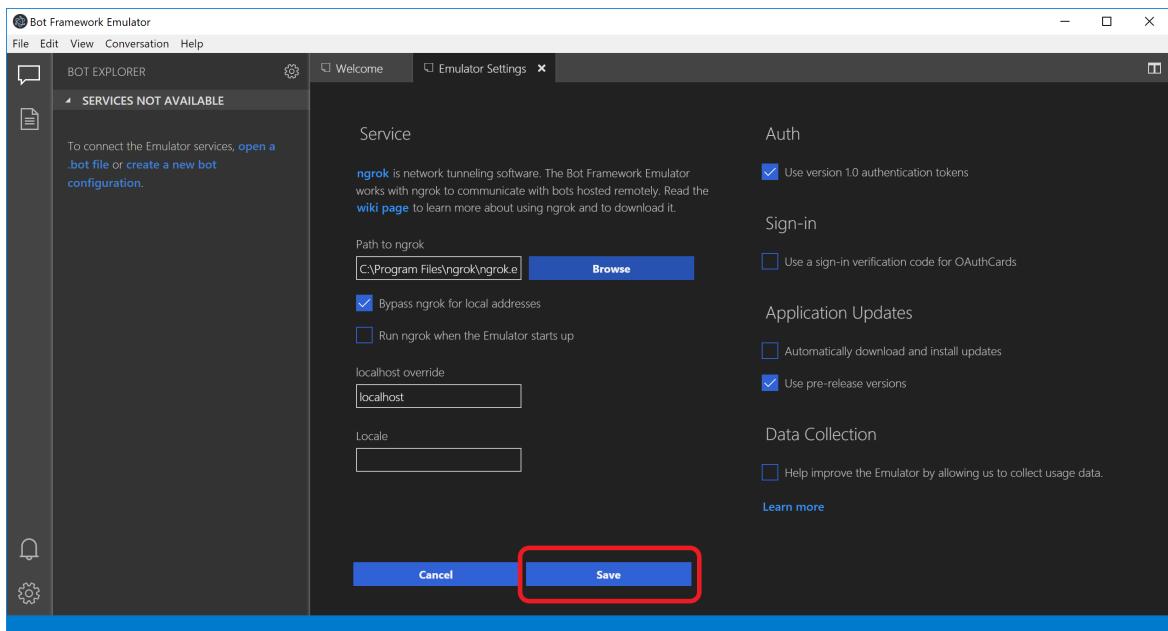
1. Перейдите на страницу параметров Эмулятора, нажав кнопку параметров (с изображением шестеренки) на панели навигации слева.



2. Снимите флагок *Help improve the Emulator by allowing us to collect usage data* (Помогите нам улучшить эмулятор, разрешив собирать данные об использовании) в разделе Data Collection (Сбор данных).



3. Нажмите кнопку "Сохранить".



Если вы передумаете, его можно снова включить, установив флажок.

## Дополнительные ресурсы

Эмулятор Bot Framework — это решение с открытым исходным кодом. Вы можете [принять участие](#) в разработке, [отправляя сведения об ошибках и предложениях](#).

Подробные сведения см. в статье об [устранении общих проблем](#) и других статья об устранении неполадок в этом разделе.

## Дальнейшие действия

Воспользуйтесь проверяющим ПО промежуточного слоя для отладки подключенного к каналу бота.

[Отладка бота с помощью файлов записей разговоров](#)

# Отладка ботов из любого канала с помощью ngrok

27.03.2021 • 4 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Когда программа-робот находится в разработке, вы можете выполнять [отладку Bot локально с помощью интегрированной среды разработки](#), например Visual Studio или Visual Studio Code. Вы также можете выполнить [отладку программы-робота с помощью эмулятора Bot Framework](#), поступающих в локальную среду и проверив сообщения, которые отправляет и получает Bot. Можно даже внести небольшие изменения в код программы Bot и включить функцию для [отладки робота с промежуточным по](#).

Когда Bot уже находится в рабочей среде, вы можете отлаживать программу Bot из любого [канала](#) с помощью [ngrok](#). Простое подключение программы Bot к нескольким каналам является ключевой функцией, доступной в среде Bot. В этой статье мы покажем, как локально выполнить отладку программы Bot с любого канала, для которого настроена Рабочая программа-робот, с помощью [ngrok](#). Мы использовали пример [еочобот](#), подключенный к [каналу Microsoft Teams](#) по всей статье, чтобы получить инструкции.

## Предварительные условия

- Подписка на [Microsoft Azure](#).
- Установите [ngrok](#).
- [Базовый робот](#), подключенный к любому [каналу](#).

## Запуск ngrok

[ngrok](#) — это кросс-платформенное приложение, которое позволяет предоставлять веб-сервер, работающий на локальном компьютере, к Интернету. В сущности, мы будем использовать [ngrok](#) для пересылки сообщений из внешних каналов в Интернете непосредственно на локальный компьютер, чтобы разрешить отладку, в отличие от стандартной конечной точки обмена сообщениями, настроенной в портал Azure.

1. Откройте терминал и перейдите в папку, в которой находится исполняемый файл [ngrok](#).
2. Запустите [ngrok](#) с помощью следующей команды, чтобы создать новый туннель.

```
ngrok http 3978 -host-header="localhost:3978"
```

### NOTE

Обратите внимание, что указанный порт является портом, на котором запущена программа-робот. Вы можете использовать любой порт localhost, который вы хотите.

3. Когда [ngrok](#) запустится, скопируйте и сохраните общедоступный URL-адрес пересылки для последующего использования.

```
ngrok by @inconshreveable

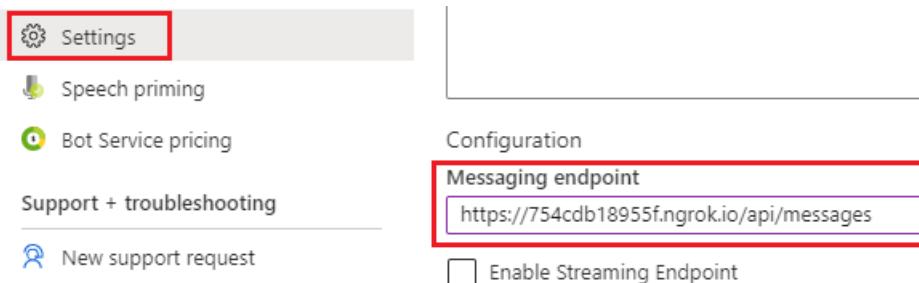
Session Status           online
Session Expires          7 hours, 59 minutes
Version                  2.3.35
Region                   United States (us)
Web Interface            http://127.0.0.1:4041
Forwarding               http://754cdb18955f.ngrok.io -> http://localhost:3978
Forwarding               https://754cdb18955f.ngrok.io -> http://localhost:3978

Connections              ttl     opn      rt1      rt5      p50      p90
                           0       0       0.00    0.00    0.00    0.00
```

## Настройка на портале Azure

Во время выполнения ngrok войдите в портал Azure и просмотрите параметры Bot, чтобы выполнить определенную настройку.

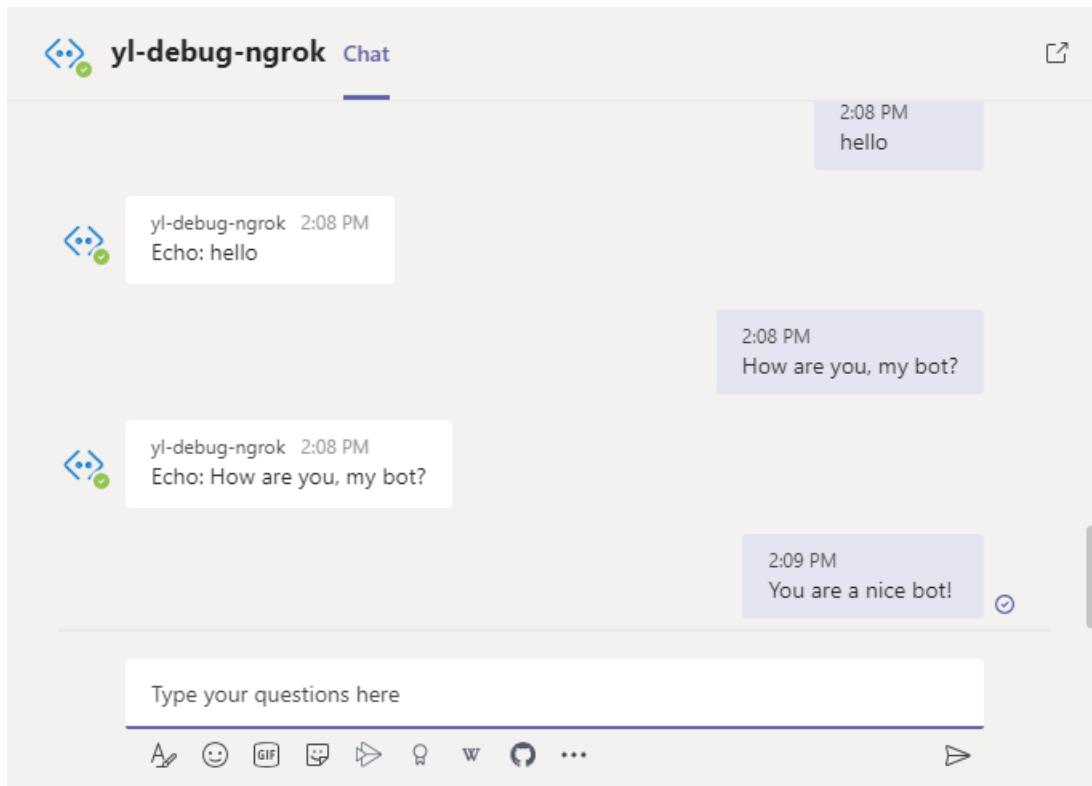
1. Выберите **регистрацию каналов Bot**, подключенных к локальному роботу.
2. Прокрутите вниз до пункта **Конфигурация**. Скопируйте и вставьте URL-адрес пересылки ngrok в поле **Конечная точка обмена сообщениями**. Убедитесь, что в конце URL-адреса вы сохраняете "/АПИ/мессажес".



3. Прокрутите экран вверх и выберите **сохранить**.

## Проверить

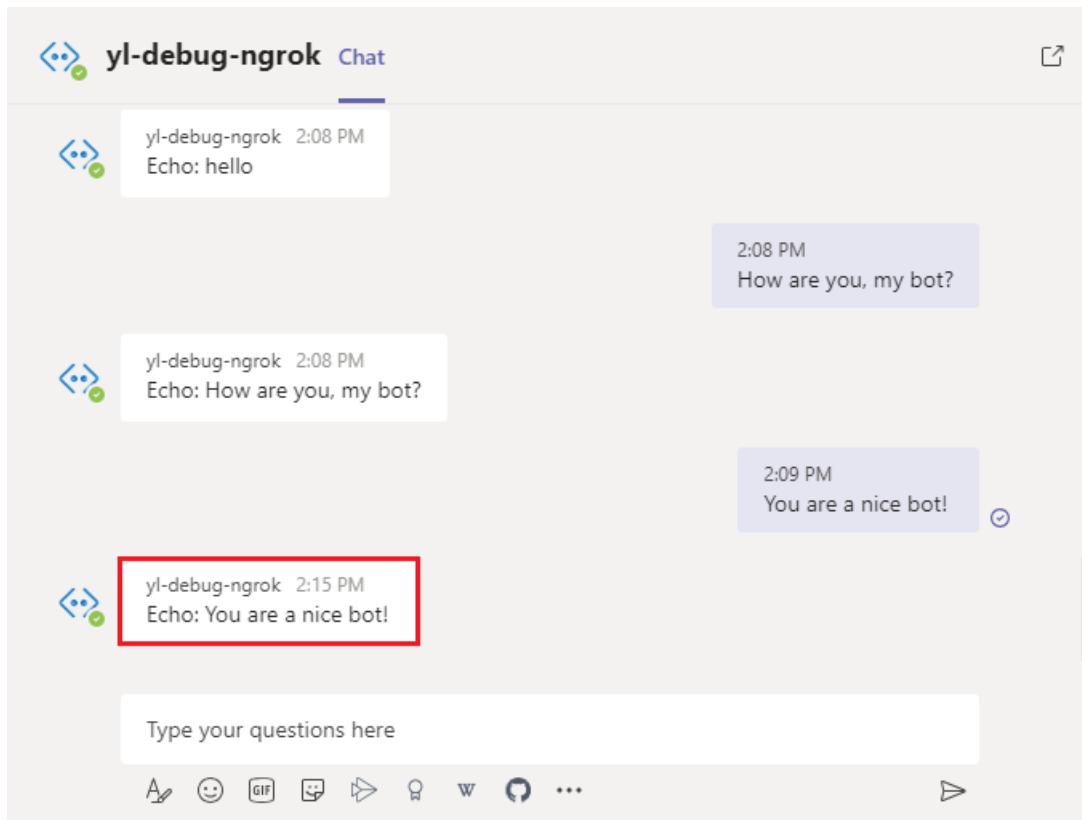
На этом этапе входящие сообщения от внешних каналов к роботу будут отправляться на ваш локальный робот. Пример программы-робота, который мы будем использовать для демонстрации, уже настроен в реальном времени для Microsoft Teams. Прочтите статью [Подключение программы-робота к Microsoft Teams](#) о подключении местного канала программы-робота к Microsoft Teams Channel.



Можно задать точки останова в Visual Studio локально. Развернув свойство Text из объекта входящих действий, вы увидите, что сообщение, отправленное программой Bot из команд, перехватывается локально для отладки.

```
10  namespace Microsoft.BotBuilderSamples.Bots
11  {
12      [assembly: Reference]
13      public class EchoBot : ActivityHandler
14      {
15          protected override async Task<Object> OnMessageActivityAsync(ITurnContext<Object> turnContext, CancellationToken cancellationToken)
16          {
17              var replyText = $"Echo: {turnContext.Activity.Text}";
18              await turnContext.SendActivityAsync(MessageFactory.Text(replyText), cancellationToken);
19          }
20      }
21      protected override async Task<Object> OnMembersAddedAsync(IList<ChannelAccount> membersAdded, ITurnContext<Object> turnContext, CancellationToken cancellationToken)
22      {
23          var welcomeText = "Hello and welcome!";
24          foreach (var member in membersAdded)
25          {
26              if (member.Id != turnContext.Activity.Recipient.Id)
27              {
28                  await turnContext.SendActivityAsync(MessageFactory.Text(welcomeText), cancellationToken);
29              }
30          }
31      }
32  }
```

Отсюда можно выполнять отладку в обычном режиме и выполнить код пошаговым образом. Его можно использовать для отладки программы Bot с любого канала.



## Дополнительные сведения

- [Подключение бота к каналам](#)

# Отладка навыка или потребителя навыка

27.03.2021 • 3 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Может потребоваться отладка потребительской ленты, на которой вы хотите выполнить отладку, которая выполняется локально, а другая выполняется в облаке. В этом случае можно использовать средство ngrok, чтобы предоставить локальную конечную точку Bot в качестве общедоступного URL-адреса.

## Предварительные условия

- Подписка на [Microsoft Azure](#).
- [ngrok](#) установлен.
- Установленный [эмулятор Bot-платформы](#).
- Знание [навыков](#), [Отладка робота](#), [Отладка Bot из любого канала с помощью ngroka](#) [Отладка с помощью эмулятора](#).
- Зарегистрированный навык и программы-роботы потребитель.

## Отладка местного потребителя навыков

В этом случае вам не нужно изменять конфигурацию развернутого навыка, и вы можете использовать эмулятор Bot Framework для непосредственного тестирования потребителя.

1. Настройте конечную точку туннелирование для локальной программы Bot и настройте ее регистрацию в портал Azure, как описано в разделе [Отладка Bot с любого канала с помощью ngrok](#).
2. Обновите конфигурацию приложения потребителя и задайте в качестве конечной точки сайта навыка URL-адрес, созданный ngrok.
3. Наконец, запустите потребитель в локальной среде и подключитесь к нему, как описано в разделе [Отладка с помощью эмулятора](#).

### TIP

Локальному потребителю потребуется действительный идентификатор приложения и пароль.

## Отладка местного робота по навыкам

При тестировании изменений в навыке может потребоваться выполнять и отлаживать его локально, когда доступ к нему осуществляется из потребительской ленты, размещенной на промежуточном сервере.

В этом случае необходимо изменить конфигурацию развернутого потребителя навыков. Так как вы будете использовать потребителя для тестирования программы-робота.

1. Настройте конечную точку туннелирование для локальной программы Bot и настройте ее регистрацию в портал Azure, как описано в разделе [Отладка Bot с любого канала с помощью ngrok](#).
2. Обновите конфигурацию приложения потребителя и задайте в качестве конечной точки квалификации URL-адрес, созданный ngrok. Вы можете изменить конфигурацию непосредственно на сервере, иначе, возможно, придется повторно развернуть потребителя навыков.

3. Запустите навык локально и вызовите его из потребителя навыков.

**TIP**

Для локального навыка потребуется действительный идентификатор приложения и пароль.

# Отладка бота с помощью проверяющего ПО промежуточного слоя

27.03.2021 • 15 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье описывается отладка программы-робота с помощью промежуточного слоя проверки. Эта функция позволяет Bot Framework Emulator отлаживать входящий и исходящий трафик для бота, а также просматривать текущее состояние бота. Можно использовать сообщение трассировки для отправки данных в эмулятор, а затем проверять состояние программы-робота в любом заданном состоянии диалога.

Чтобы продемонстрировать, как выполнять отладку и проверку состояния сообщений бота, мы используем бот EchoBot, созданный локально с помощью Bot Framework версии 4 ([C#](#) | [JavaScript](#) | [Python](#)). Отладку бота также можно выполнять с помощью [IDE](#) или [Bot Framework Emulator](#), но для отладки состояния в боте нужно добавить проверяющее ПО промежуточного слоя. Соответствующие примеры для бота доступны здесь: [C#](#), [JavaScript](#) и [Python](#).

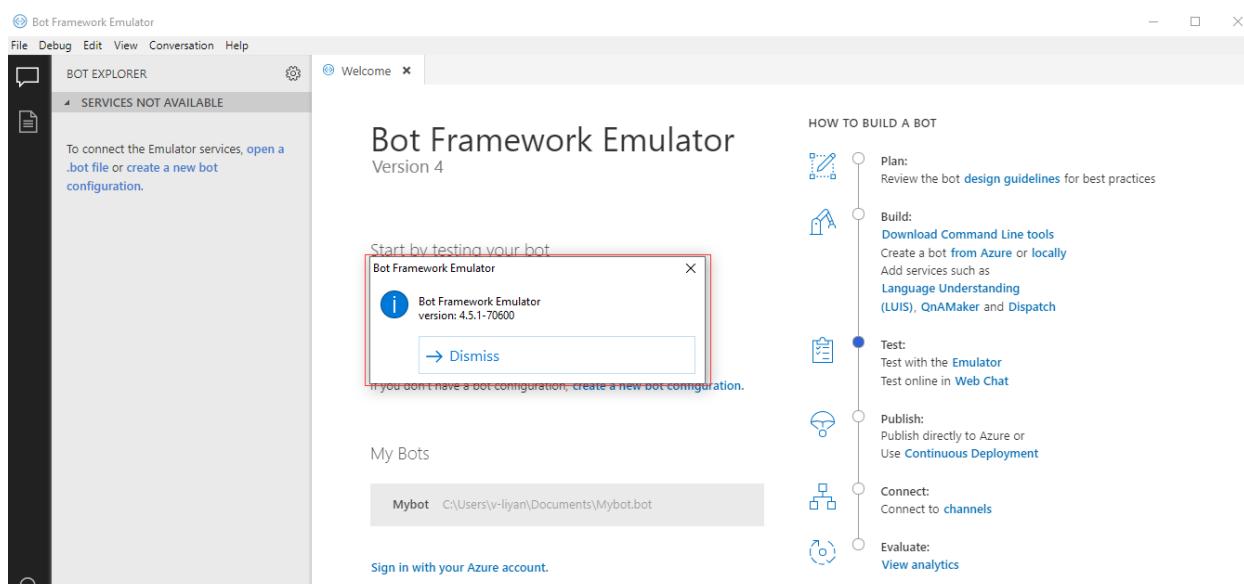
## Предварительные требования

- Установленное приложение [Bot Framework Emulator](#).
- Понимание принципов работы [промежуточного слоя](#) бота.
- Понимание принципов [управления состоянием](#) бота.
- Установленное средство [ngrok](#) (если вы хотите отладить бота, настроенного в Azure для использования дополнительных каналов).

## Обновление эмулятора до последней версии

Прежде чем использовать по промежуточного слоя для проверки Bot для отладки программы-робота, необходимо обновить эмулятор до версии 4,5 или более новой. Проверьте [последнюю версию](#) на наличие обновлений.

Чтобы проверить версию эмулятора, выберите в меню пункт **Справка > о программе**. Вы увидите текущую версию эмулятора.



## Обновление кода бота

- [C#](#)
- [JavaScript](#)
- [Python](#)

Настройте состояние проверки и добавьте проверяющее ПО промежуточного слоя в адаптер в файле `Startup.cs`. Состояние проверки предоставляется путем внедрения зависимостей. См. обновление кода ниже или пример проверки: [C#](#).

`Startup.cs`.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers().AddNewtonsoftJson();

    services.AddSingleton<IStorage, MemoryStorage>();

    // The Inspection Middleware needs some scratch state to keep track of the (logical) listening
    // connections.
    services.AddSingleton<InspectionState>();

    // You can inspect the UserState
    services.AddSingleton<UserState>();

    // You can inspect the ConversationState
    services.AddSingleton<ConversationState>();

    // Create the Bot Framework Adapter.
    services.AddSingleton<IBotFrameworkHttpAdapter, AdapterWithInspection>();

    // Create the bot as a transient. In this case the ASP Controller is expecting an IBot.
    services.AddTransient<IBot, EchoBot>();
}
```

`AdapterWithInspection.cs`

```

namespace Microsoft.BotBuilderSamples
{
    public class AdapterWithInspection : BotFrameworkHttpAdapter
    {
        public AdapterWithInspection(IConfiguration configuration, InspectionState inspectionState,
UserState userState, ConversationState conversationState, ILogger<BotFrameworkHttpAdapter> logger)
            : base(configuration, logger)
        {
            // Inspection needs credentials because it will be sending the Activities and User and
Conversation State to the emulator
            var credentials = new MicrosoftAppCredentials(configuration["MicrosoftAppId"],
configuration["MicrosoftAppPassword"]);

            Use(new InspectionMiddleware(inspectionState, userState, conversationState, credentials));

            OnTurnError = async (turnContext, exception) =>
            {
                // Log any leaked exception from the application.
                logger.LogError(exception, $"[OnTurnError] unhandled error : {exception.Message}");

                // Send a message to the user
                await turnContext.SendActivityAsync("The bot encountered an error or bug.");
                await turnContext.SendActivityAsync("To continue to run this bot, please fix the bot source
code.");

                // Send a trace activity, which will be displayed in the Bot Framework Emulator
                await turnContext.TraceActivityAsync("OnTurnError Trace", exception.Message,
"https://www.botframework.com/schemas/error", "TurnError");
            };
        }
    }
}

```

Обновите класс бота в файле EchoBot.cs.

**EchoBot.cs**

```

private readonly ConversationState _conversationState;
private readonly UserState _userState;

public EchoBot(ConversationState conversationState, UserState userState)
{
    _conversationState = conversationState;
    _userState = userState;
}

public override async Task OnTurnAsync(ITurnContext turnContext, CancellationToken cancellationToken =
default)
{
    await base.OnTurnAsync(turnContext, cancellationToken);

    await _conversationState.SaveChangesAsync(turnContext, false, cancellationToken);
    await _userState.SaveChangesAsync(turnContext, false, cancellationToken);
}

protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
 CancellationToken cancellationToken)
{
    var conversationStateProp = _conversationState.CreateProperty<CustomState>("customState");
    var convProp = await conversationStateProp.GetAsync(turnContext, () => new CustomState { Value = 0 },
    cancellationToken);

    var userStateProp = _userState.CreateProperty<CustomState>("customState");
    var userProp = await userStateProp.GetAsync(turnContext, () => new CustomState { Value = 0 },
    cancellationToken);

    await turnContext.SendActivityAsync(MessageFactory.Text($"Echo: {turnContext.Activity.Text} conversation
state: {convProp.Value} user state: {userProp.Value}"), cancellationToken);

    convProp.Value++;
    userProp.Value++;
}

```

## Локальное тестирование бота

После обновления кода можно запустить Бот локально и протестировать функцию отладки с помощью двух эмуляторов: один для отправки и получения сообщений, а другой — для проверки состояния сообщений в режиме отладки. Чтобы протестировать бота локально, сделайте следующее:

- Перейдите к каталогу бота в терминале и выполните следующую команду, чтобы запустить бота локально:

- C#
- JavaScript
- Python

```
dotnet run
```

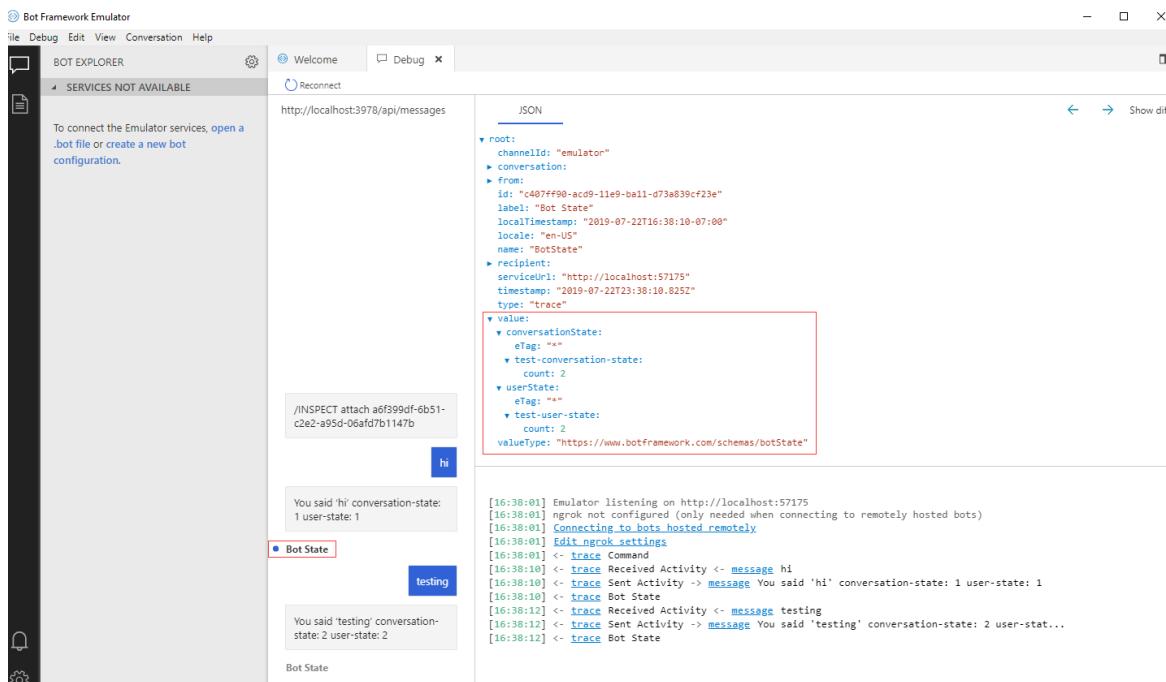
- Откройте эмулятор. Щелкните **Открыть бота**. Вставьте в поле "URL-адрес бота" `http://localhost:3978/api/messages`, а также значения **MicrosoftAppId** и **MicrosoftAppPassword**. Если вы используете бота на JavaScript, эти значения можно найти в файле `.env`. Если вы используете бота на C#, эти значения можно найти в файле `appsettings.json`. Нажмите кнопку **Соединить**.
- Теперь откройте другое окно эмулятора. Второе окно эмулятора будет работать в качестве отладчика. Следуйте инструкциям, описанным выше. Установите флажок **Открыть в режиме отладки** и щелкните **Подключить**.

4. На этом этапе вы увидите команду с уникальным идентификатором ( `/INSPECT attach <identifier>` ) в эмуляторе отладки. Скопируйте всю команду с идентификатором из эмулятора отладки и вставьте ее в окно чата первого эмулятора.

#### NOTE

Уникальный идентификатор создается каждый раз при запуске эмулятора в режиме отладки после добавления промежуточного слоя проверки в код программы-робота.

5. Теперь вы можете отправить сообщения в поле разговора первого эмулятора и изучить сообщения в эмуляторе отладки. Чтобы проверить состояние сообщений, щелкните **состояние бота** в эмуляторе отладки и **значения unfold** в правом окне JSON . Состояние программы-робота вы увидите в эмуляторе отладки:



## Проверка состояния бота, настроенного в Azure

Чтобы проверить состояние бота, настроенного в Azure и подключенного к каналам (например, Teams), необходимо установить и запустить [ngrok](#).

### Запуск ngrok

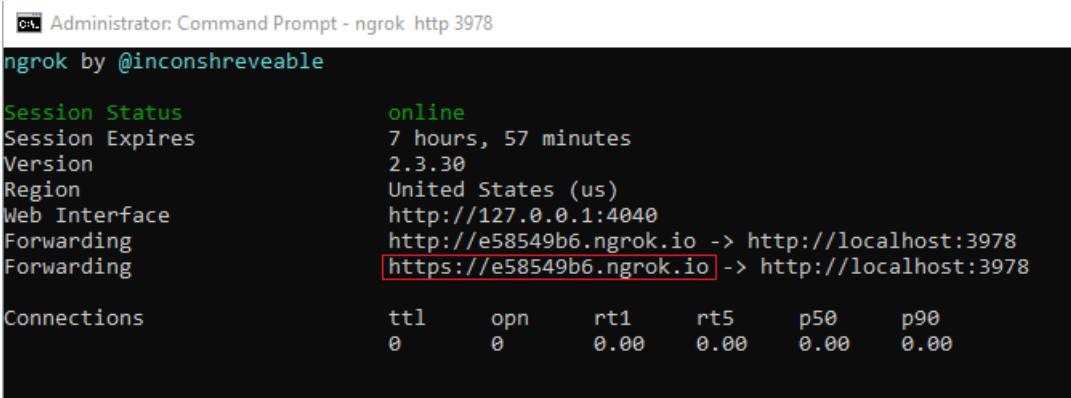
На этом этапе вы обновили эмулятор до последней версии и добавили промежуточное по для проверки в код программы-робота. Теперь вы можете запустить ngrok и подготовить локального бота к работе с Регистрацией каналов Azure Bot. Перед запуском ngrok нужно запустить бота локально.

Чтобы запустить бота локально, сделайте следующее:

- Перейдите в папку бота в терминале и настройте для регистрации прм использование [последних сборок](#).
- Запустите бот на локальном компьютере. Вы увидите, что программа-робот предоставляет номер порта, например `3978` .
- Откройте другое окно командной строки и перейдите в папку проекта бота. Выполните следующую команду:

```
ngrok http 3978
```

4. Это подключит ngrok к запущенному локально боту. Скопируйте общедоступный IP-адрес.



```
Administrator: Command Prompt - ngrok http 3978
ngrok by @inconshreveable

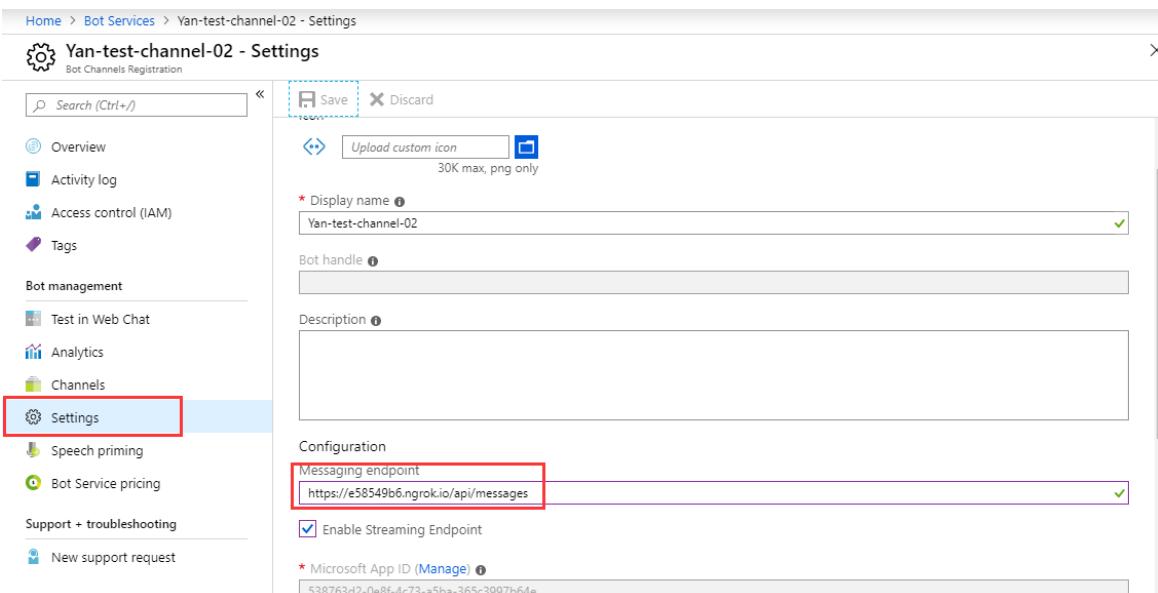
Session Status          online
Session Expires        7 hours, 57 minutes
Version                2.3.30
Region                 United States (us)
Web Interface          http://127.0.0.1:4040
Forwarding             http://e58549b6.ngrok.io -> http://localhost:3978
Forwarding             https://e58549b6.ngrok.io -> http://localhost:3978

Connections            ttl     opn      rt1      rt5      p50      p90
                        0       0       0.00    0.00    0.00    0.00
```

### Обновление Регистраций каналов для бота

Локальный бот подключен к ngrok, и вы можете подготовить его к работе с Регистрацией каналов бота в Azure.

1. Перейдите к Регистрации каналов бота в Azure. В меню слева щелкните **Параметры** и задайте для **конечной точки обмена сообщениями** значение с IP-адресом ngrok. При необходимости добавьте /api/messages после IP-адреса. Например, <https://e58549b6.ngrok.io/api/messages>). Установите флагок **Включить конечную точку потоковой передачи** и щелкните **Сохранить**.



#### TIP

Если кнопка **Сохранить** неактивна, снимите флагок **Включить конечную точку потоковой передачи** и снова щелкните **Сохранить**. Затем установите флагок **Включить конечную точку потоковой передачи** и снова щелкните **Сохранить**. Убедитесь, что флагок **Включить конечную точку потоковой передачи** установлен, а конфигурация конечной точки сохранена.

2. Перейдите в группу ресурсов бота, щелкните **Развертывание** и выберите Регистрацию каналов бота (предыдущее успешное развертывание). Щелкните **Входные данные** слева, чтобы получить значения appId и appSecret. Добавьте в файл .env бота (или файл appsettings.json в случае использования бота на C#) значения appId и appSecret.

## Yan-test-channel-02\_979 - Inputs

Deployment

<input type="text" value="Search (Ctrl+ /)"/>	<input type="button" value="&lt;&lt;"/>
<input checked="" type="checkbox"/> Overview	<input type="button" value=""/>
<input checked="" type="checkbox"/> Inputs	<input type="button" value=""/>
<input type="checkbox"/> Outputs	<input type="button" value=""/>
<input type="checkbox"/> Template	<input type="button" value=""/>
botEnv	<input type="text" value="prod"/> <input type="button" value=""/>
botId	<input type="text" value="Yan-test-channel-02"/> <input type="button" value=""/>
location	<input type="text" value="westus2"/> <input type="button" value=""/>
sku	<input type="text" value="S1"/> <input type="button" value=""/>
kind	<input type="text" value="bot"/> <input type="button" value=""/>
siteName	<input type="text" value="nosite"/> <input type="button" value=""/>
createNewStorage	<input type="text" value="false"/> <input type="button" value=""/>
storageAccountName	<input type="text" value="luis1aks1gzsps"/> <input type="button" value=""/>
storageAccountLocation	<input type="text"/> <input type="button" value=""/>
storageAccountResourceId	<input type="text"/> <input type="button" value=""/>
appId	<input type="text" value="33374300-0ef3-4372-a5fa-36312097b34e"/> <input type="button" value=""/>
appSecret	<input type="text" value="y03nqfWmB0ce27704cpU32f5"/> <input type="button" value=""/>
azureWebJobsBotFrameworkDirectLineSecret	<input type="text"/> <input type="button" value=""/>

3. Запустите эмулятор, щелкните **Открыть Bot** и вставьте `http://localhost:3978/api/messages URL-адрес Bot`. Вставьте в поля **Идентификатор приложения Майкрософт** и **Пароль приложения Майкрософт** те же значения appId и appSecret, которые вы добавили в файл `.env` (`appsettings.json`) бота. Щелкните **Подключить**.
4. Теперь запущенный бот подключен к Регистрации каналов бота в Azure. Чтобы протестировать веб-чат, щелкните **Тестируйте в веб-чате** и отправьте сообщения в поле чата.

Yan-test-channel-02 - Test in Web Chat

Bot Channels Registration

Search (Ctrl+ /)

Test

Start over

Overview

Activity log

Access control (IAM)

Tags

Bot management

Test in Web Chat

Analytics

Channels

Settings

Speech priming

Bot Service pricing

Support + troubleshooting

New support request

hi

You

Hello and welcome!

Yan-test-channel-02

You said 'hi' conversation-state: 0 user-state: 0

Yan-test-channel-02

hello

You

You said 'hello' conversation-state: 1 user-state: 1

Yan-test-channel-02

inspection middleware testing

You

You said 'inspection middleware testing' conversation-state: 2 user-state: 2

Yan-test-channel-02 at 4:49:31 PM

Type your message...

Microphone icon

5. Теперь давайте активируем режим отладки в эмуляторе. В эмуляторе выберите **Отладка > начать отладку**. Введите IP-адрес ngrok (не забудьте добавить **/АПИ/мессажес**) для URL-адреса Bot (например, <https://e58549b6.ngrok.io/api/messages>). В качестве **идентификатора приложения Microsoft** введите идентификатор приложения для программы-робота. В качестве **пароля приложения Microsoft** введите секрет приложения Bot. Также убедитесь, что флагок **Открыть в режиме отладки** установлен. Нажмите кнопку **Соединить**.
6. При включении режима отладки UUID будет создан в эмуляторе. UUID — это уникальный идентификатор, создаваемый каждый раз при запуске режима отладки в эмуляторе. Скопируйте и вставьте UUID в поле чата **Тестируовать в веб-чате** или в поле чата канала. В поле чата вы увидите сообщение Attached to session, all traffic is being replicated for inspection (Подключение к сеансу; весь трафик реплицируется для проверки).

Чтобы начать отладку бота, отправьте сообщения в настроенное поле чата канала. Локальный эмулятор будет автоматически обновлять сообщения со всеми подробностями для отладки. Чтобы проверить состояние сообщений бота, щелкните **Состояние бота** и разверните блок **values** в окне JSON справа.

The screenshot shows a bot interface with two messages from a user and their corresponding responses. The first message is 'hi' with a timestamp of [16:51:12]. The second message is 'testing inspection middleware' with a timestamp of [16:55:04]. Below the messages is a log of bot activity.

```
[16:51:12] Emulator listening on http://localhost:57512...
[16:51:12] ngrok not configured (only needed when connecting to remotely hosted bots)
[16:51:12] Connecting to bots hosted remotely
[16:51:12] Edit ngrok settings
[16:51:12] Error: The bot is remote, but the service URL is localhost. Without tunneling software you will not receive replies.
[16:51:12] Connecting to bots hosted remotely
[16:51:12] Configure ngrok
[16:51:13] <- trace Command
[16:55:04] <- trace Received Activity <- message hi
[16:55:04] <- trace Sent Activity -> message You said 'hi' conversation-state: 1 user-state: 1
[16:55:05] <- trace Bot State
[16:55:10] <- trace Received Activity <- message testing inspection middleware
[16:55:10] <- trace Sent Activity -> message You said 'testing inspection middleware' conversat...
[16:55:10] <- trace Bot State
```

## Дополнительные ресурсы

- Попробуйте пример послойного слоя-образца на [C#](#), [JavaScript](#) или [Python](#).
- См. также статьи об [устранении общих проблем](#) и других неполадок в этом разделе.
- См. также статью об [отладке ботов с помощью эмулятора](#).

# Отладка бота с помощью файлов записей разговоров

27.03.2021 • 7 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Один из залогов успеха в тестировании и отладке бота — возможность записать и просмотреть набор условий, возникающих при запуске бота. В этой статье описывается, как создать и использовать файл записи разговора с ботом, чтобы предоставить детальный набор действий пользователя и ответов бота для тестирования и отладки.

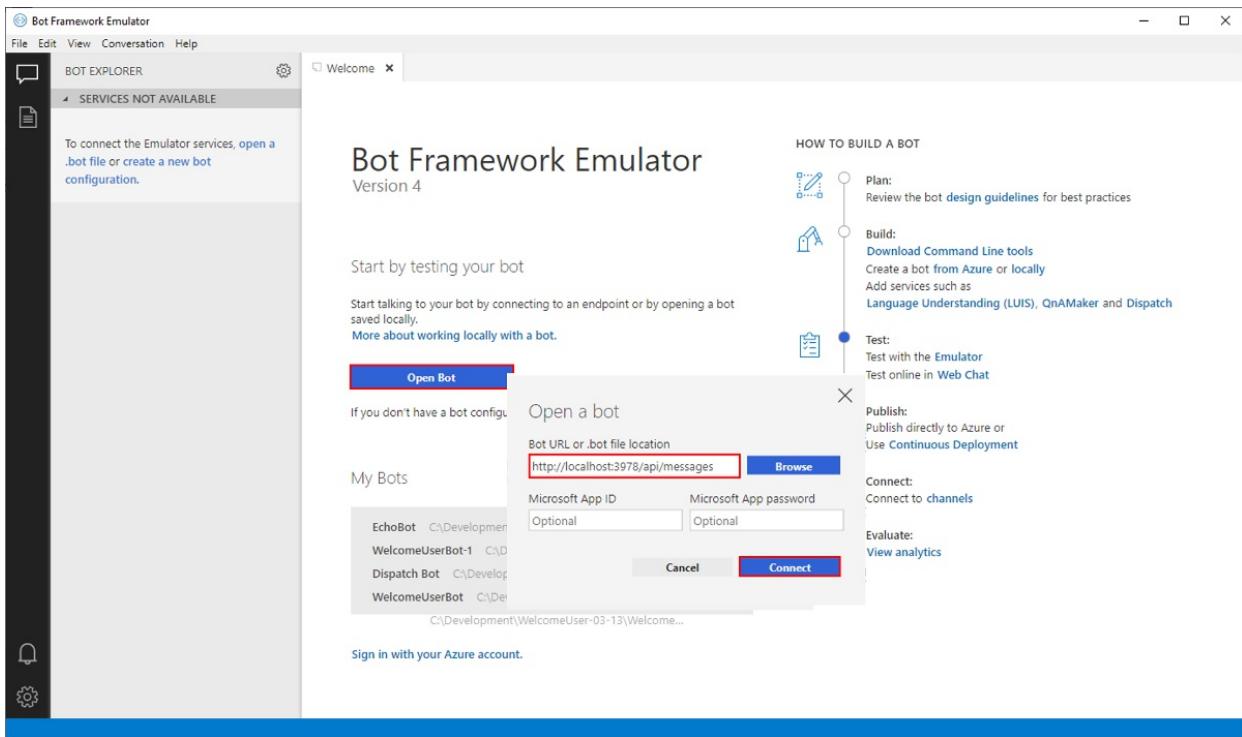
## Файл записи разговора с ботом

Файл записи разговора с ботом — это специализированный JSON-файл, в котором хранятся данные о взаимодействии между пользователем и ботом. В нем сохраняется не только содержимое сообщения, но и сведения о взаимодействии, например идентификаторы пользователя и канала, тип и возможности канала, время взаимодействия и т. д. Затем всю эту информацию можно использовать для поиска и устранения проблем при тестировании или отладке бота.

## Создание и сохранение файла записи разговора с ботом

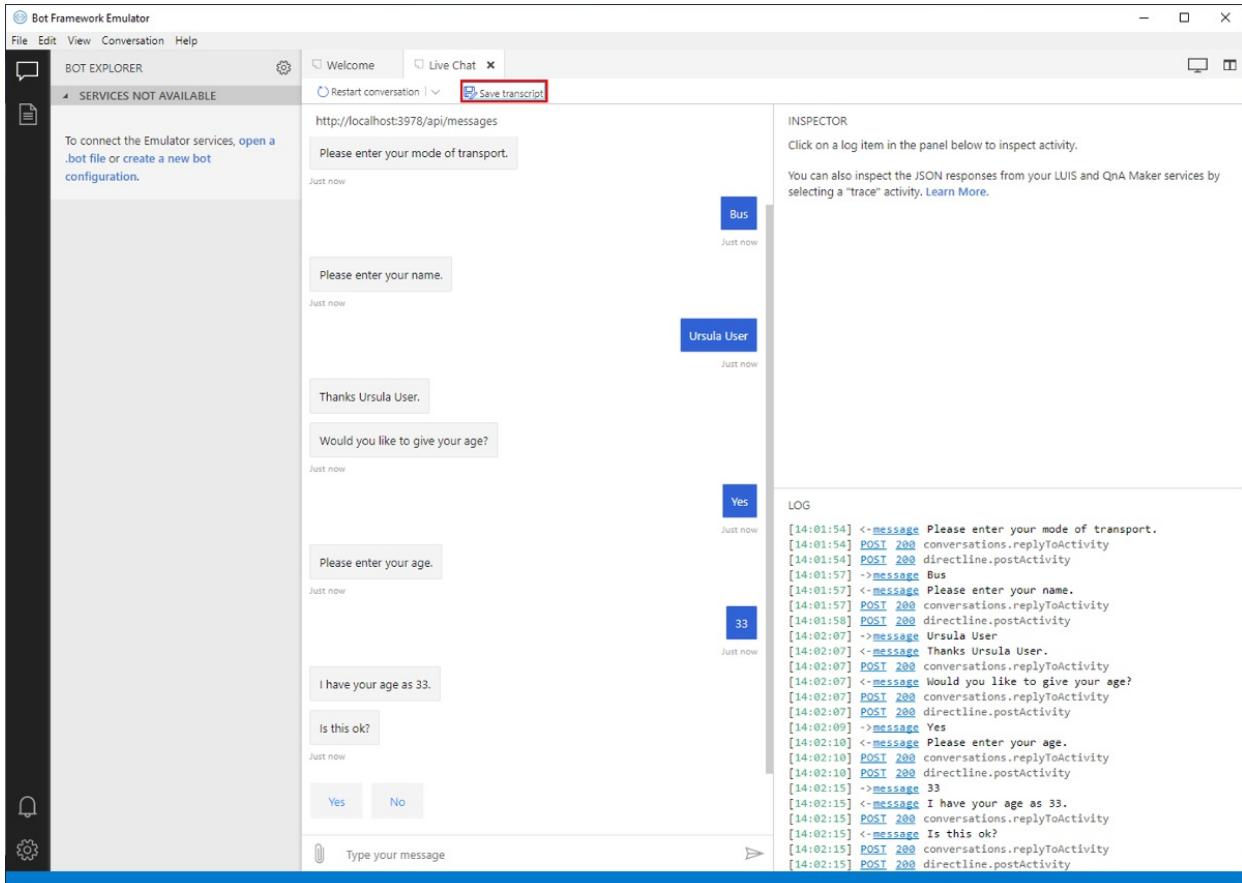
В этой статье показано, как создавать файлы записей разговоров с ботами при помощи средства [Bot Framework Emulator](#) от Майкрософта. Файлы транскрипции также могут быть созданы программно; Дополнительные сведения об этом подходе см. в статье [хранение записей BLOB-объектов](#). В этой статье мы будем использовать пример кода [бота с несколькими шагами запросов](#) для Bot Framework, который спрашивает у пользователя имя, возраст и вид используемого им транспорта, но для создания файла с расшифровкой разговора можно использовать любой код, доступный через средство Bot Framework Emulator корпорации Майкрософт.

Чтобы начать этот процесс, убедитесь, что тестируемый код бота выполняется в вашей среде разработки. Запустите эмулятор Bot Framework, нажмите кнопку *Open Bot* (*открыть программу-робот*), а затем введите адрес `/localhost:порт`, показанный в браузере, а затем — "/API/messages", как показано на рисунке ниже. Теперь нажмите кнопку *подключить*, чтобы подключить эмулятор к роботу.

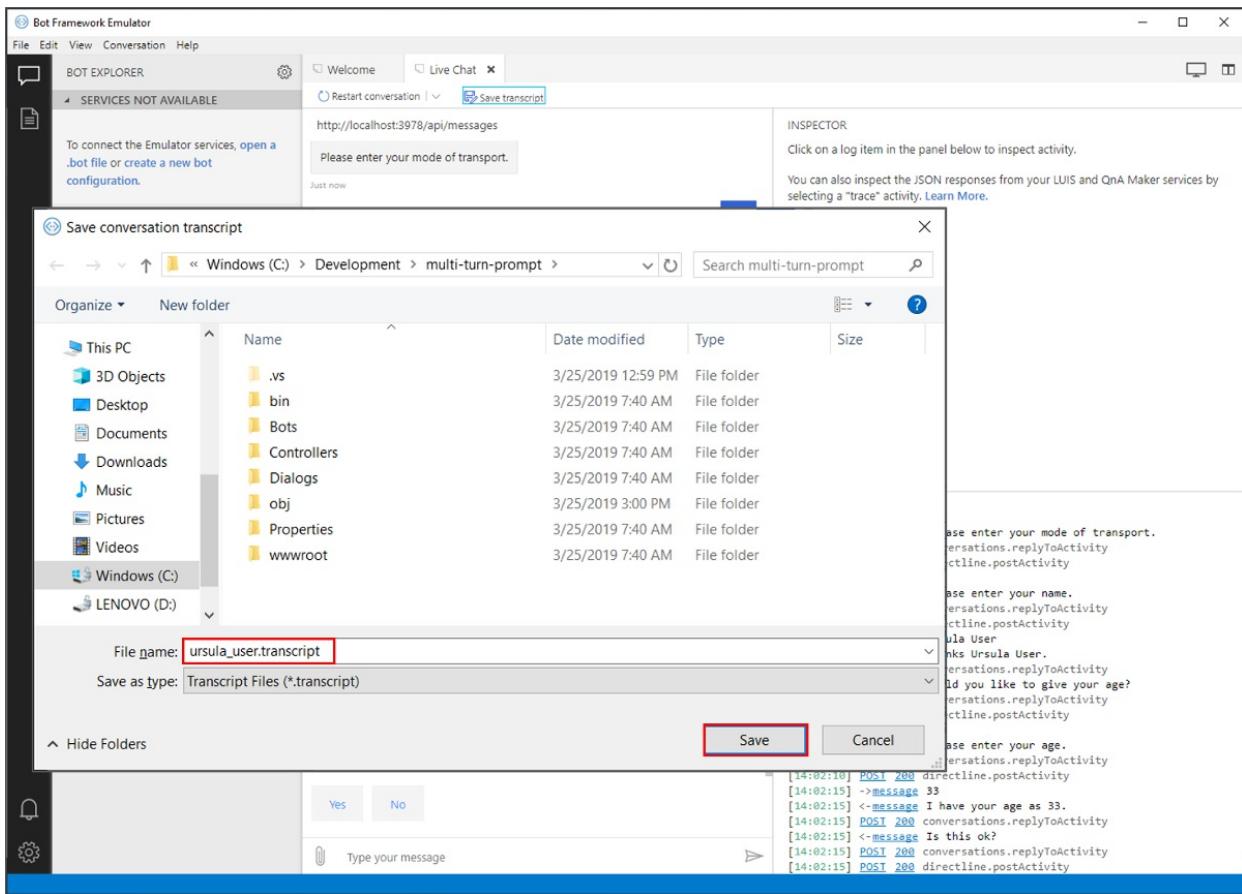


После подключения эмулятора к выполняемому коду Протестируйте код, отправив имитацию взаимодействия пользователя с Bot. В этом примере мы передаем сведения о виде транспорта, имени и возрасте пользователя. После введения всех действий пользователя, которые необходимо сохранить, используйте эмулятор Bot Framework, чтобы создать и сохранить файл транскрипции, содержащий этот диалог.

На вкладке *Live Chat* (показанной ниже) нажмите кнопку *Save transcript*.



Выберите расположение и имя для файла записи разговора и нажмите кнопку сохранения.

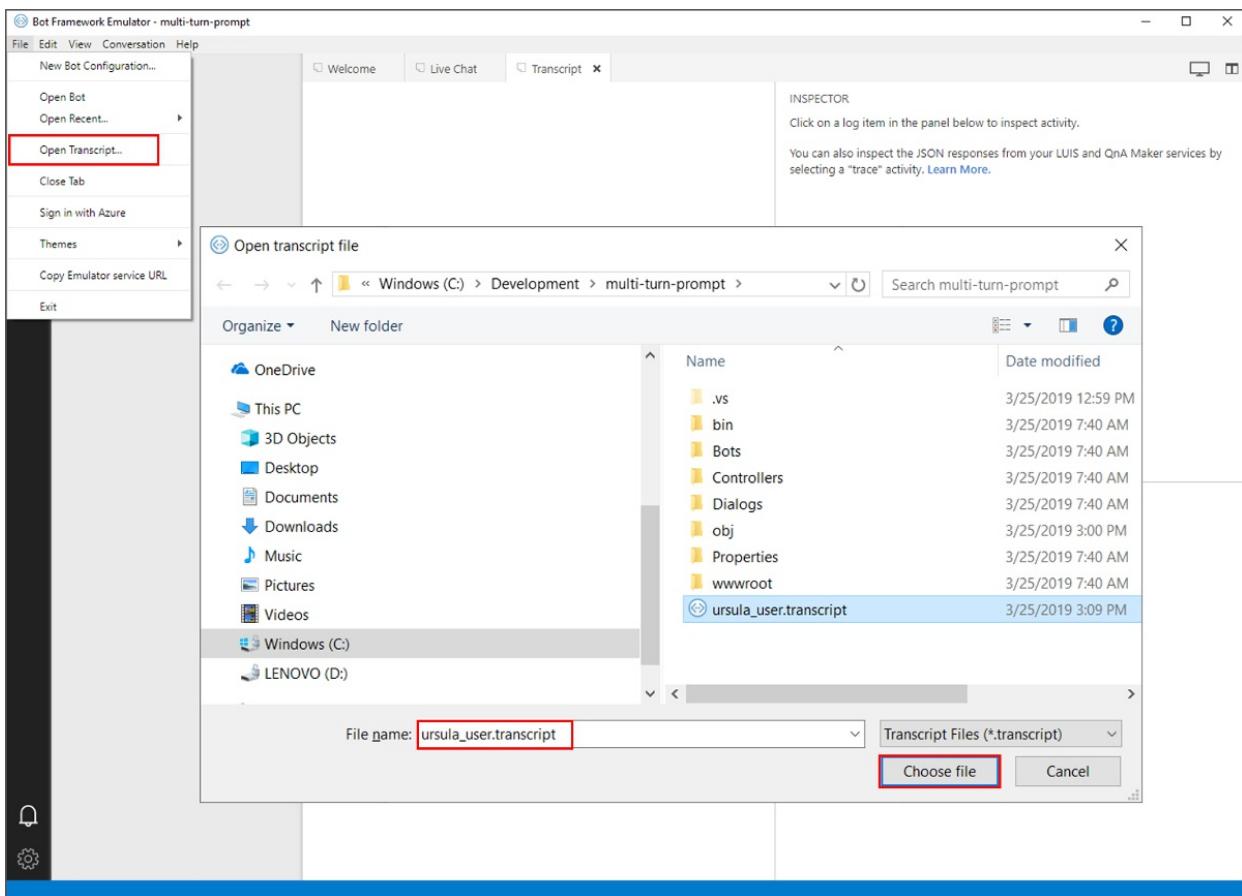


Все взаимодействия пользователей и ленты, введенные для тестирования кода с помощью эмулятора, теперь сохранены в файле транскрипции, который можно впоследствии перезагрузить, чтобы упростить отладку взаимодействия между пользователем и программой-роботом.

## Получение файла записи разговора с ботом

Чтобы получить файл записи программы-робота с помощью эмулятора Bot Framework, выберите *файл*, а затем *откройте запись разговора...* в левом верхнем углу эмулятора, как показано ниже. Затем выберите нужный файл записи разговора. (Записи можно также получить из элемента *управления "список записей"* в разделе *"ресурсы"* эмулятора).

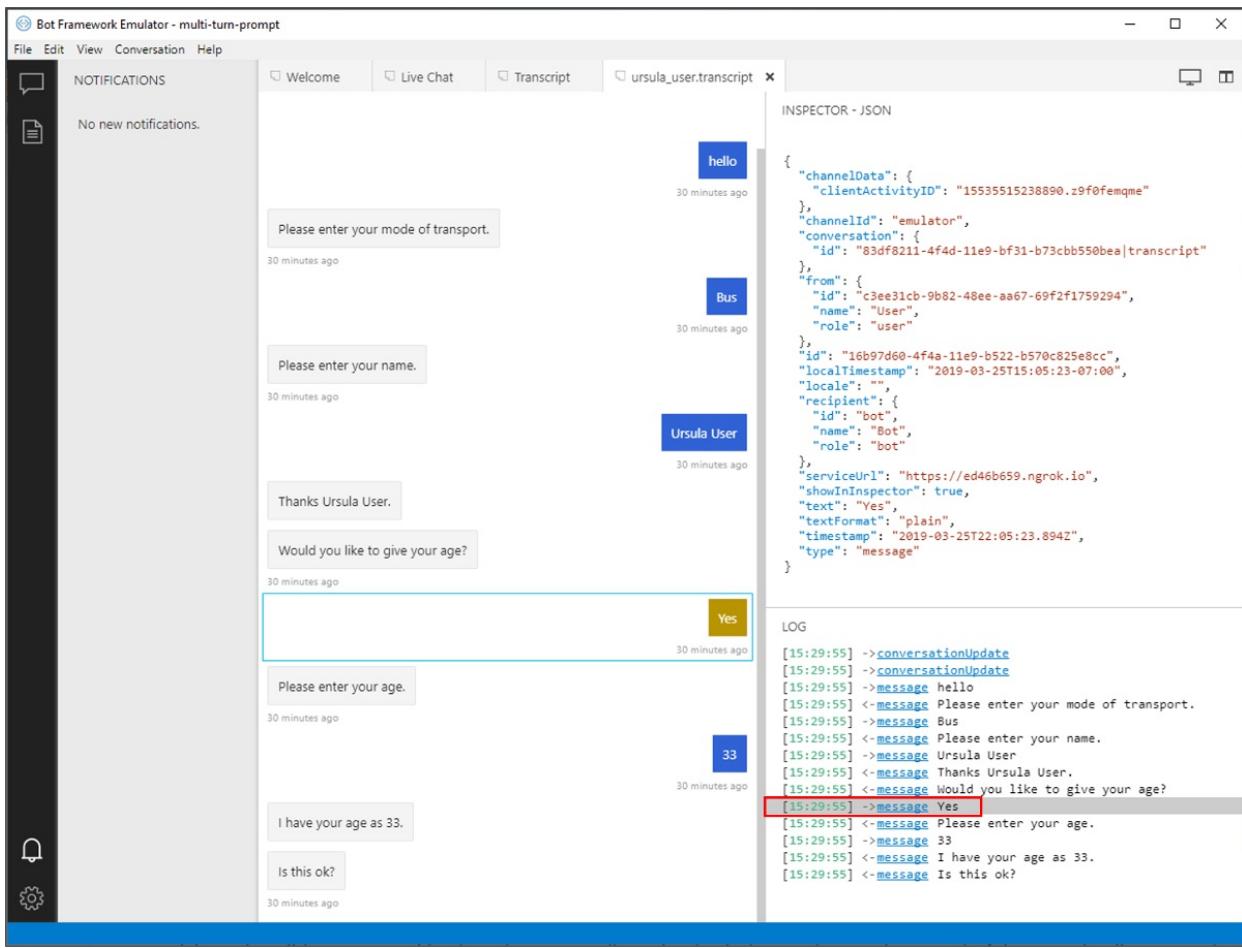
В этом примере мы получаем файл расшифровки с именем ursula\_user.transcript. При выборе *файла транскрипции* вся сохраненная беседа будет автоматически загружена в новую вкладку с заголовком *запись*.



## Отладка с помощью файла записи разговора

Теперь, когда файл записи разговора загружен, все готово к отладке взаимодействия между пользователем и ботом. Для этого просто щелкните любое событие или действие, записанное в разделе журнала, показанном в правой нижней области эмулятора. В приведенном ниже примере мы выбрали первое действие пользователя — отправку сообщения "Hello". При этом вся информация в файле транскрипции, относящаяся к конкретному взаимодействию, отображается в окне инспектора эмулятора в формате JSON. Просмотрев некоторые из этих значений внизу, мы видим:

- что действие имеет тип *message* (сообщение);
- время отправки сообщения;
- что отправленный обычный текст содержит строку *Yes*;
- что сообщение было отправлено нашему боту;
- идентификатор пользователя и сведения о нем;
- идентификатор канала, его возможности и сведения о нем.



Этот подробный уровень информации позволяет поэтапно отслеживать взаимодействие между пользователем и ботом. Это полезно для отладки, если бот либо ответил ненадлежащим образом, либо не ответил вообще. Задав оба этих значения и записав действия, предшествовавшие неудачному взаимодействию, вы можете переходить по коду, найти место, где бот не отвечает должным образом, и устранить эти проблемы.

Использование файлов записей разговоров вместе с Bot Framework Emulator — это лишь один из многих способов тестирования и отладки кода бота и взаимодействия с пользователями. Другие способы тестирования и отладки ботов описываются в перечисленных ниже ресурсах.

## Дополнительные сведения

Дополнительные сведения о тестировании и отладке:

- [Рекомендации по тестированию и отладке ботов](#)
- [Отладка ботов с помощью Bot Framework Emulator](#)
- [Сведения об устранении общих проблем и другие статьи об устранении неполадок в этом разделе.](#)
- [Отладка в Visual Studio](#)

# Отладка с помощью адаптивных средств

27.03.2021 • 2 minutes to read • [Edit Online](#)

Адаптивные инструменты платформы Bot — это расширение VS Code, помогающее разработчикам эффективно управлять файлами. LG., lu и диалогового окна (. Dialog).

Адаптивные инструменты имеют разнообразные средства и параметры, упрощающие отладку, анализ и расширение языковых файлов. Такие функции, как выделение синтаксиса, проверки диагностики и отладка, позволяют разработчикам устранять проблемы с языковыми файлами, в то время как автозаполнение и предложение улучшают и упрощают процесс создания ресурсов Bot.

## Предварительные требования

- Установите [Visual Studio Code](#).
- Установите [эмулятор Bot Framework](#).
- Программа-робот с одним или несколькими из следующих типов файлов:
  - [.LG](#)
  - [.lu](#)
  - диалоговое окно (. Dialog). Примеры файлов. Dialog можно найти в образце [адаптивбот](#) .

## О адаптивных средствах

В [файле readme по адаптивным инструментам](#) содержится вся информация, необходимая для установки и использования адаптивных средств.

ФАЙЛ сведений содержит следующие сведения:

- [Установка](#) адаптивных средств.
- [Функции языка](#), такие как выделение синтаксиса и автозавершение.
- [Наведите указатель мыши, предложения и навигацию по шаблонам](#), которые упрощают навигацию и расширение файлов.
- [Отладка](#) с помощью эмулятора.
- [Пользовательские параметры](#) в VS Code.

## Дополнительные сведения

- [Формат файлов LG](#)
- [Формат файлов LU](#)
- См. раздел сведения об [устранении неполадок](#) в разделах, посвященных устранению неполадок.

# Развертывание бота

27.03.2021 • 22 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье описано, как развернуть простого бота в Azure. Мы объясним, как подготовить бота к развертыванию, развернуть его в Azure и протестировать в Web Chat. Эту статью полезно изучить до выполнения описанных действий, чтобы ознакомиться со всеми процессами, связанными с развертыванием бота.

## IMPORTANT

Убедитесь, что используется последняя версия [Azure CLI](#). Если вы используете версию Azure CLI, предшествующую версии [2.2.0](#), возникнет ошибка из-за нерекомендуемых команд CLI. Кроме того, развертывание с помощью Azure CLI, описанное в этой статье, не стоит сочетать с развертыванием на портале Azure.

## Предварительные требования

- Подписка на [Microsoft Azure](#).
- Бот C#, JavaScript, TypeScript или Python, который разработан на локальном компьютере.
- Последняя версия [Azure CLI](#).
- Навыки работы с [Azure CLI и шаблонами ARM](#).

## Подготовка к развертыванию

- [C#](#)
- [JavaScript/TypeScript](#)
- [Python](#)

Если вы развертываете бот на C#, убедитесь, что он [создан в режиме выпуска](#). В Visual Studio для конфигурации решения укажите значение `Release`. Затем для решения выполните повторную сборку с очисткой, прежде чем продолжить. Развёртывание может завершиться ошибкой, если для конфигурации решения задано значение **Отладка**.

При создании программы-робота с помощью [шаблона Visual Studio](#) созданный исходный код включает в себя `DeploymentTemplates` папку, содержащую шаблоны ARM. В описанном здесь процессе развертывания используется один из шаблонов ARM для подготовки необходимых для бота ресурсов Azure с помощью Azure CLI.

С появлением пакета SDK Bot Framework 4.3 *не рекомендуется* использовать файл `.bot`. Вместо этого мы используем `appsettings.json` файл конфигурации для управления ресурсами Bot. Сведения о переносе параметров из файла Bot в файл конфигурации см. в разделе [Управление ресурсами Bot](#).

#### NOTE

Пакет [VSIX](#) включает версии .NET Core 2.1 и .NET Core 3.1 шаблонов C#. При создании ботов в Visual Studio 2019 следует использовать шаблоны .NET Core 3.1. В текущих примерах ботов используются шаблоны .NET Core 3.1. Примеры, использующие шаблоны .NET Core 2.1, можно найти в ветви [4.7-archive](#) репозитория BotBuilder-Samples. Сведения о развертывании .NET Core 3.1 программы-роботы в Azure см. в статье [развертывание программы Bot в Azure](#).

## 1. Вход в Azure

Созданный и протестированный локально бот можно развернуть в Azure. Откройте командную строку, чтобы войти на портал Azure.

```
az login
```

Она откроет окно браузера с интерфейсом для входа.

#### NOTE

При развертывании бота в облаке, отличном от Azure, например US Gov, необходимо выполнить `az cloud set --name <name-of-cloud>` перед `az login`, где `<name-of-cloud>` обозначает имя зарегистрированного облака, например `AzureUSGovernment`. Если вы хотите вернуться в общедоступное облако, можно запустить `az cloud set --name AzureCloud`.

## 2. Настройка подписки

Укажите подписку, которая будет использоваться по умолчанию.

```
az account set --subscription "<azure-subscription>"
```

Если вы не уверены, какую подписку выбрать для развертывания бота, просмотрите список подписок в учетной записи с помощью команды `az account list`.

## 3. Создание регистрации приложения

На этом шаге вы создадите регистрацию приложения Azure, что обеспечит следующие возможности.

- Взаимодействие пользователя с ботом через несколько каналов, например *Web Chat*.
- Определение *параметров подключения OAuth* для проверки подлинности пользователя и создания *маркера*, с помощью которого бот будет обращаться к защищенным ресурсам от имени пользователя.

### 3.1 Создание регистрации приложения Azure

Чтобы создать регистрацию приложения Azure, выполните приведенную ниже команду.

```
az ad app create --display-name "displayName" --password "AtLeastSixteenCharacters_0" --available-to-other-tenants
```

ПАРАМЕТР	ОПИСАНИЕ
display-name	Отображаемое имя приложения. Оно указывается на портале Azure в общем списке ресурсов и в той группе ресурсов, в которую включено это приложение.

ПАРАМЕТР	ОПИСАНИЕ
password	Пароль для приложения, также известный как <b>секрет клиента</b> . Это тот пароль, который вы создали для этого ресурса. Он должен содержать не менее 16 символов с минимум одной буквой в верхнем или нижнем регистре и минимум одним специальным символом.
available-to-other-tenants	Указывает, что приложение может использоваться из любого клиента Azure AD. Включите этот параметр, чтобы разрешить боту работать с каналами службы Azure Bot.

### 3.2 Запись значений appId и appSecret

После выполнения команды на предыдущем шаге (3.1) необходимо будет записать два значения:

- `password`, созданное на предыдущем шаге;
- `appId`, которое можно найти в выходных данных JSON.

Скопируйте и сохраните значения `appId` и `password`. Они потребуются на шаге развертывания ARM для присвоения значений параметрам `appId` и `appSecret` соответственно.

## 4. Создание службы приложения бота

При создании службы приложения бота можно развернуть бот в новой или существующей группе ресурсов. Для этого можно использовать [шаблон Azure Resource Manager \(ARM\)](#). Шаблон ARM — это JSON-файл, который декларативно определяет один или несколько ресурсов Azure и устанавливает зависимости между развернутыми ресурсами. Убедитесь, что у вас есть правильный путь к каталогу шаблонов развертывания ARM проекта `DeploymentTemplates`. Он нужен для присваивания значения файлу шаблона. Выберите любой вариант, который вам подходит.

- [Развертывание с помощью шаблона ARM в новой группе ресурсов](#).
- [Развертывание с помощью шаблона ARM в существующей группе ресурсов](#).

#### IMPORTANT

Боты Python нельзя развертывать в группе ресурсов, содержащей службы или боты Windows. Несколько программы-роботы Python можно развернуть в одной группе ресурсов, но в другой группе ресурсов необходимо создать другие службы (LUIS, QnA и т. д.).

#### Развертывание с помощью шаблона ARM (в новой группе ресурсов)

На этом шаге вы создадите службу приложения для бота, то есть определите этап развертывания для бота. Вы примените для этого шаблон ARM, создадите план обслуживания и группу ресурсов. Выполните следующую команду Azure CLI, чтобы запустить развертывание в области подписки из локального файла шаблона.

#### TIP

Используйте шаблон ARM для *новой* группы ресурсов, `template-with-new-rg.json`.

```
az deployment sub create --template-file "<path-to-template-with-new-rg.json" --location <region-location-name> --parameters appId=<app-id-from-previous-step> appSecret=<password-from-previous-step> botId=<id-or-bot-app-service-name> botSku=F0 newAppServicePlanName=<new-service-plan-name> newWebAppName=<bot-app-service-name> groupName=<new-group-name> groupLocation=<region-location-name> newAppServicePlanLocation=<region-location-name>" --name "<bot-app-service-name>"
```

#### NOTE

Этот шаг может занять несколько минут.

ПАРАМЕТР	ОПИСАНИЕ
name	Имя развертывания.
template-file	Путь к шаблону ARM. Обычно файл <code>template-with-new-rg.json</code> размещается в папке <code>deploymentTemplates</code> проекта бота. Это путь к существующему файлу шаблона. Можно указать абсолютный или относительный путь к текущему каталогу. Все шаблоны бота создают файлы шаблонов ARM.
location	Расположение. Значения из <code>az account list-locations</code> . Расположение по умолчанию можно настроить с помощью <code>az configure --defaults location=&lt;location&gt;</code> .
параметры	Параметры развертывания в формате списка пар "ключ — значение". Введите следующие значения параметров:

- `appId` — значение *идентификатора приложения* из выходных данных JSON, созданных на шаге [Создание регистрации приложения](#).
- `appSecret` — пароль, указанный на шаге [Создание регистрации приложения](#).
- `botId` — имя создаваемого ресурса регистрации канала бота. Оно должно быть глобально уникальным. Оно используется как неизменяемый идентификатор бота и сохраняется в качестве отображаемого имени бота, но это значение вы можете изменить.
- `botSku` — ценовая категория, где возможны значения F0 (Бесплатный) или S1 (Стандартный).
- `newAppServicePlanName` — имя нового плана службы приложений.
- `newWebAppName` — имя службы приложений для бота.
- `groupName` — имя новой группы ресурсов.
- `groupLocation` — расположение группы ресурсов Azure.
- `newAppServicePlanLocation` — расположение плана службы приложений.

#### Развертывание с помощью шаблона ARM в существующей группе ресурсов

На этом шаге вы создадите службу приложения для бота, то есть определите этап развертывания для бота. Если вы используете существующую группу ресурсов, можно выбрать существующий план службы приложений или создать новый. Выберите любой вариант, который вам подходит.

- [Вариант 1. Существующий план Службы приложений](#)
- [Вариант 2. Новый план Службы приложений](#)

#### NOTE

Этот шаг может занять несколько минут.

#### Вариант 1. Существующий план Службы приложений

В этом варианте мы используем существующий план Службы приложений, но при этом создаем новое веб-приложение и новую регистрацию каналов бота.

Эта команда ниже определяет идентификатор и отображаемое имя бота. Параметр `botId` должен быть глобально уникальным. Он используется как неизменяемый идентификатор бота. Отображаемое имя бота является изменяемым.

#### TIP

Используйте шаблон ARM для существующей группы ресурсов, [template-with-preexisting-rg.json](#).

```
az deployment group create --resource-group "<name-of-resource-group>" --template-file "<path-to-template-with-preexisting-rg.json>" --parameters appId=<app-id-from-previous-step> appSecret=<password-from-previous-step> botId=<id or bot-app-service-name> newWebAppName=<bot-app-service-name> existingAppServicePlan=<name-of-app-service-plan> appServicePlanLocation=<region-location-name> --name "<bot-app-service-name>"
```

#### Вариант 2. Новый план Службы приложений

В этом варианте мы создаем план Службы приложений, веб-приложение и регистрацию каналов бота.

#### TIP

Используйте шаблон ARM для существующей группы ресурсов, [template-with-preexisting-rg.json](#).

```
az deployment group create --resource-group "<name-of-resource-group>" --template-file "<path-to-template-with-preexisting-rg.json>" --parameters appId=<app-id-from-previous-step> appSecret=<password-from-previous-step> botId=<id or bot-app-service-name> newWebAppName=<bot-app-service-name> newAppServicePlanName=<name-of-app-service-plan> newAppServicePlanLocation=<region-location-name> --name "<bot-app-service-name>"
```

ПАРАМЕТР	ОПИСАНИЕ
name	Имя развертывания.
resource-group	Имя группы ресурсов Azure.
template-file	Путь к шаблону ARM. Обычно файл <code>template-with-preexisting-rg.json</code> размещается в папке <code>deploymentTemplates</code> проекта. Это путь к существующему файлу шаблона. Можно указать абсолютный или относительный путь к текущему каталогу. Все шаблоны бота создают файлы шаблонов ARM.
location	Расположение. Значения из <code>az account list-locations</code> . Расположение по умолчанию можно настроить с помощью <code>az configure --defaults location=&lt;location&gt;</code> .

ПАРАМЕТР	ОПИСАНИЕ
параметры	Параметры развертывания в формате списка пар "ключ — значение". Введите следующие значения параметров:

- `appId` — значение `appId`, созданное на шаге [3.1 Создание регистрации приложения](#).
- `appSecret` — пароль, указанный на шаге [3.1 Создание регистрации приложения](#).
- `botId` — имя создаваемого ресурса регистрации канала бота. Оно должно быть глобально уникальным. Оно используется как неизменяемый идентификатор бота и сохраняется в качестве отображаемого имени бота, но это значение вы можете изменить.
- `newWebAppName` — имя службы приложений для бота.
- `newAppServicePlanName` — имя создаваемого ресурса плана службы приложений.
- `appServicePlanLocation` — расположение плана службы приложений.

## 5. Подготовка кода к развертыванию

### 5.1. получение или создание необходимых файлов

Перед развертыванием бота нужно подготовить файлы проекта.

Убедитесь, что вы находитесь в папке проекта программы-робота. Затем подготовьте код Bot для развертывания.

- [C#](#)
- [JavaScript](#)
- [TypeScript](#)
- [Python](#)

```
az bot prepare-deploy --lang Csharp --code-dir "." --proj-file-path "MyBot.csproj"
```

Необходимо указать путь к CSIROJ-файлу относительно папки --code-dir. Для этого можно применить аргумент --proj-file-path. Эта команда разрешит аргументы --code-dir и --proj-file-path в значение ./MyBot.csproj.

Эта команда создаст файл `.deployment` в папке проекта бота.

### 5.2 Архивация каталога кода вручную

Если для развертывания кода бота используются ненастроенные интерфейсы [API развертывания ZIP-файлов](#), Web App или Kudu демонстрирует следующее поведение.

*Kudu по умолчанию предполагает, что развертывание из ZIP-файлов готово к выполнению и не требует дополнительных шагов сборки, таких как `npm install` или `dotnet restore/dotnet publish`.*

Важно включить весь код сборки и все необходимые зависимости в развертываемый ZIP-файл, иначе бот не будет работать должным образом.

## IMPORTANT

Прежде чем архивировать файлы проекта, убедитесь, что вы зашли в папку проекта бота.

- Для ботов на C# это папка, в которой расположен CSPROJ-файл.
- Для ботов на JavaScript это папка, в которой расположен файл app.js или index.js.
- Для ботов на TypeScript это папка, которая содержит папку *src* (с файлами bot.ts и index.ts).
- Для ботов Python это папка, в которой расположен файл app.py.

Перед выполнением команды для создания ZIP-файла убедитесь, что **в** папке проекта выбраны все файлы и папки. В папке проекта будет создан один ZIP-файл. Если расположение корневой папки выбрано неверно, **бот не сможет запуститься на портале Azure**.

## Развертывание кода в Azure

Теперь мы готовы развернуть код в виде веб-приложения Azure. Запустите следующую команду из командной строки, чтобы выполнить развертывание с помощью принудительного развертывания в Kudu из ZIP-файла веб-приложения.

```
az webapp deployment source config-zip --resource-group "<resource-group-name>" --name "<name-of-web-app>" -  
-src "<project-zip-path>"
```

ПАРАМЕТР	ОПИСАНИЕ
resource-group	Имя группы ресурсов Azure, которая содержит бота. (Это будет группа ресурсов, которую вы использовали или создали при регистрации приложения для бота.)
name	Имя веб-приложения, которое вы использовали ранее.
src	Путь к созданному ранее ZIP-файлу проекта.

## NOTE

Этот шаг может занять несколько минут. Кроме того, после завершения развертывания бот станет доступным для тестирования через несколько минут.

## Тестирование в веб-чате

1. В браузере перейдите на [портал Azure](#).
2. На панели слева щелкните **Группы ресурсов**.
3. На панели справа найдите свою группу.
4. Щелкните имя группы.
5. Щелкните ссылку регистрации канала бота.
6. В области **Bot Channels Registration** (Регистрация канала бота) щелкните **Test in Web Chat** (Протестировать в Web Chat). Или на панели справа щелкните поле **Тест**.

См. сведения о регистрации каналов бота в руководстве по [регистрации бота с помощью службы Azure Bot](#).

## Дополнительные сведения

Развертывание бота в Azure подразумевает оплату за используемые службы. Руководство по [управлению счетами и расходами](#) поможет вам понять, как расшифровывать счета Azure, отслеживать использование и расходы, а также управлять учетными записями и подписками. См. также [документацию по интерфейсу Command-Line Azure \(CLI\)](#) и [Azure CLI заметки о выпуске](#).

## Дальнейшие действия

[Настройка непрерывного развертывания](#)

# Непрерывное развертывание с использованием GIT в службе приложений Azure

27.03.2021 • 3 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье показано, как настроить для бота непрерывное развертывание. Вы можете включить непрерывное развертывание, чтобы автоматически развертывать изменения в коде из исходного репозитория в Azure. В этом разделе мы опишем настройку непрерывного развертывания на примере GitHub. Сведения о настройке непрерывного развертывания с использованием других систем управления версиями см. в разделе со ссылками на дополнительные ресурсы внизу страницы.

## Предварительные требования

- Если у вас еще нет подписки Azure, [создайте бесплатную учетную запись](#), прежде чем начинать работу.
- Вам **необходимо** [развернуть бота в Azure](#), прежде чем включать непрерывное развертывание.

## Подготовка репозитория

Убедитесь, что корень репозитория содержит нужные файлы проекта. Это позволит вам автоматически получать сборки из поставщика сборки.

ПАРАМЕТРЫ ВЫПОЛНЕНИЯ	ФАЙЛЫ В КОРНЕВОМ КАТАЛОГЕ
ASP.NET Core	.sln или .csproj
Node.js	server.js, app.js или package.json со скриптом запуска
Python	app.py

## Непрерывное развертывание с помощью GitHub

Чтобы включить непрерывное развертывание для GitHub, перейдите на страницу **Службы приложений** нужного бота на портале Azure.

1. Щелкните **Центр развертывания** > GitHub > **Авторизовать**.

The screenshot shows the Microsoft Azure Deployment Center interface. On the left, there is a navigation sidebar with links like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Security, Deployment (Quickstart, Deployment slots, Deployment Center), Settings (Configuration, Authentication / Authorization, Application Insights, Identity, Backups), and a search bar. The 'Deployment Center' link is highlighted with a red box. The main content area is titled 'Deployment Center' with a gear icon. It displays a four-step process: SOURCE CONTROL (step 1, highlighted with a red box), BUILD PROVIDER (step 2), CONFIGURE (step 3), and SUMMARY (step 4). Below this, it says 'Continuous Deployment (CI / CD)'. There are three cards: 'Azure Repos' (Configure continuous integration with an Azure Repo, part of Azure DevOps Services (formerly known as VSTS)), 'GitHub' (Configure continuous integration with a GitHub repo, highlighted with a red box), and 'Bitbucket' (Configure continuous integration with a Bitbucket repo). A blue 'Authorize' button is located at the bottom right of the GitHub card.

а. В открывшемся окне браузера щелкните **Авторизовать** AzureAppService.



## Authorize Azure App Service

Azure App Service by [AzureAppService](#)  
wants to access your [REDACTED] account



**Repository webhooks and services**

Admin access



**Repositories**

Public and private



**Workflow**

Update GitHub Action Workflow files.



### Organization access



[Authorize AzureAppService](#)

Authorizing will redirect to  
<https://functions.azure.com>

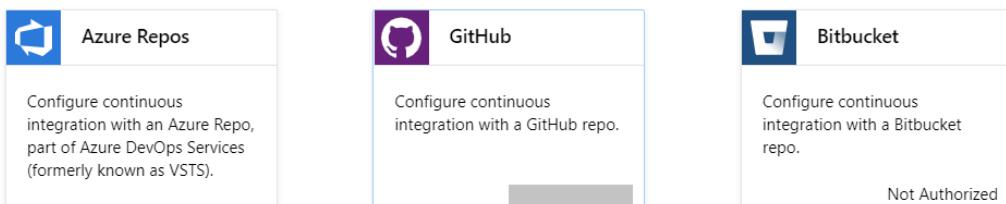
- b. После авторизации AzureAppService вернитесь в **Центр развертывания** на портале Azure.
2. Нажмите кнопку **Продолжить**.

# Deployment Center

App Service Deployment Center enables you to choose the location of your code as well as options for build and deployment to the cloud. [Learn more](#)



## Continuous Deployment (CI / CD)



[Change Account](#)

[Continue](#)

- На странице **Поставщик сборки** выберите нужный поставщик сборки и щелкните **Продолжить**.
- На странице **Настройка** введите необходимые сведения и щелкните **Продолжить**. Требуемые сведения будут зависеть от выбранной службы управления версиями и поставщика сборки.
- На странице **Сводка** проверьте параметры и щелкните **Готово**.

Теперь у вас настроено непрерывное развертывание для GitHub. Новые фиксации в выбранном репозитории в приложении Службы приложений будут непрерывно развертываться. Фиксации и развертывания можно отслеживать на странице **Центра развертывания**.

## Отключение непрерывного развертывания

Хотя ваш бот и настроен для непрерывного развертывания, не используйте интерактивный редактор кода для внесения изменений в бот. Если вы хотите использовать интерактивный редактор кода, можно временно отключить непрерывное развертывание.

Чтобы отключить непрерывное развертывание, сделайте следующее:

- На портале Azure перейдите в колонку **параметров Службы приложений** для нужного бота и щелкните **Центр развертывания**.
- Нажмите кнопку **Отключить** для отключения непрерывного развертывания. Чтобы снова включить непрерывное развертывание, повторите действия, описанные в соответствующем разделе выше.

## Дополнительные ресурсы

- См. сведения о [непрерывном развертывании в Службе приложений Azure](#).
- При использовании GitHub Actions для поставщика сборки в репозитории создается рабочий процесс. См. сведения об использовании [GitHub Actions](#) на GitHub.

# Управление ботом

27.03.2021 • 5 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В браузере перейдите на [портал Azure](#). Выберите приложение ресурсов, например **Регистрация каналов Bot**. В области навигации вы увидите разделы, описанные ниже.

## Параметры ресурсов Bot

Существует несколько различных типов ресурсов Bot, таких как **робот веб-приложения** или **Регистрация каналов Bot**. Информация, доступная для каждого типа ресурса, в основном одинакова.

### Общие сведения

В верхней части области навигации находятся ссылки на общие сведения, применимые к роботу.

 Overview

 Activity log

 Access control (IAM)

 Tags

Ссылка	Описание
<b>Обзор</b>	Содержит высокоуровневые сведения о Bot, например <b>идентификатор под писки Bot</b> и <b>конечную точку обмена сообщениями</b> . В обзоре программы для <b>робота веб-приложения</b> можно также загрузить исходный код Bot.
<b>Журнал действий</b>	Содержит подробные сведения о диагностике и аудите для ресурсов Azure и платформы Azure, от которых они зависят. Дополнительные сведения см. в статье <a href="#">Общие сведения о журналах платформы Azure</a> .
<b>Управление доступом (IAM)</b>	Отображает доступ к ресурсам Azure для пользователей Access или других субъектов безопасности. Дополнительные сведения см. в <a href="#">разделе Просмотр доступа пользователя к ресурсам Azure</a> .
<b>Теги</b>	Отображает теги для ресурсов Azure, групп ресурсов и подписок, чтобы логически организовывать их в таксономию. Дополнительные сведения см. в статье <a href="#">Использование тегов для организации ресурсов в Azure</a> .

### Параметры

В разделе **Параметры** приведены ссылки на большинство вариантов управления для программы-робота.

## Settings

-  Bot profile
-  Configuration
-  Channels
-  Speech priming
-  Pricing
-  Test in Web Chat
-  Encryption
-  Properties
-  Locks

Ссылка	Описание
<b>Профиль бота</b>	Управляет различными параметрами профиля Bot, такими как отображаемое имя, значок и описание.
<b>Конфигурация</b>	Управляет различными параметрами Bot, такими как аналитика, конечная точка обмена сообщениями и параметры подключения OAuth.
<b>Каналы</b>	Настраивает каналы, используемые Bot для взаимодействия с пользователями.
<b>Подготовка речи</b>	Управляет подключениями между приложением LUIS и службой Распознавание речи Bing.
<b>Цены</b>	Управляет ценовой категорией для службы Bot.
<b>Тестирование в веб-чате</b>	Использует встроенное управление веб-чат для быстрого тестирования программы Bot.
<b>Шифрование</b>	Управляет ключами шифрования.
<b>Свойства</b>	Список свойств ресурсов Bot, таких как идентификатор ресурса, идентификатор подписки и идентификатор группы ресурсов.
<b>Блокировки</b>	Управляет блокировками ресурсов.

## Наблюдение

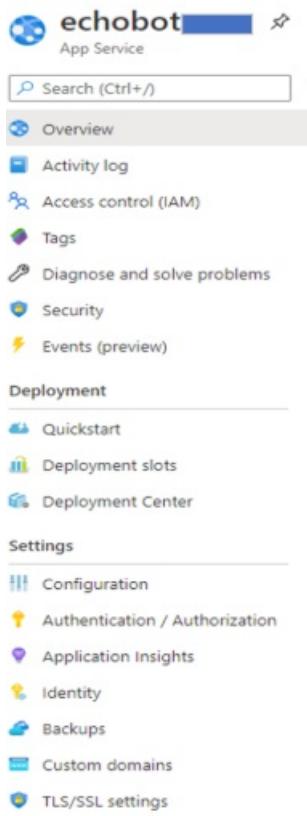
В разделе **МОНИТОРИНГ** приведены ссылки на большинство параметров мониторинга программы-робота.

Ссылка	Описание
<b>Анализ в ходе сеанса</b>	Позволяет аналитике просматривать собранные данные с помощью Application Insights.

# Параметры службы приложений

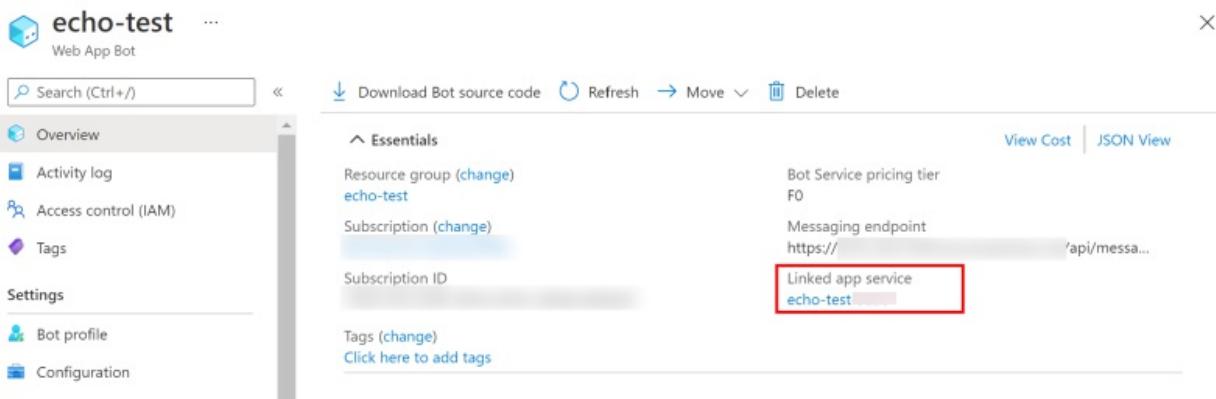
Приложение-робот, также известное как *Служба приложения*, имеет набор параметров приложения, доступ к которым можно получить с помощью портал Azure. Эти параметры приложения являются переменными, передаваемыми как переменные среды в код приложения. Дополнительные сведения см. в разделе [Настройка приложения службы приложений на портал Azure](#).

1. В браузере перейдите на [портал Azure](#).
2. Найдите службу приложений и выберите ее имя.
3. Отобразится служба приложений.



Левая панель **Параметры приложения** содержит подробные сведения о программе-роботе, такие как среда Bot, параметры отладки и ключи параметров приложения.

Если вы создали робот веб-приложения, вы можете найти ярлык для соответствующей службы приложений в области **Обзор**.



## MicrosoftAppID и MicrosoftAppPassword

MicrosoftAppID и MicrosoftAppPassword хранятся в файле настроек бота ( `appsettings.json` или `.env` ) или Azure Key Vault. Чтобы извлечь их, скачайте файл параметров или конфигурации бота (для старых

ботов, если он существует) либо обратитесь к Azure Key Vault. Это может быть необходимо при локальном тестировании с использованием идентификатора и пароля.

**NOTE**

Служба робота для **регистрации каналов Bot** поставляется с *микрософтапплид*, но поскольку с этим типом службы не связана служба приложений, для поиска *микрософтаппассворд* также не отображается колонка **параметров приложения**. Сведения о том, как создать пароль, см. в статье о [пароле для регистрации канала ленты](#).

## Дальнейшие действия

Теперь, когда вы ознакомились с колонкой службы ботов на портале Azure, узнайте, как использовать сетевой редактор кода для настройки вашего бота.

[Принципы работы бота](#)

# Регистрация каналов бота

27.03.2021 • 9 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье показано, как зарегистрировать робот с помощью **службы Azure Bot**. Если Bot размещен в других местах, вы также можете сделать его доступным в Azure и подключить его к поддерживаемым каналам. Существует два способа регистрации робота в Azure:

1. Следуя инструкциям в статье Создание программы [-робота с помощью службы Azure Bot](#), программа Bot создается и регистрируется в Azure и создается веб-приложение для размещения робота. Этот подход используется при разработке и размещении программы-робота в Azure.
2. Чтобы создать и разработать программу Bot локально, выполните действия, описанные в этой статье. При регистрации робота вы предоставляете веб-адрес, на котором размещена программа Bot. Вы по-прежнему можете разместить его в Azure.

## IMPORTANT

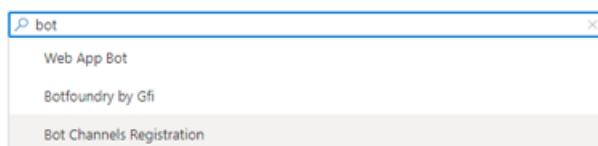
Только вы должны зарегистрировать робота, если он не размещен в Azure. Программы-роботы, созданные с помощью портал Azure или интерфейса командной строки, уже зарегистрированы в службе Azure Bot.

Дополнительные сведения см. в статьях [развертывание программы Bot](#) и [создание робота с помощью службы Azure Bot](#).

## Создание приложения регистрации

1. В браузере перейдите на [портал Azure](#). Если у вас нет подписки, вы можете зарегистрироваться для получения [бесплатной учетной записи](#).
2. На панели слева щелкните **создать ресурс**.
3. В верхней центральной панели поиска введите *Bot*. В раскрывающемся списке выберите **Регистрация каналов Bot**.

New 



4. Нажмите кнопку **Создать**.
5. В форме "**Регистрация каналов Bot**" Укажите запрашиваемые сведения о программе-роботе, как указано в таблице ниже.

ПАРАМЕТР	РЕКОМЕНДУЕМОЕ ЗНАЧЕНИЕ	ОПИСАНИЕ
----------	------------------------	----------

ПАРАМЕТР	РЕКОМЕНДУЕМОЕ ЗНАЧЕНИЕ	ОПИСАНИЕ
<b>Имя бота</b>	<Your bot's handle>	<i>Маркер Bot</i> — это уникальный идентификатор для программы-робота. Вы можете выбрать другое отображаемое имя для программы-робота в колонке <i>Параметры</i> после создания программы-робота.
<b>Подписка</b>	<Your subscription>	Выберите подписку Azure, которую нужно использовать.
<b>Группа ресурсов</b>	<Your resource group name>	Создайте новую <a href="#">группу ресурсов</a> или выберите существующую.
<b>Расположение</b>	западная часть США	Выберите географическое расположение группы ресурсов. Обычно лучше выбрать расположение, близкое к вам. Невозможно изменить расположение после создания группы ресурсов.
<b>Ценовая категория</b>	F0	Выберите ценовую категорию. Вы можете обновить ценовую категорию в любое время. Дополнительные сведения см. на странице <a href="#">цен на службу Bot</a> .
<b>Messaging endpoint</b> (Конечная точка обмена сообщениями)	https:// <your bot name> . azurewebsites.NET/API/messages	Введите URL-адрес конечной точки обмена сообщениями своего бота. Это поле можно оставить пустым, так как после развертывания Bot необходимо ввести нужный URL-адрес.
<b>Application Insights</b>	C	Решите, хотите ли вы <b>включить</b> или <b>выключить</b> <a href="#">Application Insights</a> . Если вы выберете параметр <b>Вкл.</b> , необходимо также указать региональное расположение.
<b>ИДЕНТИФИКАТОР и пароль приложения</b>	Создание идентификатора приложения и пароля	Используйте этот параметр, если вам нужно вручную ввести идентификатор и пароль приложения Майкрософт. Подробные сведения см. в следующем разделе <a href="#">Регистрация приложений вручную</a> . В противном случае в процессе регистрации будет создан новый идентификатор приложения Майкрософт.

6. Оставьте поле *Конечная точка обмена сообщениями* пустым для Now. После развертывания Bot необходимо ввести нужный URL-адрес.
7. Щелкните **идентификатор приложения и пароль Microsoft**, выделенные на снимке экрана

ниже. Следующие шаги позволяют создать пароль приложения регистрации и идентификатор приложения для использования в файлах конфигурации Bot `appsettings.json` (.NET), `.env` (JavaScript) И `config.py` (Python).

### Bot Channels Registration

Bot Service

Bot handle \*

Subscription \*

Resource group \*  Create new

Location \*  West US 2

Pricing tier [View full pricing details](#)  S1 (1K Premium Msgs/Unit)

Messaging endpoint  https URL

Application Insights  On Off

Microsoft App ID and password  > Auto create App ID and password

8. Нажмите кнопку **Создать**.
9. Щелкните ссылку **создать идентификатор приложения на портале регистрации приложений**.
10. В отображаемом окне *Регистрация приложения* щелкните новую вкладку **Регистрация** в левом верхнем углу.
11. Введите имя приложения бота, которое вы хотите зарегистрировать.
12. Для поддерживаемых типов учетных записей выберите тип в соответствии с вашими требованиями. См. также [использование учетных данных Bot](#).
13. Нажмите кнопку **зарегистрировать**. После завершения Azure отобразит страницу обзора для приложения.
14. Скопируйте значение **идентификатора приложения (клиента)** и сохраните его в файле.
15. На панели слева щелкните **сертификат и секреты**.
16. В разделе *Секреты клиента* щелкните **New client secret** (Новый секрет клиента).
17. Добавьте описание, чтобы отличать этот секрет от других секретов, которые могут вам понадобиться для создания этого приложения.
18. Для параметра **срок действия** задается значение.
19. Нажмите кнопку **Добавить**.
20. Скопируйте секрет клиента и сохраните его в файл. Обязательно запишите файл в безопасном месте.
21. Вернитесь к окну Регистрация канала бота и вставьте значения Идентификатор приложения и Секрет клиента в поля Код приложения Майкрософт и Пароль соответственно.
22. Нажмите кнопку **OK**.

23. Нажмите кнопку "**создать**" и дождитесь создания ресурса. Он отобразится в списке ресурсов.

#### NOTE

Приложение регистрации будет показывать глобальный регион, даже если вы выбрали другую. Это ожидаемое поведение.

## Регистрация приложений вручную

Регистрацию вручную необходимо выполнять в таких ситуациях:

- Вы не можете выполнить регистрацию в своей организации, и требуется третья сторона, чтобы создать идентификатор приложения для бота, сборка которого выполняется.
- Необходимо вручную создать собственный идентификатор и пароль приложения.

#### IMPORTANT

При создании приложения в разделе **Поддержка типов учетных записей** выберите один из следующих параметров для нескольких типов клиентов:

ТИП	ОПИСАНИЕ
Учетные записи в любом каталоге организации (любой клиент Azure AD)	Этот параметр обеспечивает меньшее воздействие, запрещая доступ и в случае, если OAuth не поддерживается.
Учетные записи в любом каталоге организации (любой Azure AD — клиент) и личные учетные записи Microsoft (например, Xbox, Outlook.com).	Этот вариант хорошо подходит для поддержки проверки подлинности OAuth и Bot.

Дополнительные сведения см. в статье [регистрация приложения на платформе Microsoft Identity](#).

## Обновление бота

1. Пакет SDK для Bot Framework для .NET. Задайте следующие значения ключей в `appsettings.json` файле:

- `MicrosoftAppId = <appId>`
- `MicrosoftAppPassword = <appSecret>`

2. Пакет SDK для Bot Framework для Node.js. Задайте следующие переменные среды в `.env` файле:

- `MICROSOFT_APP_ID = <appId>`
- `MICROSOFT_APP_PASSWORD = <appSecret>`

3. Пакет SDK для Bot Framework для Python. Задайте следующие переменные среды в `config.py` файле:

- `APP_ID = os.environ.get("MicrosoftAppId", <appId>)`
- `APP_PASSWORD = os.environ.get("MicrosoftAppPassword", <appSecret>)`

## Тестирование бота

После создания службы бота [проверьте ее в компоненте "Веб-чат"](#).

## Дальнейшие действия

В этом разделе вы узнали, как зарегистрировать размещенный бот в службе Bot. Далее узнайте, как управлять службой Bot.

[Управление ботом](#)

# Аналитика бота

27.03.2021 • 5 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Analytics — это расширение [Application Insights](#). Application Insights предоставляет данные **уровня службы** и данные инструментирования, например трафик, задержку и интеграции. Analytics поддерживает отчеты **уровня общения** для данных пользователя, сообщений и каналов.

## Просмотр аналитики для бота

Чтобы получить доступ к Analytics, откройте бот на портале Azure и щелкните **Analytics**.

Слишком много данных? Вы можете [включить и настроить выборку](#) для службы Application Insights, связанной с ботом. Это позволяет сократить объем трафика телеметрии и хранимых данных, сохраняя статистически верный анализ.

### Указание канала

Выберите, какие каналы отображаются на схемах ниже. Обратите внимание, что если бот не включен на канале, данные этого канала будут отсутствовать.

The screenshot shows a list of communication channels on the left and right sides, each with a checkbox. On the left, checked channels include Bing, Direct Line, Facebook, Kaizala, Microsoft Teams, Skype for Business, Telegram, and Web Chat. On the right, checked channels include Cortana, Email, GroupMe, Kik, Skype, Slack, Twilio (SMS), and WeChat. A date range selector at the top right shows "11/3/2018 - 12/4/2018".

- Bing
- Cortana
- Direct Line
- Email
- ...

11/3/2018 - 12/4/2018

<input checked="" type="checkbox"/> Bing	<input checked="" type="checkbox"/> Cortana
<input checked="" type="checkbox"/> Direct Line	<input checked="" type="checkbox"/> Email
<input checked="" type="checkbox"/> Facebook	<input checked="" type="checkbox"/> GroupMe
<input checked="" type="checkbox"/> Kaizala	<input checked="" type="checkbox"/> Kik
<input checked="" type="checkbox"/> Microsoft Teams	<input checked="" type="checkbox"/> Skype
<input checked="" type="checkbox"/> Skype for Business	<input checked="" type="checkbox"/> Slack
<input checked="" type="checkbox"/> Telegram	<input checked="" type="checkbox"/> Twilio (SMS)
<input checked="" type="checkbox"/> Web Chat	<input checked="" type="checkbox"/> WeChat

- Установите флажок, чтобы включить канал в диаграмму.
- Снимите флажок, чтобы удалить канал из диаграммы.

### Указание периода времени

Анализ доступен только за последние 90 дней. Сбор данных начинается после включения Application Insights.

The screenshot shows a list of time intervals on the left and right sides, each with a radio button. On the left, radio buttons are shown for "Last Hour", "Last Day", "Last Week", and "Last Month". On the right, radio buttons are shown for "Last Month" and "Last 90 Days". A date range selector at the top right shows "11/3/2018 - 12/4/2018".

- Bing
- Cortana
- Direct Line
- Email
- ...

11/3/2018 - 12/4/2018

<input type="radio"/> Last Hour	<input type="radio"/> Last Day	<input type="radio"/> Last Week	<input checked="" type="radio"/> Last Month	<input type="radio"/> Last 90 Days
---------------------------------	--------------------------------	---------------------------------	---------------------------------------------	------------------------------------

Щелкните стрелку раскрывающегося меню и выберите количество времени отображения диаграммы. Обратите внимание, что изменение общего интервала времени приведет к соответствующему изменению

шага приращения времени (ось X) на диаграммах.

## Общий итог

Общее число активных пользователей и действий, отправленных и полученных в течение заданного интервала времени. Дефисы -- указывают на отсутствие действий.

## Сохранение

При хранении отслеживается, сколько пользователей, отправивших одно сообщение, вернулись позже и отправили еще одно. Диаграмма — это скользящее 10-дневное окно; результаты не изменяются при изменении интервала времени.

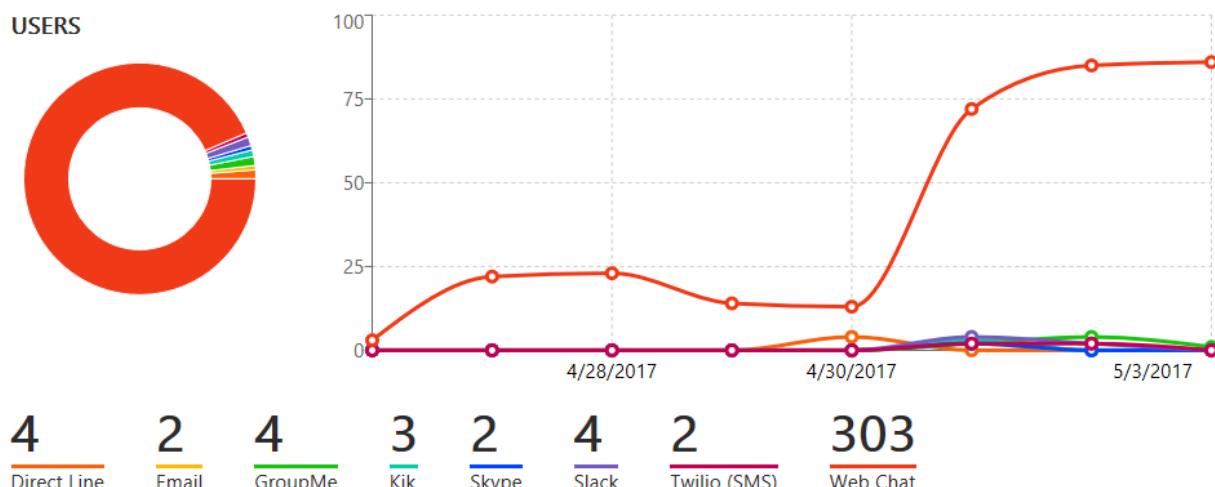
RETENTION - % USERS WHO MESSAGED AGAIN (LAST 10 DAYS)

Date	Users	Days later									
		1	2	3	4	5	6	7	8	9	10
4/23/2017	0	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
4/24/2017	13	31%	8%	8%	8%	8%	8%	8%	15%	8%	
4/25/2017	43	2%	2%	2%	5%	5%	2%	5%	2%		
4/26/2017	27	4%	4%	4%	4%	4%	4%	4%			
4/27/2017	37	3%	3%	3%	3%	3%	3%				
4/28/2017	22	5%	5%	5%	5%	5%					
4/29/2017	23	9%	9%	9%	9%						
4/30/2017	14	7%	7%	7%							
5/1/2017	17	18%	12%								
5/2/2017	87	9%									

Обратите внимание, что последняя возможная дата — два дня назад. Пользователь отправил сообщение позавчера и *вернулся* вчера.

## Пользователь

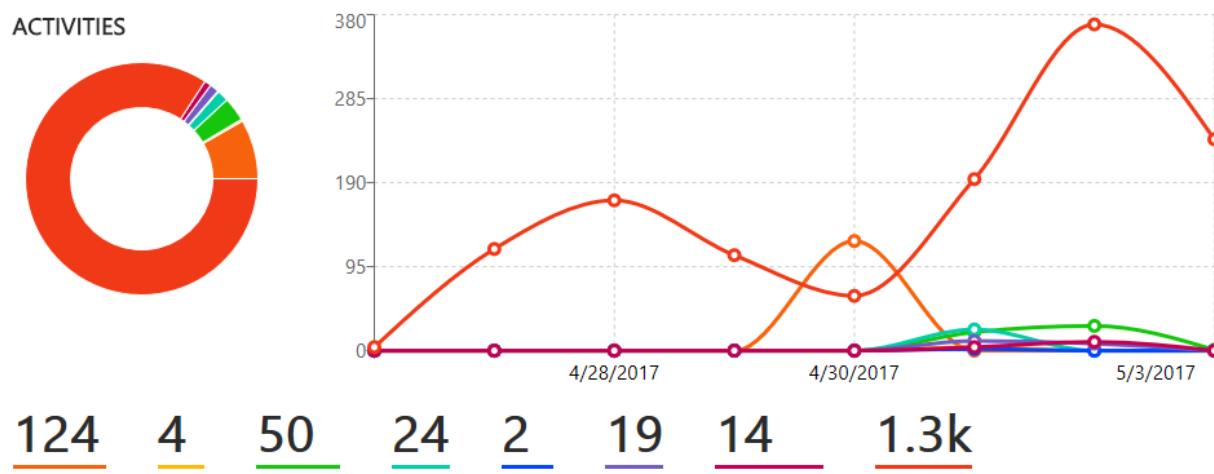
На диаграмме пользователей отслеживается, сколько пользователей получили доступ к боту с помощью каждого канала за указанный интервал времени.



- На процентной диаграмме указано процентное соотношение пользователей, использовавших каждый канал.
- На линейном графике показано, сколько пользователей получали доступ к боту за определенное время.
- В условных обозначениях линейного графика указано цветовое представление каждого канала и общее число пользователей во время заданного периода времени.

## Действия

На диаграмме действий отслеживается количество действий, отправленных и полученных с помощью каждого канала за указанный интервал времени.



- На процентной диаграмме показано процентное соотношение сообщений, переданных по каждому каналу.
- На линейном графике показано количество действий, полученных и отправленных за указанный интервал времени.
- В условных обозначениях линейного графика указано цветовое линейное представление каждого канала и общее число действий, отправленных и полученных на этом канале во время заданного периода времени.

## Включение аналитики

Чтобы расширение Analytics стало доступно, нужно включить и настроить Application Insights. После включения Application Insights начнет собирать данные. Например, если служба Application Insights включена неделю назад для бота, созданного шесть месяцев назад, он соберет данные за неделю.

### NOTE

Analytics требуется [ресурс подписки Azure](#) и Application Insights. Чтобы получить доступ к Application Insights, откройте бот на [портале Azure](#) и щелкните **Параметры**.

Application Insights можно добавить при создании ресурса бота.

Также ресурс Application Insights можно создать и добавить к боту позднее.

1. Создайте ресурс [Application Insights](#).
2. Откройте бот на панели мониторинга. Щелкните **Параметры** и прокрутите вниз до раздела **Analytics**.
3. Введите сведения для подключения бота к Application Insights. Все поля обязательные.

### Analytics

Application Insights Instrumentation key [?](#)

Instrumentation key (Azure Application Insights key)

Application Insights API key [?](#)

API key (User-Generated Application Insights API key)

Application Insights Application ID [?](#)

Application ID (Application Insights Application ID)

Дополнительные сведения о том, как найти эти значения, см. в статье [Ключи Application Insights](#).

## Дополнительные ресурсы

- [Ключи Application Insights](#)

# Настройка параметров регистрации для программы-робота

27.03.2021 • 5 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Параметры регистрации робота, такие как отображаемое имя, значок и описание, можно просмотреть и изменить в области « **профиль Bot** ». Параметры конфигурации Bot, такие как конечная точка обмена сообщениями, идентификатор приложения Microsoft и Application Insights, можно просмотреть и изменить в области **конфигурации** .

## Профиль бота

The screenshot shows the 'Bot profile' settings page for a 'Web App Bot' named 'echo-test'. The main area displays fields for 'Display name' (set to 'echo-test'), 'Bot handle' (set to 'echo-test'), and 'Description'. A note at the top right states: 'Please allow 30 minutes for changes to bot settings to be reflected in all regions. Changes to icons may take up to 24 hours.' On the left, a sidebar lists other settings: 'Bot profile' (selected), 'Configuration', 'Channels', 'Speech priming', 'Pricing', 'Test in Web Chat', 'Encryption', 'Properties', 'Locks', 'Monitoring', 'Conversational analytics', 'Automation', 'Tasks (preview)', and 'Export template'. At the bottom are 'Apply' and 'Discard changes' buttons.

Ниже приведен список полей **профиля Bot** :

ПОЛЕ	ОПИСАНИЕ
Значок	Пользовательский значок для визуального отображения программы-робота в каналах и в качестве значка для программы-робота в Microsoft Teams или других службах. Этот значок должен быть в формате PNG, размер которого не превышает 30 КБ. Это значение можно изменить в любое время.

ПОЛЕ	ОПИСАНИЕ
Отображаемое имя	Имя бота на каналах и в каталогах. Имя может иметь длину до 35 символов. Это значение можно изменить в любое время.
Bot handle (Дескриптор бота)	Уникальный идентификатор для программы Bot. Это значение нельзя изменить после создания бота с помощью службы Bot.

Чтобы сохранить изменения, нажмите кнопку **Применить** в нижней части колонки.

## Конфигурация

Ниже приведен список полей **конфигурации**.

ПОЛЕ	ОПИСАНИЕ
Messaging endpoint (Конечная точка обмена сообщениями)	Конечная точка для обмена данными с ботом.
Microsoft App ID (Идентификатор приложения Майкрософт)	Уникальный идентификатор для бота. Это значение невозможно изменить. Вы можете создать пароль, щелкнув ссылку <b>Управление</b> .

ПОЛЕ	ОПИСАНИЕ
Application Insights Instrumentation key (Ключ инструментирования Application Insights)	Уникальный ключ для телеметрии бота. Скопируйте ключ Azure Application Insights в это поле, если вы хотите получать данные телеметрии для этого бота. Это значение является необязательным. Для ботов, созданных на портале Azure, этот ключ создается автоматически. Дополнительные сведения об этом поле см. в статье <a href="#">Ключи Application Insights</a> .
Ключ API Application Insights	Уникальный ключ для аналитики бота. Скопируйте ключ API Azure Application Insights в это поле, если вы хотите просматривать аналитику бота на панели мониторинга. Это значение является необязательным. Дополнительные сведения об этом поле см. в статье <a href="#">Ключи Application Insights</a> .
Application Insights Application ID (Идентификатор приложения Application Insights)	Уникальный идентификатор для Bot Analytics. Скопируйте ключ идентификатора Azure Insights в это поле, если вы хотите просматривать аналитику бота на панели мониторинга. Это значение является необязательным. Для ботов, созданных на портале Azure, этот ключ создается автоматически. Дополнительные сведения об этом поле см. в статье <a href="#">Ключи Application Insights</a> .

Чтобы сохранить изменения, нажмите кнопку **Применить** в нижней части колонки.

## Идентификатор приложения и пароль

ИДЕНТИФИКАТОР и пароль приложения регистрации присваиваются переменным Bot `MicrosoftAppID` и `MicrosoftAppPassword` в файлах `appsettings.json` (.NET) и `.env` (JavaScript).

### NOTE

Регистрация каналов Bot имеет идентификатор приложения, но, так как с ней не связана служба приложений, пароль отсутствует. Чтобы создать пароль, выполните действия, описанные в следующем разделе.

### Получение пароля для регистрации

Приложение регистрации имеет **идентификатор приложения** (идентификатор приложения) и связанный с ним **пароль**. Служба Azure Bot назначает приложению уникальный идентификатор приложения. Пароль можно получить, выполнив описанные ниже действия.

1. В браузере перейдите на [портал Azure](#).
2. В списке ресурсов щелкните имя приложения регистрации.
3. На панели справа перейдите к разделу **Управление Bot** и щелкните **Параметры**. Отобразится страница **параметров** приложения регистрации.
4. Щелкните ссылку **Управление** рядом с идентификатором приложения *Microsoft*.

echo-test | Configuration

Web App Bot

Search (Ctrl+ /)

Overview

Activity log

Access control (IAM)

Tags

Settings

Bot profile

Configuration

Channels

Speech priming

Pricing

Test in Web Chat

Encryption

Properties

Locks

Monitoring

Conversational analytics

Automation

Tasks (preview)

Export template

Messaging endpoint

https:// azurewebsites.net/api/messages

Enable Streaming Endpoint

Microsoft App ID (Manage) ①  
2b83bb76-296d-431a-8831-4ac4cd420c5c

Application Insights Instrumentation key ①  
Instrumentation key (Azure Application Insights key)

Application Insights API key ①  
API key (User-Generated Application Insights API key)

Application Insights Application ID ①  
Application ID (Application Insights Application ID)

OAuth Connection Settings

No settings defined

Add Setting

Apply Discard changes

5. На странице *Сертификаты и секреты* нажмите кнопку **Создать секрет клиента**.

Client secrets

A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.

+ New client secret

DESCRIPTION	EXPIRES	VALUE
-------------	---------	-------

6. Добавьте описание, выберите срок действия и нажмите кнопку **Добавить**.

Add a client secret

Description

Expires

In 1 year

In 2 years

Never

Add Cancel

Будет создан пароль для вашего бота. Скопируйте этот пароль и сохраните его в файл. Вы сможете просмотреть этот пароль только в этот раз. Если вы не сохранили полный пароль, потребуется повторить процедуру, чтобы создать пароль, если он понадобится позже.

## Дополнительные сведения

Для обновления параметров бота из командной строки можно использовать [az bot update](#).

# Подключение бота к каналам

27.03.2021 • 7 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Канал — это подключение между приложениями связи и роботом. Программа-робот, зарегистрированная в Azure, использует каналы для упрощения взаимодействия с пользователями.

Вы можете настроить робота для подключения к любому из стандартных каналов, например Алекса, Facebook Messenger и временного резерва. Дополнительные сведения см. в статье [Регистрация каналов Bot](#).

В дополнение к предоставленным каналам можно также подключить робот к коммуникационному приложению, используя **прямую линию** в качестве канала.

Платформа Bot позволяет разрабатывать программы-роботы в независимом от канала виде путем нормализации сообщений, отправляемых программой-роботом в канал. Для этого необходимо выполнить следующие шаги:

- Преобразуйте сообщения из схемы "Bot Framework" в схему канала.
- Если канал не поддерживает все аспекты схемы Bot Framework, служба соединителя Bot пытается преобразовать сообщение в формат, поддерживаемый каналом. Например, если бот отправляет сообщение, содержащее карту с кнопками действий, в канал электронной почты, соединитель может отправить карту как изображение и включить действия как ссылки в текст сообщения.
- Для большинства каналов необходимо предоставить сведения о конфигурации канала для запуска программы-робота на канале. Для большинства каналов требуется, чтобы у робота была учетная запись в канале. Другие, например Facebook Messenger, нуждаются в том, чтобы приложение-робот было зарегистрировано в канале.

Чтобы настроить робот для подключения к каналу, выполните следующие действия.

1. Войдите на [портал Azure](#).
2. Выберите бот, который требуется настроить.
3. В колонке "Служба ботов" в разделе **Bot Management** (Управление ботом) щелкните **Каналы**.
4. Щелкните значок канала, который требуется добавить боту.

The screenshot shows the Azure Bot Service management interface. On the left, there is a sidebar with various navigation options: Overview, Activity log, Access control (IAM), Tags, Bot management (which is selected and highlighted in grey), Test in Web Chat, Analytics, Channels (which is also highlighted in grey), Settings, Speech priming, Bot Service pricing, Support + troubleshooting, and New support request. The main area is titled "Connect to channels". It displays a table with one row for Microsoft Teams. The table columns are Name, Health, and Published. The Microsoft Teams row shows "Microsoft Teams" in the Name column, "Running" in the Health column, and "--" in the Published column. To the right of the table is an "Edit" button with a pencil icon. Below the table is a link "Get bot embed codes". At the bottom of the main area, there is a section titled "Add a featured channel" with three icons: a blue circle, a globe, and a document with a list.

После настройки канала пользователи смогут использовать ваш бот.

## Подключение программы-робота к каналу

Шаги подключения для каждого канала различаются. Дополнительные сведения см. в соответствующей статье в таблице ниже.

КАНАЛ	ОПИСАНИЕ
Alexa	Обмен данными с пользователями через устройства Алекса, поддерживающие пользовательские навыки.
Direct Line	Интегрируйте робота в мобильное приложение, веб-страницу или другие приложения.
Электронная почта Office 365	Разрешить роботам обмениваться данными с пользователями через электронную почту Office 365.
Facebook	Подключите программу-робот к Facebook Messenger и к рабочей области Facebook, чтобы она могла взаимодействовать с пользователями на обеих платформах.
Kik	Настройте робота для взаимодействия с пользователями через приложение Kik Messaging.
LINE	Настройте робота для взаимодействия с пользователями через ЛИНЕЙное приложение.
Microsoft Teams	Настройка программы-робота для взаимодействия с пользователями через Microsoft Teams.
Skype	Настройка робота для взаимодействия с пользователями через Skype.
Skype для бизнеса	Настройка робота для взаимодействия с пользователями через Skype для бизнеса.
Slack	Настройте робота для взаимодействия с пользователями с помощью временного резерва.
Telegram	Настройте робота для взаимодействия с пользователями через Telegram.
Telephony (Телефония)	Настройте робота для взаимодействия с пользователями через канал телефонии Bot Framework.
Twilio	Настройте робота для взаимодействия с пользователями через облачную платформу для обмена данными Twilio.
WeChat	Настройте робота для взаимодействия с пользователями с помощью платформы WeChat.
Веб-чат	Автоматически настраивается при создании программы-робота с помощью службы платформы Bot.

КАНАЛ	ОПИСАНИЕ
Webex	Настройте робота для взаимодействия с пользователями с помощью WebEx.
Дополнительные каналы	Дополнительные каналы, доступные в виде адаптера с помощью <a href="#">предоставленных платформ</a> через боткит и <a href="#">репозитории сообщества</a> .

## Публикация бота

Процесс публикации на каждом канале разный.

### Skype

#### NOTE

Начиная с 31 октября 2019 г. канал Skype не принимает новые запросы на публикацию ботов. Это означает, что вы можете разрабатывать боты с использованием канала Skype, но бот будет доступен только 100 пользователям. Вы не сможете опубликовать бота для большего числа пользователей. Текущие боты в Skype будут работать без прерываний. Узнайте больше о том, [почему некоторые функции недоступны в Skype](#).

Боты публикуются в Skype со [страницы конфигурации](#). При публикации бот отправляется на проверку. До проверки бот поддерживает до 100 контактов. Для утвержденных ботов количество контактов не ограничено. Кроме того, их можно включать в каталог ботов Skype.

### Skype для бизнеса

#### IMPORTANT

Поддержка Skype для бизнеса Online будет прекращена 31 июля 2021 г. До этой даты клиенты смогут использовать Skype для бизнеса Online обычным образом. См. сведения о [прекращении поддержки Skype для бизнеса Online](#).

Боты Skype для бизнеса регистрируются в [клиенте Skype для бизнеса Online](#) администратором клиента.

Чтобы просмотреть состояние проверки, откройте бота на [портале Bot Framework](#) и выберите **Каналы**. Если бот не утвержден, в результатах будет указана причина. После внесения необходимых изменений отправьте бот на повторную проверку.

## Дополнительные ресурсы

Этот пакет SDK содержит примеры, которые можно использовать для создания ботов. Посетите [репозиторий примеров в GitHub](#), чтобы просмотреть список примеров.

# Реализация возможностей для определенных каналов

27.10.2020 • 16 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Некоторые каналы предоставляют функции, которые невозможно реализовать, используя только текст сообщений и вложения. Чтобы реализовать функции, связанные с каналами, вы можете передать в канал собственные метаданные через свойство `данных канала` для объекта действия. Например, используя свойство данных канала, бот может передать в Telegram команду отправки наклейки или потребовать, чтобы из Office 365 было отправлено сообщение электронной почты.

В этой статье объясняется, как реализовать функции, связанные с каналами, на основе свойства данных канала в действии сообщения.

CHANNEL	ФУНКЦИОНАЛЬНОСТЬ
Email	Отправка и получение сообщений электронной почты, которые содержат текст, тему и метаданные о важности.
Slack	Отправка сообщений Slack с полным контролем.
Facebook	Отправка уведомлений Facebook из кода приложения.
Telegram	Выполнение действий, реализованных в Telegram, таких как публикация голосового напоминания или наклейки.
Kik	Отправка и получение сообщений Kik.

## NOTE

Значение свойства данных канала для объекта действия — это объект JSON. Поэтому в примерах в этой статье показан ожидаемый формат свойства JSON `channelData` в различных сценариях. Чтобы создать объект JSON с помощью .NET, используйте класс  `JObject` (.NET).

## Создание пользовательского сообщения электронной почты

Чтобы создать пользовательское сообщение электронной почты, задайте `channelData` для свойства `Activity` объект JSON, который содержит следующие свойства:

СВОЙСТВО	ОПИСАНИЕ
<code>bccRecipients</code>	Строка адреса электронной почты, разделенная точками с запятой (;) для добавления к полю "Скрытая копия" сообщения.

СВОЙСТВО	ОПИСАНИЕ
ccRecipients	Строка адреса электронной почты, разделенная точками с запятой (;) для добавления к полю "Копия" сообщения.
htmlBody	Документ HTML, который задает текст сообщения электронной почты. Сведения о поддерживаемых элементах и атрибутах HTML приведены в документации по каналу.
importance	Уровень важности сообщения электронной почты. Допустимые значения: <code>high</code> , <code>normal</code> и <code>low</code> . Значение по умолчанию — <code>normal</code> .
subject	Тема сообщения электронной почты. Сведения о требованиях к полю приведены в документации по каналу.
toRecipients	Строка адреса электронной почты, разделенная точками с запятой (;) для добавления к полю "Кому" сообщения.

Исходящие и входящие сообщения между пользователем и Bot могут иметь `channelData` действие, которое содержит объект JSON, свойства которого указаны в предыдущей таблице. В приведенном ниже фрагменте показан пример `channelData` свойства для входящего электронного сообщения электронной почты от робота к пользователю.

СВОЙСТВО	ОПИСАНИЕ
htmlBody	HTML-код, используемый для текста сообщения.
subject	Тема сообщения.
importance	Флаг важности, используемый для сообщения: <code>low</code> , <code>normal</code> или <code>high</code> .
toRecipients	Строка адреса электронной почты, разделенная точками с запятой (;) для добавления к полю "Кому" сообщения.
ccRecipients	Строка адреса электронной почты, разделенная точками с запятой (;) для добавления к полю "Копия" сообщения.
bccRecipients	Строка адреса электронной почты, разделенная точками с запятой (;) для добавления к полю "Скрытая копия" сообщения.

## Создание сообщение Slack с полным контролем

Чтобы создать сообщение Slack с полным контролем, присвойте свойство данных канала для объекта действия объекту JSON, который определяет [сообщения](#), [вложения](#) и (или) [кнопки](#) Slack.

## NOTE

Для включения поддержки кнопок в сообщениях Slack необходимо включить **интерактивные сообщения** при подключении бота к каналу Slack.

В этом фрагменте кода демонстрируется свойство `channelData` для пользовательского сообщения Slack.

```
"channelData": {  
    "text": "Now back in stock! :tada:",  
    "attachments": [  
        {  
            "title": "The Further Adventures of Slackbot",  
            "author_name": "Stanford S. Strickland",  
            "author_icon": "https://api.slack.com/img/api/homepage_custom_integrations-2x.png",  
            "image_url": "http://i.imgur.com/OJkaVOI.jpg?1"  
        },  
        {  
            "fields": [  
                {  
                    "title": "Volume",  
                    "value": "1",  
                    "short": true  
                },  
                {  
                    "title": "Issue",  
                    "value": "3",  
                    "short": true  
                }  
            ]  
        },  
        {  
            "title": "Synopsis",  
            "text": "After @episod pushed exciting changes to a devious new branch back in Issue 1, Slackbot  
notifies @don about an unexpected deploy..."  
        },  
        {  
            "fallback": "Would you recommend it to customers?",  
            "title": "Would you recommend it to customers?",  
            "callback_id": "comic_1234_xyz",  
            "color": "#3AA3E3",  
            "attachment_type": "default",  
            "actions": [  
                {  
                    "name": "recommend",  
                    "text": "Recommend",  
                    "type": "button",  
                    "value": "recommend"  
                },  
                {  
                    "name": "no",  
                    "text": "No",  
                    "type": "button",  
                    "value": "bad"  
                }  
            ]  
        }  
    ]  
}
```

Когда пользователь нажмет кнопку в сообщении Slack, бот получит ответное сообщение, в котором свойство данных канала содержит объект JSON `payload`. Этот объект `payload` определяет содержимое исходного сообщения, нажатую кнопку и идентификатор пользователя, который нажал эту кнопку.

В этом фрагменте кода показан пример свойства `channelData` в сообщении, которое бот получает при нажатии кнопки в сообщении Slack.

```
"channelData": {  
    "payload": {  
        "actions": [  
            {  
                "name": "recommend",  
                "value": "yes"  
            }  
        ],  
        . . .  
        "original_message": "...",  
        "response_url": "https://hooks.slack.com/actions/..."  
    }  
}
```

Бот может ответить на это сообщение обычным способом или отправить ответ напрямую на конечную точку, которая определена свойством `response_url` объекта `payload`. Сведения том, как и в каких случаях следует отправлять ответ для `response_url`, см. в документации по [кнопкам Slack](#).

Чтобы создать динамические кнопки, можно использовать следующий код JSON:

```
{  
    "text": "Would you like to play a game ? ",  
    "attachments": [  
        {  
            "text": "Choose a game to play!",  
            "fallback": "You are unable to choose a game",  
            "callback_id": "wopr_game",  
            "color": "#3AA3E3",  
            "attachment_type": "default",  
            "actions": [  
                {  
                    "name": "game",  
                    "text": "Chess",  
                    "type": "button",  
                    "value": "chess"  
                },  
                {  
                    "name": "game",  
                    "text": "Falken's Maze",  
                    "type": "button",  
                    "value": "maze"  
                },  
                {  
                    "name": "game",  
                    "text": "Thermonuclear War",  
                    "style": "danger",  
                    "type": "button",  
                    "value": "war",  
                    "confirm": {  
                        "title": "Are you sure?",  
                        "text": "Wouldn't you prefer a good game of chess?",  
                        "ok_text": "Yes",  
                        "dismiss_text": "No"  
                    }  
                }  
            ]  
        }  
    ]  
}
```

Чтобы создать интерактивные меню, используйте такой код JSON:

```
{  
    "text": "Would you like to play a game ? ",  
    "response_type": "in_channel",  
    "attachments": [  
        {  
            "text": "Choose a game to play",  
            "fallback": "If you could read this message, you'd be choosing something fun to do right now.",  
            "color": "#3AA3E3",  
            "attachment_type": "default",  
            "callback_id": "game_selection",  
            "actions": [  
                {  
                    "name": "games_list",  
                    "text": "Pick a game...",  
                    "type": "select",  
                    "options": [  
                        {  
                            "text": "Hearts",  
                            "value": "menu_id_hearts"  
                        },  
                        {  
                            "text": "Bridge",  
                            "value": "menu_id_bridge"  
                        },  
                        {  
                            "text": "Checkers",  
                            "value": "menu_id_checkers"  
                        },  
                        {  
                            "text": "Chess",  
                            "value": "menu_id_chess"  
                        },  
                        {  
                            "text": "Poker",  
                            "value": "menu_id_poker"  
                        },  
                        {  
                            "text": "Falken's Maze",  
                            "value": "menu_id_maze"  
                        },  
                        {  
                            "text": "Global Thermonuclear War",  
                            "value": "menu_id_war"  
                        }  
                    ]  
                }  
            ]  
        }  
    ]  
}
```

## Создание оповещения Facebook

Чтобы создать оповещение Facebook, присвойте свойству данных канала для объекта действия объект JSON, содержащий следующие свойства:

СВОЙСТВО	ОПИСАНИЕ
notification_type	Тип уведомления (например, REGULAR, SILENT_PUSH, NO_PUSH).

СВОЙСТВО	ОПИСАНИЕ
attachment	Вложение, которое содержит изображение, видео или другой тип мультимедиа, уведомление или другое шаблонное вложение, например квитанция.

#### NOTE

Дополнительные сведения о формате и содержании свойств `notification_type` И `attachment` СМ. В документации по [API Facebook](#).

В этом фрагменте кода демонстрируется свойство `channelData` для вложения Facebook.

```
"channelData": {
    "notification_type": "NO_PUSH",
    "attachment": {
        "type": "template"
        "payload": {
            "template_type": "receipt",
            ...
        }
    }
}
```

## Создание сообщения Telegram

Чтобы создать сообщение, которое реализует специальные действия Telegram, например предоставление в совместный доступ голосового напоминания или наклейки, присвойте свойству данных канала для объекта действия объект JSON, который определяет следующие свойства:

СВОЙСТВО	ОПИСАНИЕ
method	Вызываемый метод API Telegram Bot.
параметры	Параметры указанного метода.

Поддерживаются следующие методы Telegram:

- `answerInlineQuery`
- `editMessageCaption`
- `editMessageReplyMarkup`
- `editMessageText`
- `forwardMessage`
- `kickChatMember`
- `sendAudio`
- `sendChatAction`
- `sendContact`
- `sendDocument`
- `sendLocation`
- `sendMessage`
- `sendPhoto`
- `sendSticker`

- sendVenue
- sendVideo
- sendVoice
- unbanChateMember

Дополнительные сведения об этих методах Telegram и их параметрах см. в документации по [API Telegram Bot](#).

#### NOTE

- Параметр `chat_id` применяется во всех методах Telegram. Если вы не укажете `chat_id` в числе параметров, платформа автоматически создаст идентификатор.
- Чтобы не передавать содержимое файла прямо в запросе, включите ссылку на файл через URL-адрес и тип носителя, как показано в следующем примере.
- В каждом сообщении, которое бот получает из канала Telegram, свойство `ChannelData` содержит отправленное ранее сообщение.

В этом фрагменте кода показан пример свойства `channelData`, которое определяет один метод Telegram.

```
"channelData": {
    "method": "sendSticker",
    "parameters": {
        "sticker": {
            "url": "https://domain.com/path/gif",
            "mediaType": "image/gif",
        }
    }
}
```

В этом фрагменте кода показан пример свойства `channelData`, которое определяет массив методов Telegram.

```
"channelData": [
    {
        "method": "sendSticker",
        "parameters": {
            "sticker": {
                "url": "https://domain.com/path/gif",
                "mediaType": "image/gif",
            }
        }
    },
    {
        "method": "sendMessage",
        "parameters": {
            "text": "<b>This message is HTML formatted.</b>",
            "parse_mode": "HTML"
        }
    }
]
```

При реализации метода Telegram бот получит ответное сообщение, в котором свойство данных канала содержит объект JSON. Этот объект ответа определяет содержимое исходного сообщения, включая `update_id` и не более одного необязательного параметра. См. сведения о получении входящих ответов в руководстве по [получению обновлений](#).

В этом фрагменте кода показан пример свойства `channelData` в сообщении, которое бот получает при

создании опроса.

```
"channelData": {  
    "update_id": 43517575,  
    "message": {  
        "message_id": 618,  
        "from": {  
            "id": 803613355,  
            "is_bot": false,  
            "first_name": "Joe",  
            "last_name": "Doe",  
            "username": "jdoe",  
            "language_code": "en"  
        },  
        "chat": {  
            "id": 803613355,  
            "first_name": "Joe",  
            "last_name": "Doe",  
            "username": "jdoe",  
            "type": "private"  
        },  
        "date": 1582577834,  
        "poll": {  
            "id": "5089525250643722242",  
            "question": "How to win?",  
            "options": [  
                {  
                    "text": "Be the best",  
                    "voter_count": 0  
                },  
                {  
                    "text": "Help those in need",  
                    "voter_count": 0  
                },  
                {  
                    "text": "All of the above",  
                    "voter_count": 0  
                }  
            ],  
            "total_voter_count": 0,  
            "is_closed": false,  
            "is_anonymous": true,  
            "type": "regular",  
            "allows_multiple_answers": false  
        }  
    }  
}
```

## Создание собственного сообщения Kik

Чтобы создать собственное сообщение Kik, присвойте свойству данных канала для объекта действия объект JSON, содержащий следующие свойства:

СВОЙСТВО	ОПИСАНИЕ
отправляемых из облака на устройство	Массив сообщений Kik. См. дополнительные сведения о <a href="#">формате сообщений Kik</a> .

В этом фрагменте кода демонстрируется свойство `channelData` для собственного сообщения Kik.

```

"channelData": {
    "messages": [
        {
            "chatId": "c6dd8165...",
            "type": "link",
            "to": "kikhandle",
            "title": "My Webpage",
            "text": "Some text to display",
            "url": "http://botframework.com",
            "picUrl": "http://lorempixel.com/400/200/",
            "attribution": {
                "name": "My App",
                "iconUrl": "http://lorempixel.com/50/50/"
            },
            "noForward": true,
            "kikJsData": {
                "key": "value"
            }
        }
    ]
}

```

## Создание сообщения LINE

Чтобы создать сообщение, которое реализует особые типы сообщений (например, наклейки, шаблоны или связанные с сообщениями LINE действия, включая открытие камеры на телефоне), настройте для свойства данных канала объект действия объект JSON, который определяет типы сообщений LINE и типы действий.

СВОЙСТВО	ОПИСАНИЕ
type	Имя типа сообщения или действия LINE

Поддерживаются следующие типы сообщений LINE:

- наклейка;
- гиперкарта;
- шаблон (кнопка, подтверждение, карусель);
- общая панель.

Эти действия LINE можно указать в поле действия объекта JSON типа сообщения:

- обратная передача;
- Сообщение
- URI
- средство выбора даты и времени;
- Camera
- галерея камеры;
- Расположение

Дополнительные сведения об этих методах LINE и их параметрах см. в документации по [API LINE Bot](#).

В этом фрагменте кода показан пример свойства `channelData`, которое определяет тип сообщения канала `ButtonTemplate` и три типа действий: камера, галерея камеры и средство выбора даты и времени.

```
"channelData": {
    "type": "ButtonsTemplate",
    "altText": "This is a buttons template",
    "template": {
        "type": "buttons",
        "thumbnailImageUrl": "https://example.com/bot/images/image.jpg",
        "imageAspectRatio": "rectangle",
        "imageSize": "cover",
        "imageBackgroundColor": "#FFFFFF",
        "title": "Menu",
        "text": "Please select",
        "defaultAction": {
            "type": "uri",
            "label": "View detail",
            "uri": "http://example.com/page/123"
        },
        "actions": [
            {
                "type": "cameraRoll",
                "label": "Camera roll"
            },
            {
                "type": "camera",
                "label": "Camera"
            },
            {
                "type": "datetimepicker",
                "label": "Select date",
                "data": "storeId=12345",
                "mode": "datetime",
                "initial": "2017-12-25T00:00",
                "max": "2018-01-24T23:59",
                "min": "2017-12-25T00:00"
            }
        ]
    }
}
```

## Подключение бота к Teams

Добавленный в группу бот становится участником группы, которого можно упоминать (`@mentioned`) в ходе диалога. Фактически, боты получают сообщения, только если они упоминаются (`@mentioned`), следовательно, другие диалоги в канале им не отправляются. См. сведения о [ведении бесед в групповых чатах и каналах с помощью бота Microsoft Teams](#).

Так как боты в группе или канале отвечают, только если они упоминаются (`@botname`) в сообщении, каждое сообщение, полученное ботом в канале группы, содержит собственное имя, которое должно распознаваться при анализе. Кроме того, боты могут распознавать имена других упомянутых пользователей и в свою очередь упоминать их в своих сообщениях.

### Проверка и удаление упоминания `@bot`

```
Mention[] m = sourceMessage.GetMentions();
var messageText = sourceMessage.Text;

for (int i = 0;i < m.Length;i++)
{
    if (m[i].Mentioned.Id == sourceMessage.Recipient.Id)
    {
        //Bot is in the @mention list.
        //The below example will strip the bot name out of the message, so you can parse it as if it wasn't
        included. Note that the Text object will contain the full bot name, if applicable.
        if (m[i].Text != null)
            messageText = messageText.Replace(m[i].Text, "");
    }
}
```

```
var text = message.text;
if (message.entities) {
    message.entities
        .filter(entity => ((entity.type === "mention") && (entity.mentioned.id.toLowerCase() === botId)))
        .forEach(entity => {
            text = text.replace(entity.text, "");
        });
    text = text.trim();
}
```

#### IMPORTANT

Добавлять бота по GUID рекомендуется только для тестирования. В противном случае функциональность бота будет существенно ограничена. Боты, используемые в рабочей среде, следует добавлять в Teams как часть приложения. См. сведения о [создании бота и тестировании и отладке бота Microsoft Teams](#).

## Дополнительные ресурсы

- [Сущности и типы действий](#)
- [Принципы использования действий в Bot Framework](#)

# Подключение бота к Кортане

27.03.2021 • 2 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

## IMPORTANT

Канал Кортаны был списан для всех клиентов 31 января 2021.

Список поддерживаемых каналов см. в статье [Подключение канала Bot к каналам](#).

# Сведения о Direct Line

27.03.2021 • 3 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Платформа Bot предлагает несколько каналов с прямой фирменной символикой. Важно выбрать версию, которая лучше соответствует разрабатываемому интерфейсу искусственного интеллекта.

- **Прямая линия.** Это стандартное предложение прямой линии для канала. Он работает по умолчанию с шаблонами Bot через [портал Azure](#), программы-роботы из [Bot Builder](#) примеровы программы-роботы, созданных с помощью [Azure CLI](#). Это прямая линия, которая лучше всего подходит в большинстве случаев. См. раздел [Подключение программы-робота к прямой линии](#).
- **Прямое распознавание речи.** Он обеспечивает распознавание текста и речи для текстовых служб в канале. Он позволяет клиенту выполнять потоковую передачу аудио непосредственно в канал, который затем преобразуется в текст и отправляется в Bot. Голосовая речь также может преобразовать текстовые сообщения из программы-робота в звуковые сообщения в качестве речи с помощью различных А.И. включенные голоса. В сочетании это позволяет выполнять голосовые беседы с клиентами напрямую. См. раздел [Подключение программы-робота к прямой речи](#).
- **Расширение службы прямого построчного приложения.** Она выполняется в той же подписке, в службе приложений и в сети Azure, что и программа-робот. Если у вас есть требования к сетевой изоляции, эта версия прямой строки может быть идеальной. Программы-роботы и клиенты нуждаются в специальных изменениях для работы с расширением прямого построчного приложения для обеспечения того, что трафик никогда не покидает изолированную сеть. См. раздел [расширение службы приложений Direct Line](#).

Вы можете выбрать для себя оптимальный вариант Direct Line, сравнивая возможности каждого предложения с потребностями своего решения. Со временем эти предложения будут упрощены.

ФУНКЦИЯ	DIRECT LINE	РАСШИРЕНИЕ СЛУЖБЫ ПРИЛОЖЕНИЙ DIRECT LINE	КАНАЛ DIRECT LINE SPEECH
Доступность и лицензирование	GA	GA	GA
Производительность распознавания речи и преобразования текста в речь	Standard	Standard	Высокопроизводительные
Поддержка устаревших веб-браузеров	Да	Да	Да
Поддержка пакета SDK Bot Framework	Все версии 3 и 4	Требуется более поздняя версия, чем 4.63	Требуется более поздняя версия, чем 4.63
Поддержка клиентских пакетов SDK	JS, C#	JS, C#	C++, C#, Unity, JS
Работа с Web Chat	Да	Да	Да
Виртуальная сеть	нет	Да	нет

## Дополнительные ресурсы

- [Подключение бота к Direct Line](#)
- [Подключение бота к каналу Direct Line Speech](#)
- [Расширение Службы приложений Direct Line](#)
- [Использование веб-разговора с прямыми голосовыми голосами](#)

# Подключение бота к службе Direct Line

27.03.2021 • 7 minutes to read • [Edit Online](#)

В этой статье описывается, как подключить робот к **каналу прямой линии**. Используйте этот канал, чтобы разрешить клиентскому приложению обмениваться данными с Bot.

## NOTE

Прямая линия представляет собой стандартный канал по протоколу HTTPS, обеспечивающий взаимодействие между клиентским приложением и программой-роботом. Если вместо этого требуется сетевая изоляция, используйте [расширение службы приложений Direct Line для протокола WebSocket](#).

## Добавление канала Direct Line

В первую очередь необходимо добавить прямой канал линии в Bot.

1. В браузере перейдите на [портал Azure](#).
2. На левой панели щелкните элемент **каналы**.
3. На правой панели в разделе *Добавление избранного канала* щелкните значок **прямой строки** (отмечен красным цветом на рисунке ниже).

Connect to channels

Name	Health	Published	
Web Chat	Running	--	<a href="#">Edit</a>

[Get bot embed codes](#)

Add a featured channel

4. Отобразится страница **Настройка прямой линии**. Нажмите кнопку **done (Готово)** в нижней части страницы. Это добавляет прямой канал линии в Bot, как показано на рисунке ниже.

Connect to channels

Name	Health	Published	
Direct Line	Running	--	<a href="#">Edit</a>
Web Chat	Running	--	<a href="#">Edit</a>

[Get bot embed codes](#)

## Добавление нового веб-сайта

1. В окне *Подключение к каналам* щелкните ссылку **Edit (изменить)** прямой линией.
2. В окне *Настройка прямой линии* щелкните **Добавить новый сайт** и введите имя сайта. Представляет клиентское приложение, которое необходимо подключить к Bot.

## Configure Direct Line



+ Add new site Default Site  Disable |

3. Нажмите кнопку Done(Готово).

## Управление секретными ключами

При добавлении прямого канала Bot Framework создает секретные ключи. Клиентское приложение использует эти ключи для проверки подлинности запросов API прямой линии, которые он выдает для взаимодействия с Bot. Дополнительные сведения см. в разделе [Authenticate to the Speech API](#) (Аутентификация в API речи).

1. В разделе *Настройка прямой линии* для просмотра ключа в виде обычного текста нажмите кнопку **Показать** для соответствующего ключа.

## Configure Direct Line



+ Add new site My New Site  Disable |

Default Site  
My New Site

**Secret keys**

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	<b>Show</b>
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	Show

**Version**  
Select which versions of the Direct Line protocol are enabled on this site. More information about these versions can be found in the [Direct Line reference documentation](#).

1.1  
 3.0

**Enhanced authentication options**  
For bots using [Azure Bot Service authentication](#), enable tamper-proof user IDs and the ability to specify trusted client hosts. Learn more about enhanced Direct Line [authentication features](#).

2. Копирование и безопасное хранение ключа. Используйте ключ для [проверки подлинности](#) запросов API прямой линии, которые клиентское приложение выдает для взаимодействия с Bot.

## Configure Direct Line

### NOTE

Секреты не должны предоставляться или внедряться в клиентские приложения. См. следующий шаг.

3. Рекомендуется использовать API прямой линии для [обмена ключами маркера](#). Затем клиентское приложение будет использовать маркер для проверки подлинности своих запросов в области одного диалога.

## Настройка параметров

1. Выберите версию протокола прямой строки, которую клиентское приложение будет использовать для взаимодействия с Bot.

### TIP

При создании подключения между клиентским приложением и ботом, используйте Direct Line API 3.0.

2. Чтобы сохранить конфигурацию веб-сайта, нажмите кнопку **Готово**. Этот процесс можно повторять, начиная с команды [Добавить новый веб-сайт](#), для каждого клиентского приложения, которое необходимо подключить к боту.

## Настройка расширенной проверки подлинности

При включении **расширенной проверки подлинности** вам будет предложено выбрать список **доверенных URL-адресов источника**, также известных как доверенные источники или доверенные домены, для создания маркера проверки подлинности. При включении расширенной проверки подлинности необходимо указать по крайней мере один доверенный источник.

## Add a trusted origin url

Trusted origins must be a https URL unless they are for localhost.

Add a trusted origin url

Cancel

Done

Доверенный домен — это домен, который система доверяет для проверки подлинности пользователей. В нашем случае — это домен, которому прямая линия может доверять созданию маркера.

- Если вы настраиваете Доверенные источники как часть страницы пользовательского интерфейса конфигурации, они **всегда** будут использоваться в качестве единственного набора для создания токена. Отправка None или дополнительных надежных источников при создании маркера или запуске диалога они будут пропущены (т. е. они **не добавляются** в список или перекрестно проверенные).
- Если вы не включили расширенную проверку подлинности, будет использоваться любой исходный URL-адрес, отправляемый в рамках вызовов API.

Улучшенная проверка подлинности позволяет снизить риски безопасности при подключении к Bot (например, с помощью элемента управления веб-чата). Дополнительные сведения см. в разделе [Прямая строка Расширенная проверка подлинности](#).

## Пример

Пример .NET можно загрузить из этого расположения: [пример прямой линии Bot](#).

Пример содержит два проекта:

- **Директлинебот**. Он создает робот для подключения через прямой канал линии.
- **Директлинеклиент**. Это консольное приложение, которое обращается к предыдущей программе Bot через прямой канал.

### Direct Line API

- Учетные данные для API прямой линии должны быть получены из регистрации каналов Bot или с помощью робота веб-приложения в портал Azure и разрешать вызывающему объекту подключаться только к роботу, для которого они были созданы. В проекте Bot обновите файл, `appsettings.json` используя эти значения.

```
{  
  "MicrosoftAppId": "",  
  "MicrosoftAppPassword": ""  
}
```

- В портал Azure включите прямую линию в списке каналы, а затем настройте прямой секрет строки.

Убедитесь, что установлен флажок для версии 3,0. В проекте консольного клиента обновите

`App.config` файл с помощью прямого строки секретного ключа и маркера Bot (идентификатор Bot).

```
<appSettings>  
  <add key="DirectLineSecret" value="YourBotDirectLineSecret" />  
  <add key="BotId" value="YourBotHandle" />  
</appSettings>
```

Пользовательские сообщения отправляются в Bot с помощью клиентского метода Direct Line

`Conversations.PostActivityAsync` , `ConversationId` созданного ранее.

```
while (true)
{
    string input = Console.ReadLine().Trim();

    if (input.ToLower() == "exit")
    {
        break;
    }
    else
    {
        if (input.Length > 0)
        {
            Activity userMessage = new Activity
            {
                From = new ChannelAccount(fromUser),
                Text = input,
                Type = ActivityTypes.Message
            };

            await client.Conversations.PostActivityAsync(conversation.ConversationId, userMessage);
        }
    }
}
```

# Подключение бота к каналу Direct Line Speech

27.03.2021 • 5 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье описывается подключение программы-робота к **каналу голосовой речи**. Используйте этот канал, чтобы разрешить пользователям взаимодействовать с Bot через Voice.

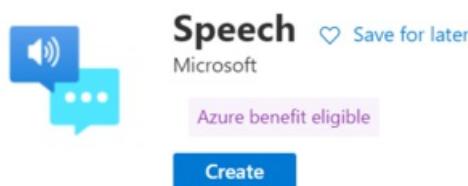
После того, как вы создали программу-робот, ее внедрение с помощью прямого перевода строки обеспечит высокую задержку и высоконадежную связь с клиентскими приложениями, использующими **речевой пакет SDK**. Эти подключения оптимизируются для речевого взаимодействия и общения.

Дополнительные сведения о Direct Line Speech и создании клиентских приложений см. в [описании пользовательского виртуального голосового помощника](#).

## Предварительные условия

Для канала речевого перевода строки требуется ресурс Cognitive Services , в частности ресурс службы **распознавания речи** . Вы можете выбрать существующий ресурс ресурсов или создать новый. Чтобы создать новый ресурс, сделайте следующее:

1. В браузере перейдите к [портал Azure](#) , чтобы создать ресурсы.
2. На панели слева щелкните **создать ресурс**.
3. На панели справа введите **речь** в поле поиска.
4. В раскрывающемся списке выберите **речь**, в котором будет открыта страница с описанием этого типа ресурсов.

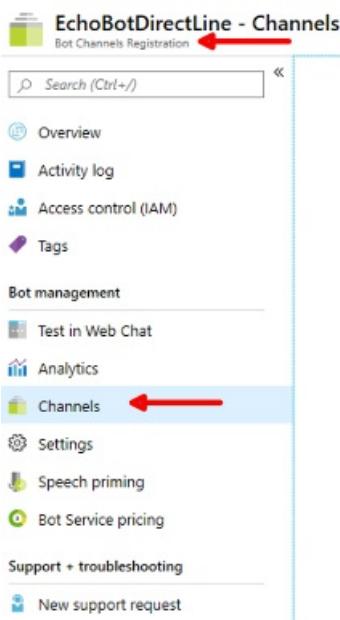


5. Щелкните **создать** и следуйте инструкциям мастера.

См. сведения о [создании ресурса Cognitive Services](#).

## Добавление канала Direct Line Speech

1. В браузере перейдите на [портал Azure](#).
2. В зависимости от того, как вы развернули программу-робот, выберите в своих ресурсах ресурс **регистрации канала** Bot или Web App Bot .
3. На панели слева выберите **каналы**.



4. На правой панели щелкните значок **прямой текст в речевом** окне.

The screenshot shows the 'Connect to channels' page. It lists existing channels: 'Direct Line' (Running) and 'Web Chat' (Running). Below this is a section titled 'Add a featured channel' with icons for Microsoft Teams, Direct Line, and Direct Line Speech. At the bottom is a 'More channels' section with 'Direct Line Speech' highlighted by a red arrow. Other options include 'Email' and 'Facebook'.

5. Настройте Direct Line Speech, как показано на рисунке ниже. В частности, добавьте учетную запись службы для **распознавания речи**, упомянутую в разделе [Предварительные требования](#).

The screenshot shows the configuration page for Direct Line Speech. It features a central icon with a speech bubble and a double-headed arrow. Below it are three input fields: 'Cognitive service account' (with a dropdown menu), 'Custom speech model id (optional)' (with a 'Copy model id from custom speech deployment' link), and 'Custom voice deployment id (optional)' (with a 'Copy deployment id from custom voice deployment' link).

6. Ознакомьтесь с условиями использования и щелкните **Save**, чтобы подтвердить выбор канала. Это приведет к добавлению канала в Bot.

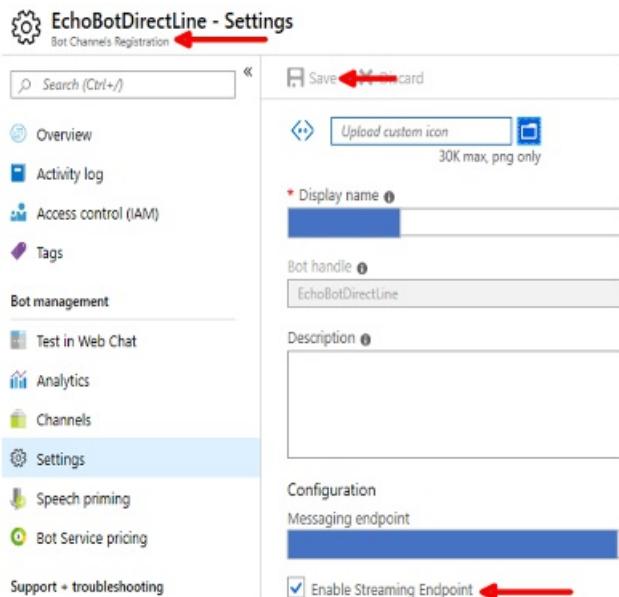
## Connect to channels

Name	Health	Published	
🌐 Direct Line	Running	--	Edit
🎙️ Direct Line Speech	Running	--	Edit
💬 Web Chat	Running	--	Edit

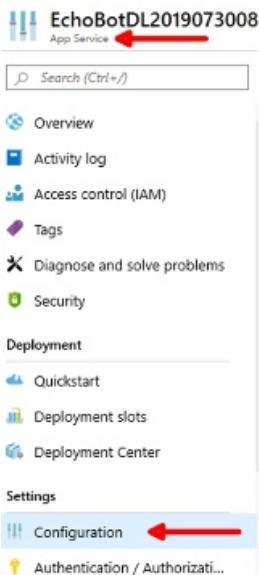
## Включение расширений протокола потоковой передачи Bot Framework

Подключив канал Direct Line Speech к боту, следует включить поддержку протокола потоковой передачи Bot Framework, чтобы организовать оптимальное взаимодействие с низкой задержкой.

1. На панели слева выберите **Параметры**.
2. На панели справа установите флажок **включить конечную точку потоковой передачи**.

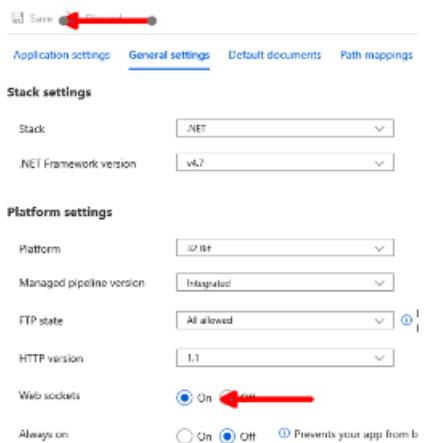


3. В верхней части страницы нажмите кнопку **Сохранить**.
4. Перейдите в службу приложений Bot.
5. На панели слева в категории **Параметры службы приложений** выберите **Конфигурация**.



6. На правой панели выберите **General settings** вкладку.

7. Значение **Web sockets** **On**.



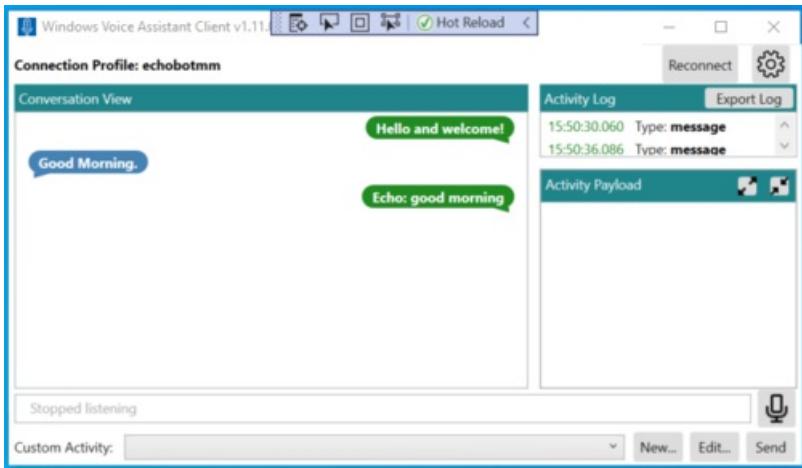
8. Щелкните **Save** в верхней части страницы конфигурации.

9. Теперь для бота включены расширения протокола потоковой передачи Bot Framework. Теперь переходите к обновлению кода для бота, чтобы [интегрировать поддержку расширения потоковой передачи](#) в существующий проект бота.

## Пример

Если вы выполнили все описанные действия, вы можете обратиться к Bot, используя клиентское приложение, загружаемое в этом расположении: [клиент помощника по Windows Voice](#).

На следующем рисунке показан интерфейс клиентского приложения при взаимодействии с простым роботом Echo. См. также [голосовую поддержку для программы Bot с помощью пакета SDK для распознавания речи](#).



## Добавление поддержки протокола в бот

### NOTE

Следующий шаг необходим только для программы-роботы, созданных до выпуска пакетов SDK 4.8.

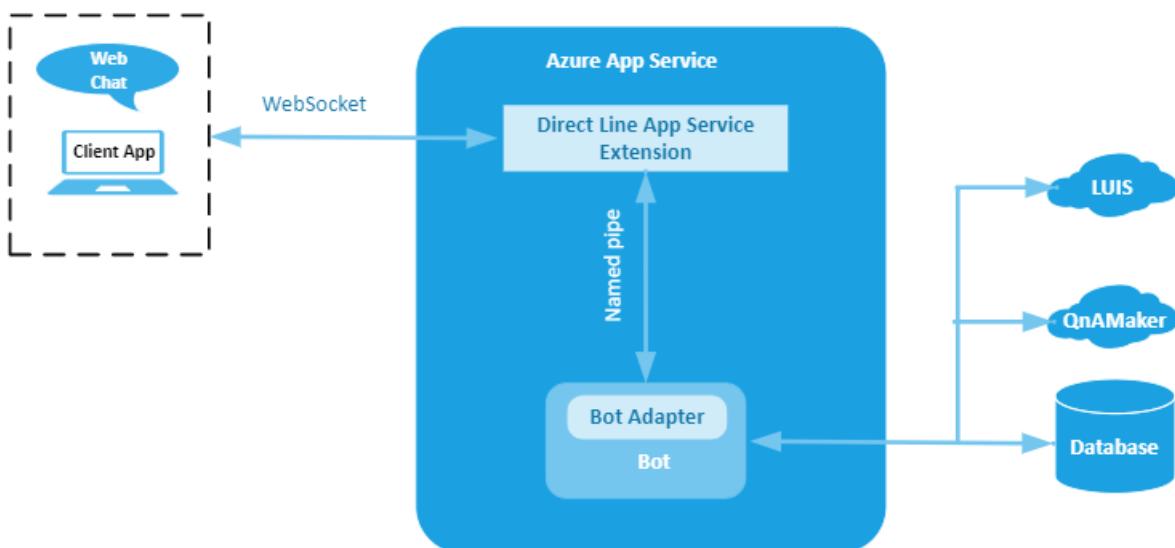
Завершив подключение канала Direct Line Speech и настройку поддержки протокола потоковой передачи Bot Framework, остается лишь добавить в бот код для поддержки оптимизированного режима связи. Следуйте инструкциям по [добавлению поддержки расширений потоковой передачи](#), чтобы обеспечить полную совместимость с Direct Line Speech.

# Расширение Службы приложений Direct Line

27.03.2021 • 3 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Расширение Службы приложений Direct Line позволяет клиентам подключаться напрямую к узлу, на котором находится бот. Она выполняется в той же подписке, в службе приложений и в сети Azure, что и программа-робот, и обеспечивает изоляцию сети и, в некоторых случаях, повышает производительность. Клиентское приложение использует [протокол WebSocket](#) для взаимодействия с Bot. На следующем рисунке показана общая архитектура расширения:



## NOTE

Если не требуется сетевая изоляция и вы хотите использовать стандартный канал по протоколу HTTPS, обратитесь к статье [Подключение программы-робота к прямой линии](#).

Расширение службы прямого потока приложений добавляет новый набор потоковых расширений к протоколу Bot Framework, который заменяет HTTP для обмена сообщениями с помощью транспорта, позволяющего отправлять двунаправленные запросы через *Постоянный сокет WebSocket*.

До применения расширений потоковой передачи интерфейс API Direct Line предоставлял клиенту один способ отправлять действия в Direct Line и два способа получать действия от Direct Line. Сообщения при этом можно было отправлять в HTTP-запросе POST и получать в HTTP-запросе GET (метод опроса) или путем открытия WebSocket для получения ActivitySets. Расширения потоковой передачи расширяют использование WebSocket, позволяя *отправлять все сообщения* через этот WebSocket. Расширения потоковой передачи можно также использовать между службами каналов и ботом.

Расширение службы прямого направления приложений предварительно установлено на всех экземплярах служб приложений Azure в каждом центре обработки данных по всему миру. Оно поддерживается и управляется корпорацией Майкрософт, и клиентам не нужно выполнять никаких действий для его развертывания. По умолчанию оно отключен в Службах приложений Azure, но его можно легко включить и настроить для подключения к размещенному боту.

**См. также:**

Имя	Описание
<a href="#">Использование бота .NET с расширением</a>	Реализуйте в боте .NET поддержку работы с именованными каналами и включите расширение Direct Line Службы приложений в ресурсе Службы приложений Azure с размещенным ботом.
<a href="#">Настройка бота Node.js для использования расширения</a>	Реализуйте в боте Node.js поддержку работы с именованными каналами и включите расширение Direct Line Службы приложений в ресурсе Службы приложений Azure с размещенным ботом.
<a href="#">Создание клиента .NET с использованием расширения</a>	Создайте клиент .NET на языке C#, который подключается к расширению службы приложений Direct Line.
<a href="#">Использование расширения с веб-чат</a>	Используйте веб-чат с расширением прямой службы приложений.
<a href="#">Использование расширения в виртуальной сети</a>	Используйте расширение службы приложений Direct Line с виртуальной сетью Azure.

## Дополнительные ресурсы

- [Подключение бота к Direct Line](#)
- [Подключение бота к каналу Direct Line Speech](#)

# Настройка бота .NET для использования расширения

27.03.2021 • 8 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье описывается, как обновить Bot для работы с именованными каналами, а также как включить расширение службы Direct Line в ресурсе службы приложений Azure, в котором размещена программа Bot.

## Предварительные требования

Чтобы выполнить описанные далее действия, необходимо создать в Azure робот веб-приложения.

## Включить расширение службы прямого построчного приложения

В этом разделе описывается, как включить расширение службы прямого направления приложений с помощью ключа расширения службы приложений из конфигурации канала прямой линии Bot.

### Обновление кода Bot

#### NOTE

Пакеты предварительной версии `Microsoft.Bot.Builder.Streaming` NuGet являются устаревшими. Начиная с версии v 4.8, пакет SDK содержит пространство имен `Microsoft.Bot.Builder.Streaming`. Если бот ранее использовал предварительные версии пакетов, их необходимо удалить, прежде чем выполнять следующие шаги.

1. Откройте проект бота в Visual Studio.
2. Убедитесь, что проект использует версию 4.8 или более позднюю версию пакета SDK для Bot Framework.
3. Разрешить приложению использовать именованные каналы:
  - Откройте файл `Startup.cs`.
  - В `Configure` методе добавьте вызов `UseNamedPipes` метода.

```

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseHsts();
    }

    app.UseDefaultFiles();
    app.UseStaticFiles();

    // Allow the bot to use named pipes.
    app.UseNamedPipes(System.Environment.GetEnvironmentVariable("APPSETTING_WEBSITE_SITE_NAME") +
".directline");

    app.UseMvc();
}

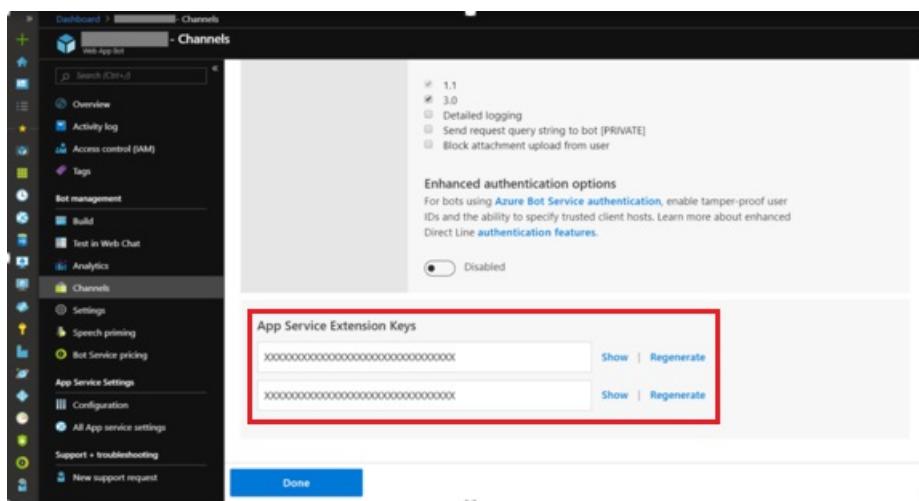
```

4. Сохраните файл Startup.cs .

5. Опубликуйте программу Bot в ресурсе робота веб-приложения Azure, чтобы развернуть обновленный код.

#### **Включение расширения службы приложений Bot Direct Line**

1. В портал Azure выберите ресурс **веб-приложения Bot** .
2. В меню с левой панелью в разделе "Управление программой- Bot " щелкните **каналы** , чтобы настроить каналы **службы Azure Bot** , принимающие сообщения от ленты.
3. Если он еще не включен, щелкните **прямой канал линии** и следуйте инструкциям, чтобы включить канал.
4. В таблице **Подключение к каналам** щелкните ссылку **Edit (изменить)** в строке **прямой** .
5. Прокрутите вниз до раздела **ключи расширения службы приложений** .
6. Щелкните ссылку **Показать** , чтобы открыть один из ключей. Скопируйте это значение для дальнейшего использования.



7. Перейдите на домашнюю страницу, щелкните значок **службы приложений** в верхней части страницы. Также можно отобразить меню портала, а затем щелкнуть пункт меню **службы приложений** на панели слева. Откроется страница **службы приложений** .

8. В поле поиска введите имя ресурса **Bot веб-приложения**. Ваш ресурс будет указан в списке. Обратите внимание, что при наведении указателя мыши на значок или пункт меню вы получаете список последних просмотренных ресурсов. Скорее всего, будет указан ресурс **Bot веб-приложения**.

9. Щелкните ссылку на ресурс.

10. В разделе **Параметры** щелкните элемент меню **Конфигурация**.

11. На панели справа добавьте следующие новые параметры.

Имя	Значение
DirectLineExtensionKey	Значение скопированного ранее ключа расширения службы приложений.
DIRECTLINE_EXTENSION_VERSION	последняя

12. Если программа Bot размещена в независимых или в другом ограниченном облаке Azure, где вы не будете получать доступ к Azure через [общедоступный портал](#), потребуется также добавить следующий новый параметр:

Имя	Значение
DirectLineExtensionABSEndpoint	Конечная точка, относящаяся к облаку Azure, в котором размещается Bot. Например, для облака государственных организаций США в качестве конечной точки используется <code>https://directline.botframework.azure.us/v3/extension</code>

13. По-прежнему в разделе **конфигурации** щелкните раздел **Общие** параметры и включите **веб-сокеты**.

14. Нажмите кнопку **Сохранить**, чтобы сохранить параметры. Эта команда перезапускает Службу приложений Azure.

## Убедитесь, что расширение и Bot настроены.

В браузере перейдите по адресу `https://<your_app_service>.azurewebsites.net/.bot`. Если все настроено правильно, вы увидите такое содержимое JSON:

`{"v": "123", "k": true, "ib": true, "ob": true, "initialized": true}`. В этих данных, которые указывают на то, что *все работает правильно*:

- **v** отображает версию сборки расширения службы приложений Direct Line.
- **k** определяет, может ли расширение прочитать ключ расширения из его конфигурации.
- **инициализировано** определяет, может ли расширение использовать ключ расширения для загрузки метаданных Bot из службы Azure Bot.
- **геообъект определяет**, может ли расширение устанавливать входящее соединение с Bot.
- **Ob** определяет, может ли расширение устанавливать исходящее соединение с Bot.

## Устранение неполадок

- Если значения "**Гео**" и "**Ob**", отображаемые **конечной точкой** Bot, имеют значение false, это означает, что робот и расширение прямой службы приложений не могут подключаться друг к другу.

1. Дважды проверьте код для использования именованных каналов, добавленный в Bot.
  2. Убедитесь, что программа Bot может запускаться и работать вообще. Полезными средствами являются **тестирование в веб-обсуждении**, подключение дополнительного канала, удаленная отладка или ведение журнала.
  3. Перезапустите всю **службу приложений Azure**, которая размещена на сервере Bot, чтобы обеспечить чистый запуск всех процессов.
- Если вы получаете сообщение об ошибке HTTP 500,34-ANCM Mixed Hosting, программа Bot пытается использовать `InProcess` модель размещения. Это исправлено путем явного задания запуска программы-робота `OutOfProcess`. Дополнительные сведения см. в статье о [модели размещения вне процесса](#) в основной документации по AZP.NET.
  - Если **инициализированное значение конечной точки** Bot равно `false`, это означает, что расширению службы приложений с прямыми линиями не удается проверить ключ расширения службы приложений, добавленный в **Параметры приложения** Bot выше.
    1. Подтвердите, что значение введено правильно.
    2. Переключитесь на альтернативный ключ расширения, показанный на странице **Настройка прямой линии** программы-робота.
  - Если вы попытаетесь использовать OAuth с расширением прямой службы приложений и столкнулись с ошибкой "не удалось получить идентификатор робота из утверждения аудитории".  
Объект `ClaimsIdentity` с `AudienceClaim` назначеными должны быть заданы для `BotFrameworkHttpAdapter`. Для этого разработчик может создать подкласс адаптера, как показано в примере ниже:

```
public class AdapterWithStaticClaimsIdentity : BotFrameworkHttpAdapter
{
    public AdapterWithStaticClaimsIdentity(IConfiguration configuration, ILogger<BotFrameworkHttpAdapter>
logger, ConversationState conversationState = null)
        : base(configuration, logger)
    {
        // Manually create the ClaimsIdentity and create a Claim with a valid AudienceClaim and the AppID
        // for a bot using the Direct Line App Service extension.
        var appId = configuration.GetSection(MicrosoftAppCredentials.MicrosoftAppIdKey)?.Value;
        ClaimsIdentity = new ClaimsIdentity(new List<Claim>{
            new Claim(AuthenticationConstants.AudienceClaim, appId)
        });
    }
}
```

## Следующие шаги

[Использование веб-разговора с расширением прямой службы приложений](#)

# Настройка бота Node.js для работы с расширением

27.03.2021 • 7 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье описывается, как обновить Bot для работы с **именованными каналами**, а также как включить расширение службы Direct Line в ресурсе **службы приложений Azure**, в котором размещена программа Bot.

## Предварительные условия

Чтобы выполнить описанные далее действия, необходимо создать **веб-приложение Bot** (Bot), созданное в Azure.

## Включить расширение службы прямого построчного приложения

В этом разделе описывается, как включить расширение службы прямого направления приложений с помощью ключа расширения службы приложений из конфигурации канала прямой линии Bot.

## Обновление бота Node.js для работы с расширением Direct Line Службы приложений

1. BotBuilder v 4.7.0 или более поздней версии требуется использовать Node.js Bot с расширением службы приложений Direct Line.
2. Разрешите приложению использовать **именованный канал расширения Direct Line Службы приложений**.

Обновите index.js Bot (под назначением адаптера и Bot), чтобы включить следующий код, который извлекает имя службы приложений из среды и указывает адаптеру подключиться к соответствующему именованному каналу:

```
adapter.useNamedPipe(async (context) => {
    await myBot.run(context);
},
process.env.APPSETTING_WEBSITE_SITE_NAME + '.directline'
);
```

3. Сохраните файл `index.js`.
4. Обновите `web.Config` файл, чтобы добавить `AspNetCore` обработчик и правило, необходимые для расширения службы приложений напрямую в запросы на обслуживание:

Укажите `Web.Config` файл в `wwwroot` каталоге программы Bot и измените его содержимое, чтобы включить в `Handlers` разделы и следующие записи `Rules`:

```

<handlers>
    <add name="aspNetCore" path="*.bot/*" verb="*" modules="AspNetCoreModule"
resourceType="Unspecified" />
</handlers>

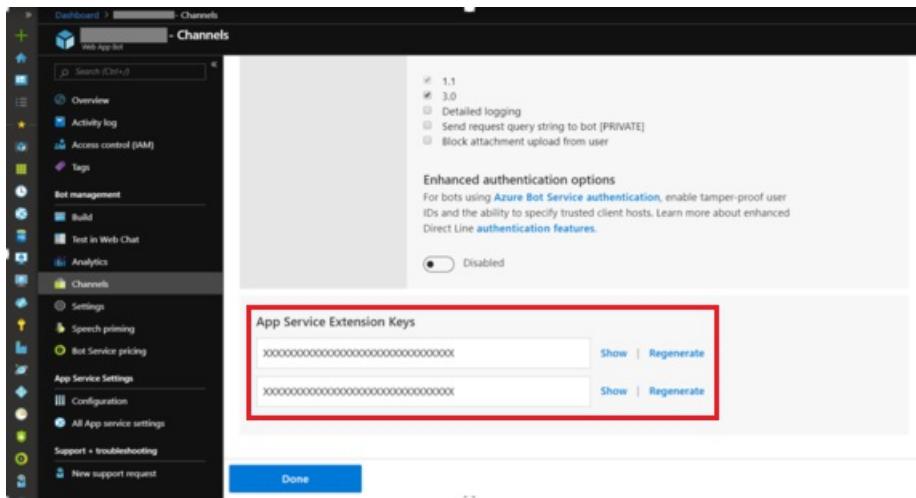
<rewrite>
    <rules>
        <!-- Do not interfere with Direct Line App Service Extension requests. (This rule should be as
high in the rules section as possible to avoid conflicts.) -->
        <rule name ="DLASE" stopProcessing="true">
            <conditions>
                <add input="{REQUEST_URI}" pattern="^/.bot"/>
            </conditions>
        </rule>
    </rules>
</rewrite>

```

## 5. Опубликуйте робот в ресурсе Bot веб-приложения Azure.

### Включение расширения службы приложений Bot Direct Line

1. В портал Azure выберите ресурс **веб-приложения Bot**.
2. В меню с левой панелью в разделе "Управление программой- Bot" щелкните **каналы**, чтобы настроить каналы **службы Azure Bot**, принимающие сообщения от ленты.
3. Если он еще не включен, щелкните **прямой канал линии** и следуйте инструкциям, чтобы включить канал.
4. В таблице **Подключение к каналам** щелкните ссылку **Edit (изменить)** в строке прямой.
5. Прокрутите вниз до раздела **Ключи расширений Службы приложений**.
6. Щелкните ссылку **Показать**, чтобы открыть один из ключей. Это значение будет использоваться в следующих шагах.



7. В меню с левой панелью в разделе **Параметры приложения** щелкните элемент **конфигурации**.
8. На панели справа добавьте следующие новые параметры.

Имя	Значение
DirectLineExtensionKey	<App_Service_Extension_Key>
DIRECTLINE_EXTENSION_VERSION	последняя

Где *App\_Service\_Extension\_Key* — это сохраненное ранее значение.

9. Если программа-робот размещена в независимых или в другом ограниченном облаке Azure (т. е. Вы не будете получать доступ к Azure через [общедоступный портал](#)), потребуется также добавить следующий новый параметр:

Имя	Значение
DirectLineExtensionABSEndpoint	<URL_of_Direct_Line_App_Gateway>

Здесь *URL\_of\_Direct\_Line\_App\_Gateway* относится к облаку Azure, в котором размещается Bot. Для государственных организаций США это значение <https://directline.botframework.azure.us/v3/extension>

10. По-прежнему в разделе *конфигурации* щелкните раздел **Общие** параметры и включите **веб-сокеты**.
11. Нажмите кнопку **Сохранить**, чтобы сохранить параметры. Эта команда перезапускает Службу приложений Azure.

## Подтвердите, что настроено прямое расширение приложения и Bot.

В браузере откройте страницу [https://<ваша\\_служба\\_приложений>.azurewebsites.net/.bot](https://<ваша_служба_приложений>.azurewebsites.net/.bot). Если все настроено правильно, вы увидите такое содержимое JSON:

`{"v": "123", "k": true, "ib": true, "ob": true, "initialized": true}` . В этих данных, которые указывают на то, что **все работает правильно**:

- *v* отображает версию сборки расширения Службы приложений (ASE) Direct Line.
- *k* определяет, может ли ASE Direct Line считывать ключ расширения Службы приложений из своей конфигурации.
- *initialized* определяет, может ли ASE Direct Line с помощью соответствующего ключа расширения скачивать метаданные бота из службы Azure Bot.
- *ib* определяет, может ли ASE Direct Line устанавливать входящее соединение с ботом.
- *ob* определяет, может ли ASE Direct Line устанавливать исходящее соединение с ботом.

## Устранение неполадок

- Если значения "Гео" и "Ob", отображаемые \*конечной точкой .Bot, имеют значение false, это означает, что робот и расширение службы приложений Direct Line не могут подключаться друг к другу.
  1. Дважды проверьте код для использования именованных каналов, добавленный в Bot.
  2. Убедитесь, что программа Bot может запускаться и работать вообще. Полезными средствами являются **тестирование в веб-обсуждении**, подключение дополнительного канала, удаленная отладка или ведение журнала.
  3. Перезапустите всю **службу приложений Azure**, которая размещена на сервере Bot, чтобы обеспечить чистый запуск всех процессов.
- Если **инициализированное значение конечной точки**.Bot равно false, это означает, что расширению службы приложений с прямыми линиями не удается проверить **ключ расширения службы приложений**, добавленный в **Параметры приложения** Bot выше.
  1. Подтвердите, что значение введено правильно.
  2. Переключитесь на альтернативный **ключ расширения службы приложений**, показанный

на странице **настройки канала прямой линии** ленты.

## Дальнейшие действия

[Использование веб-разговора с расширением прямой службы приложений](#)

# Создание клиента .NET для подключения к расширению Службы приложений Direct Line

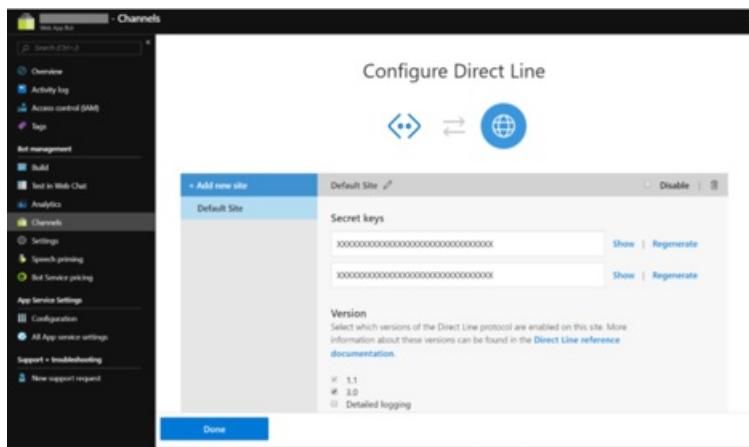
27.03.2021 • 5 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье описано, как создать клиент .NET на языке C#, который подключается к расширению Службы приложений Direct Line. См. также о [настройке бота .NET для расширения](#).

## Получение секретного ключа прямой строки

1. В браузере перейдите на [портал Azure](#).
2. На портале Azure найдите ресурс **службы Azure Bot**.
3. Щелкните **каналы**, чтобы настроить каналы Bot
4. Если канал Direct Line еще не включен, щелкните его, чтобы включить.
5. Если он уже включен, в таблице "Подключение к каналам" щелкните ссылку **Изменить** в строке Direct Line.
6. Прокрутите вниз до раздела "Сайты". Обычно здесь отображается сайт по умолчанию, если вы его не удалили и не переименовали.
7. Щелкните **Показать ссылку**, чтобы отобразить один из ключей, а затем скопируйте и сохраните его значение. Это значение будет использоваться в разделе [Создание клиента прямой линии C#](#).



### NOTE

Клиент Direct Line использует это значение секрета для подключения к расширению Службы приложений Direct Line. Вы можете создать дополнительные сайты, если хотите, а также использовать эти секретные значения.

## Добавление источника пакета NuGet для предварительного просмотра

Пакеты NuGet предварительной версии, необходимые для создания клиента Direct Line на C#, можно получить из веб-канала NuGet.

1. В Visual Studio перейдите к пункту меню **Сервис** -> **Параметры**.
2. Выберите **Диспетчер пакетов NuGet** -> **Источники пакетов**.
3. Нажмите кнопку "+", чтобы добавить новый источник пакета со следующими значениями.
  - Имя: DL ASE Preview
  - Источник: <https://botbuilder.myget.org/F/experimental/api/v3/index.json>
4. Чтобы сохранить новые значения, нажмите кнопку **Обновить**.
5. Щелкните **OK**, чтобы выйти из режима настройки источника пакетов.

## Создание клиента прямой линии C#

Взаимодействие с расширением службы приложений Direct Line существенно отличается от обычной работы с Direct Line, так как значительная часть взаимодействия выполняется через *WebSocket*. Обновленный клиент Direct Line содержит вспомогательные классы для открытия и закрытия *WebSocket*, отправки команд через *WebSocket* и получения действий от бота. В этом разделе описывается, как создать простой клиент C# для взаимодействия с ботом.

1. В Visual Studio создайте новый проект консольного приложения .NET Core 2,2.
2. Добавьте в этот проект **клиент NuGet Direct Line**.
  - Щелкните "Зависимости" в дереве решения.
  - Выберите **Manage NuGet Packages...** (Управление пакетами NuGet...).
  - Измените источник пакета на **DL ASE Preview** (см. раздел [Добавление предварительной версии источника пакета NuGet](#)).
  - Найдите пакет [Microsoft.Bot.Connector.Directline](#) версии v3.0.3-Preview1 или более поздней.
  - Щелкните **Установить пакет**.
3. Создайте клиент и маркер для него с помощью секрета. Этот шаг будет одинаковым для любого клиента Direct Line на C#, не считая значения конечной точки, которое нужно указать в боте с добавлением пути `.bot/`, как показано далее. Не забудьте `/` в конце.

```
string endpoint = "https://<your_bot_name>.azurewebsites.net/.bot/";  
string secret = "<your_bot_direct_line_secret_key>";  
  
var tokenClient = new DirectLineClient(  
    new Uri(endpoint),  
    new DirectLineClientCredentials(secret));  
var conversation = await tokenClient.Tokens.GenerateTokenForNewConversationAsync();
```

Обратите внимание на следующее.

- Значение конечной точки — это URL-адрес бота, полученный при развертывании бота в Azure. См. сведения о [настройке бота .NET для расширения](#).
  - Значение секрета, отображаемое как `YOUR_BOT_SECRET` — это значение, сохраненное ранее в разделе *Сайты*.
4. Получив ссылку на беседу после создания маркера, вы сможете с помощью идентификатора этой беседы открыть *WebSocket* с новым свойством `StreamingConversations` для `DirectLineClient`. Для этого следует создать обратный вызов, который будет использоваться в боте для отправки клиенту `ActivitySets`:

```

public static void ReceiveActivities(ActivitySet activitySet)
{
    if (activitySet != null)
    {
        foreach (var a in activitySet.Activities)
        {
            if (a.Type == ActivityTypes.Message && a.From.Id.Contains("bot"))
            {
                Console.WriteLine($"<Bot>: {a.Text}");
            }
        }
    }
}

```

5. Теперь все готово для открытия WebSocket в `StreamingConversations` свойстве с помощью маркера диалога, `conversationId` и вашего `ReceiveActivities` обратного вызова:

```

var client = new DirectLineClient(
    new Uri(endpoint),
    new DirectLineClientCredentials(conversation.Token));

await client.StreamingConversations.ConnectAsync(
    conversation.ConversationId,
    ReceiveActivities);

```

6. Теперь этот клиент можно использовать для запуска диалога и отправки `Activities` боту:

```

var startConversation = await client.StreamingConversations.StartConversationAsync();
var from = new ChannelAccount() { Id = "123", Name = "Fred" };
var message = Console.ReadLine();

while (message != "end")
{
    try
    {
        var response = await client.StreamingConversations.PostActivityAsync(
            startConversation.ConversationId,
            new Activity()
            {
                Type = "message",
                Text = message,
                From = from
            });
    }
    catch (OperationException ex)
    {
        Console.WriteLine(
            $"OperationException when calling PostActivityAsync: ({ex.StatusCode})");
    }
    message = Console.ReadLine();
}

```

# Применение Web Chat с расширением Службы приложений Direct Line

27.03.2021 • 3 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье описывается, как использовать веб-чат с расширением прямой службы приложений. Для поддержки собственных расширений службы приложений Direct Line требуется веб-чат версии 4.9.1 или более поздней.

## Интеграция клиента Web Chat

### NOTE

Адаптивные карты, отправляемые через расширение Службы приложений Direct Line, не обрабатываются так же, как в других версиях канала Direct Line. Из-за этого в представлении JSON адаптивной карты, отправляемой в Web Chat из расширения Службы приложений Direct Line, не будут использоваться значения по умолчанию, добавленные каналом, если эти поля игнорируются ботом при создании карты.

В целом сохраняется тот же подход, что описан выше. За исключением, которое в версии 4.9.1 или более поздней версии веб-чата, существует встроенная поддержка установки двустороннего *WebSocket*, который вместо подключения к <https://directline.botframework.com/> подключается напрямую к расширению службы приложений,енному с помощью программы Bot. Прямой строковый URL-адрес для Bot будет иметь `https://<your_app_service>.azurewebsites.net/.bot/` прямую линейную *конечную точку* в расширении службы приложений. Если вы настроили собственное доменное имя, или программа-робот размещена в независимых облачах Azure, замените соответствующий URL-адрес и добавьте `/bot/` путь к интерфейсам API для расширения службы приложений Direct Line.

1. Обменяйте секрет на маркер, выполнив инструкции из [статьи об аутентификации](#). Вместо получения маркера в `https://directline.botframework.com/v3/directline/tokens/generate` вы создадите маркер непосредственно из расширения службы Direct Line Service в `https://<your_app_service>.azurewebsites.net/.bot/v3/directline/tokens/generate`.
2. Пример, демонстрирующий получение маркера, см. в статье [примеры веб-чата](#).

```

<!DOCTYPE html>
<html lang="en-US">
    <head>
        <title>Web Chat</title>
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <script>
            crossorigin="anonymous"
            src="https://cdn.botframework.com/botframework-webchat/latest/webchat-minimal.js"
        </script>
        <style>
            html,
            body {
                background-color: #f7f7f7;
                height: 100%;
            }

            body {
                margin: 0;
            }

            #webchat {
                box-shadow: 0 0 10px rgba(0, 0, 0, 0.05);
                height: 100%;
                margin: auto;
                max-width: 480px;
                min-width: 360px;
            }
        </style>
    </head>
    <body>
        <div id="webchat" role="main"></div>
        <script>
            (async function() {
                <!-- NOTE: It is highly recommended to replace the below fetch with a call to your own token
                service as described in step 2 above, and to avoid exposing your channel secret in client side code.
                -->
                const res = await
                fetch('https://<your_app_service>.azurewebsites.net/.bot/v3/directline/tokens/generate', { method:
                    'POST', headers:{'Authorization':'Bearer ' + '<Your Bot's Direct Line channel secret>'}});
                const { token } = await res.json();

                window.WebChat.renderWebChat(
                    {
                        directLine: await window.WebChat.createDirectLineAppServiceExtension({
                            domain: 'https://<your_app_service>.azurewebsites.net/.bot/v3/directline',
                            token
                        })
                    },
                    document.getElementById('webchat')
                );

                document.querySelector('#webchat > *').focus();
            })().catch(err => console.error(err));
        </script>
    </body>
</html>

```

#### TIP

В реализации JavaScript Bot убедитесь, что в файле `web.config` включены WebSockets, как показано ниже.

```
<configuration>
  <system.webServer>
    <webSocket enabled="true"/>
    ...
  </system.webServer>
</configuration>
```

# Использование расширения Службы приложений Direct Line в виртуальной сети

27.03.2021 • 2 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье описывается, как применить расширение службы приложений Direct Line в виртуальной сети Azure (VNET).

## Создание Среды службы приложений Azure и других ресурсов Azure

1. Расширение Службы приложений Direct Line доступно во всех **Службах приложений Azure**, включая размещенные в **Среде службы приложений Azure**. Среда службы приложений Azure обеспечивает изоляцию и идеально подходит для работы в виртуальной сети.
  - Инструкции по созданию внешней Среды службы приложений вы можете найти в [этой статье](#).
  - Инструкции по созданию внутренней Среды службы приложений вы можете найти в [этой статье](#).
2. После создания Среды службы приложений следует добавить в нее план Службы приложений, в котором вы сможете развернуть ботов (что приведет к запуску расширения Службы приложений Direct Line). Для этого:
  - Перейдите на сайт <https://portal.azure.com/>.
  - Создайте новый ресурс "план службы приложений".
  - В поле "Регион" выберите свою Среду службы приложений.
  - Завершите создание плана службы приложений.

## Настройте группы безопасности сети (NSG) для виртуальной сети.

1. Для расширения службы приложений Direct Line требуется исходящее подключение, чтобы оно могло создавать HTTP-запросы. Это можно настроить в качестве исходящего правила в ВИРТУАЛЬНОЙ сети NSG, связанной с подсетью Среды службы приложений. Требуется правило следующего вида:

ПОЛЕ	ЗНАЧЕНИЕ
Источник	Любой
Исходный порт	*
Назначение	Тег службы
Тег целевой службы	AzureBotService
Диапазоны портов назначения	443
Протокол	Любой
Действие	Allow

# Подключение бота к Alexa

27.03.2021 • 5 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Вы можете настроить бот для взаимодействия с людьми с помощью устройств Alexa, поддерживающих пользовательские навыки.

## IMPORTANT

Для программы-робота необходимо использовать [пакет SDK для Bot Framework](#) версии 4.8 или более поздней.

При создании новой программы-робота через портал Azure Bot будет использовать последнюю версию, доступную в это время. При наличии существующего Bot может потребоваться обновить версию пакета SDK.

## Создание навыка Alexa

- Войдите в консоль [Alexa Developer Console](#) и нажмите кнопку "Create Skill" (Создать навык).
- На следующем экране введите имя нового навыка. На этой странице можно **выбрать модель для добавления в навык** (по умолчанию выбран параметр **Custom** (Настраиваемая)), а также **выбрать метод для размещения внутренних ресурсов навыка** (по умолчанию выбран параметр **Provision your own** (Подготовка собственных ресурсов)). Оставьте выбранные параметры по умолчанию и нажмите кнопку **Create Skill** (Создать навык).

### Choose a model to add to your skill

There are many ways to start building a skill. You can design your own custom model or start with a pre-built model. Pre-built models are interaction models that contain a package of intents and utterances that you can add to your skill.

<b>Custom</b> <small>Design a unique experience for your users. A custom model enables you to create all of your skill's interactions.</small>  "Alexa, what's in the news?"	<b>Flash Briefing</b> <small>Give users control of their news feed. This pre-built model lets users control what updates they listen to.</small>  "Alexa, turn on the kitchen lights"	<b>Smart Home</b> <small>Give users control of their smart home devices. This pre-built model lets users turn off the lights and other devices without getting up.</small>  "Alexa, play music by Lady Gaga"	<b>Music</b> <small>Give users complete control of their music. This pre-built model lets users search, pause, skip, or shuffle in your skill.</small>  "Alexa, play Interstellar"	<b>Video</b> <small>Let users find and consume video content. This pre-built model supports content searches and content suggestions.</small>  "Alexa, record a dirty diaper"
<b>Baby Activity</b> <small>Let users log and retrieve events for their infants. This pre-built model supports diaper changes, feedings, sleep, and weight.</small>  "Alexa, book a room"	<b>Meetings</b> <small>This pre-built model leverages Alexa for Business APIs to allow users to search for and book available meeting rooms in their office.</small>  "Alexa, play Interstellar"			

### Choose a method to host your skill's backend resources

You can provision your own backend resources or you can have Alexa host them for you. If you decide to have Alexa host your skill, you'll get access to our code editor, which will allow you to deploy code directly to AWS Lambda from the developer console.

<b>Provision your own</b> <small>Provision your own endpoint and backend resources for a skill. With this option, you will not gain access to the console's code editor.</small>  "Alexa-Hosted (Node.js)" <small>Alexa will host skills in your account up to the AWS Free Tier limits and get you started with a Node.js template. You will gain access to an AWS Lambda endpoint, 5 GB of media storage with 15 GB of monthly data transfer, and a table for session persistence. <a href="#">Learn more</a></small>	<b>Alexa-Hosted (Python)</b> <small>Alexa will host skills in your account up to the AWS Free Tier limits and get you started with a Python template. You will gain access to an AWS Lambda endpoint, 5 GB of media storage with 15 GB of monthly data transfer, and a table for session persistence. <a href="#">Learn more</a></small>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

- На следующем экране будет предложено **выбрать шаблон**. Параметр **Start from scratch** (Начать с нуля) будет выбран по умолчанию. Оставьте параметр **Start from scratch** (Начать с нуля) выбранным и нажмите кнопку **Choose** (Выбрать).

## Choose a template

Select a quick start template to get started with a predefined skill or simply "Start from scratch"

<p><b>Start from scratch</b></p> <p>Design a unique experience for your users and define the custom model from scratch.</p> <p><b>SELECTED</b></p>	<p> <b>Fact Skill</b></p> <p>Build an engaging facts skill about any topic. Alexa will select a fact at random and share it with the user when the skill is invoked. Capabilities featured are custom intents and built-in intents.</p>	<p> <b>Quiz Game Skill</b></p> <p>Build an engaging quiz game for your users. Alexa will pick facts and quiz the users from the list provided by you. Capabilities featured are slots and custom intents.</p>	<p> <b>High-Low Game Skill</b></p> <p>Try to guess a target number and Alexa will tell you if the number she had in mind was higher or lower. Capabilities featured are slots and custom intents.</p>
----------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

4. Теперь вы увидите панель мониторинга навыка. Перейдите в **редактор JSON** в разделе **Interaction Model** (Модель взаимодействия), расположенному в меню слева.
5. Вставьте приведенный ниже код JSON в **редактор JSON**, заменив приведенные ниже значения.
  - **YOUR SKILL INVOCATION NAME** — это имя, которое пользователи будут использовать для вызова вашего навыка в Alexa. Например, если бы именем вызова навыка было "adapter helper" (вспомогательный адаптер), то пользователь мог бы сказать "Alexa, launch adapter helper" (Alexa, запусти вспомогательный адаптер), чтобы запустить навык.
  - **EXAMPLE PHRASES** — нужно предоставить три примера фраз, которые пользователи могут использовать для взаимодействия с навыком. Например, если бы пользователь сказал "Alexa, ask adapter helper to give me details of the alexa adapter" (Alexa, попроси вспомогательный адаптер предоставить мне сведения об адаптере Alexa), то примером фразы было бы "give me details of the alexa adapter" (предоставить мне сведения об адаптере Alexa).

```

{
    "interactionModel": {
        "languageModel": {
            "invocationName": "<YOUR SKILL INVOCATION NAME>",
            "intents": [
                {
                    "name": "GetUserIntent",
                    "slots": [
                        {
                            "name": "phrase",
                            "type": "phrase"
                        }
                    ],
                    "samples": [
                        "{phrase}"
                    ]
                },
                {
                    "name": "AMAZON.StopIntent",
                    "samples": []
                }
            ],
            "types": [
                {
                    "name": "phrase",
                    "values": [
                        {
                            "name": {
                                "value": "<EXAMPLE PHRASE>"
                            }
                        },
                        {
                            "name": {
                                "value": "<EXAMPLE PHRASE>"
                            }
                        },
                        {
                            "name": {
                                "value": "<EXAMPLE PHRASE>"
                            }
                        }
                    ]
                }
            ]
        }
    }
}

```

6. Нажмите кнопку **Save Model** (Сохранить модель), а затем щелкните **Build Model** (Создать модель). Конфигурация вашего навыка будет обновлена.
7. Получите свой **идентификатор навыка Алекса** либо из URL-адреса на портале Алекса, либо перейдя в [Консоль разработчика Алекса](#) и щелкнув **просмотреть идентификатор квалификации**. Идентификатор навыка Алекса должен иметь значение, например "amzn1.ask.skill.1234567890abcdef".
8. На портале Bot Framework перейдите на страницу конфигурации канала Alexa и вставьте в поле **Enter Skill Id Идентификатор навыка Alexa**.
9. На портале Alexa в меню слева перейдите в раздел **Endpoint** (Конечная точка). Выберите значение **HTTPS** для параметра **Service Endpoint Type** (Тип конечной точки службы), а в качестве конечной точки **Default Region** (Регион по умолчанию) задайте значение **Alexa Service Endpoint URI** (URI конечной точки службы Alexa), скопированное на странице конфигурации Alexa на

портале Bot Framework.

10. В раскрывающемся списке под текстовым полем, в котором определена конечная точка, необходимо выбрать тип используемого сертификата. Выберите **My development endpoint is a sub-domain of a domain that has a wildcard certificate from a certificate authority** (Моя конечная точка разработки является поддоменом домена с групповым сертификатом из центра сертификации).

Service Endpoint Type

Select how you will host your skill's service endpoint.

AWS Lambda ARN ?  
(Recommended)

HTTPS ?

Default Region ?  
(Required)

My development endpoint is a sub-domain of a domain that has a wildcard certificate fro...?

11. Нажмите кнопку **Save Endpoints** (Сохранить конечные точки) на портале Alexa.

12. Нажмите кнопку **Save** (Сохранить) на странице конфигурации канала Alexa на портале Bot Framework.

Вам потребуется опубликовать свой навык в Alexa, прежде чем другие пользователи смогут с ним взаимодействовать. Вы можете проверить свой навык, прежде чем опубликовать его, в Alexa с помощью своего устройства Alexa или на вкладке **Test** (Тестирование) для навыка. Чтобы перейти на вкладку **Test** (Тестирование), перейдите к своему навыку из консоли [Alexa Developer Console](#).

# Подключение бота к электронной почте Office 365

27.03.2021 • 4 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В дополнении к [каналам](#), бот может обмениваться данными с пользователями через электронную почту Office 365. Если у бота настроен доступ к электронной почте, то когда приходит новое письмо, он получает сообщение. Затем, согласно своей бизнес-логике, он может на него ответить. Например, бот может отправить в ответ письмо электронной почты со следующим текстом: "Hi! Благодарим вас за заказ. We will begin processing it immediately."

## WARNING

Нарушением [Соглашения по интерактивным службам](#) в Bot Framework является создание "спам-ботов", в том числе ботов, которые отправляют ненужные или нежелательные сообщения электронной почты.

## NOTE

Если вы используете Microsoft Exchange Server, перед настройкой канала электронной почты убедитесь, что [автообнаружение](#) включено.

## Настройка учетных данных электронной почты

Чтобы подключить бота к каналу электронной почты, необходимо ввести учетные данные Office 365 в настройки канала электронной почты. Федеративная проверка подлинности с использованием любого поставщика, который заменяет AAD, не поддерживается.

## NOTE

Не используйте личные учетные записи электронной почты для ботов, так как каждое сообщение, отправленное на эту учетную запись электронной почты, также будет отправлено боту. Это может привести к тому, что бот отправит неправильный ответ отправителю. Поэтому боты должны использовать только специальные учетные записи электронной почты О365.

Чтобы добавить канал электронной почты, откройте бот на [портале Azure](#), щелкните колонку **Каналы**, а затем выберите **Адрес эл. почты**. Введите действительные учетные данные электронной почты и нажмите кнопку **Сохранить**.

Enter your Email credentials  
[How do I connect my bot to Office 365 email?](#)

Email Address

address@email.com

Email Password

Password

В настоящее время канал электронной почты работает только с Office 365. Другие службы электронной почты не поддерживаются.

## Настройка электронной почты

Канал электронной почты поддерживает отправку пользовательских значений для создания более сложных настраиваемых сообщений электронной почты с помощью `channelData` Свойства Activity. В приведенном ниже фрагменте показан пример `channelData` для входящего пользовательского сообщения электронной почты от робота к пользователю.

СВОЙСТВО	ОПИСАНИЕ
htmlBody	HTML-код, используемый для текста сообщения.
subject	Тема сообщения.
importance	Флаг важности, используемый для сообщения: <code>low</code> , <code>normal</code> или <code>high</code> .
toRecipients	Строка адреса электронной почты, разделенная точками с запятой (;) для добавления к полю "Кому" сообщения.
ccRecipients	Строка адреса электронной почты, разделенная точками с запятой (;) для добавления к полю "Копия" сообщения.
bccRecipients	Строка адреса электронной почты, разделенная точками с запятой (;) для добавления к полю "Скрытая копия" сообщения.

Дополнительные сведения о `channelData` свойстве Activity см. в [разделе Создание настраиваемого сообщения электронной почты](#).

## Другие замечания

Если ваш бот не возвращает код состояния HTTP 200 OK в течение 15 секунд в ответ на входящие сообщения электронной почты, канал электронной почты попытается повторно отправить сообщение, и бот может получить это же действие сообщения электронной почты еще раз. Дополнительные сведения см. в руководствах по [использованию HTTP со ботами и устранении ошибок времени ожидания](#).

**NOTE**

При использовании учетной записи Office 365 с поддержкой многофакторной проверки подлинности (MFA) перед настройкой указанной учетной записи для канала электронной почты обязательно отключите MFA. В противном случае подключение завершится ошибкой.

## Дополнительные ресурсы

- Подключение бота к [каналам](#).
- [Реализация функциональных возможностей канала](#) с помощью пакета SDK Bot Framework для .NET.
- Дополнительные сведения о функциях, поддерживаемых в каждом канале, см. в статье [Разделенные на категории действия по каналам](#).

# Подключение бота к Facebook

27.03.2021 • 24 minutes to read • [Edit Online](#)

Вы можете подключить бота к Facebook Messenger и Facebook Workplace, чтобы он мог взаимодействовать с пользователями на обеих платформах. Приведенные ниже инструкции показывают, как подключить робот к этим двум каналам.

## NOTE

Пользовательский интерфейс Facebook может выглядеть несколько иначе, в зависимости от используемой версии.

## Подключение бота к Facebook Messenger

Дополнительные сведения о разработке приложений для Facebook Messenger см. в [документации по платформе Messenger](#). Вы можете просмотреть [инструкции перед запуском, краткое руководство и руководство по настройке](#) Facebook.

Чтобы настроить бот для обмена данными с Facebook Messenger, включите Facebook Messenger на странице Facebook и подключите бота.

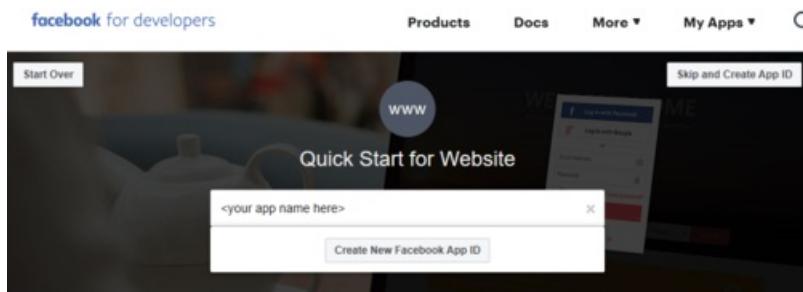
### Копирование идентификатора страницы

Доступ к боту осуществляется на странице Facebook.

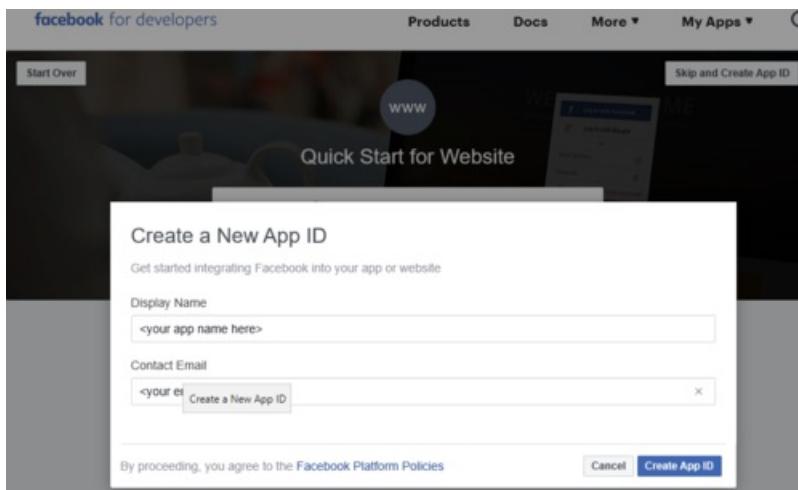
1. [Создайте страницу Facebook](#) или перейдите на имеющуюся.
2. Перейдите к странице **About** (О программе) на странице Facebook, а затем скопируйте и сохраните **идентификатор страницы**.

### Создание приложения Facebook

1. В браузере перейдите на страницу [создания приложения Facebook](#).
2. Введите имя приложения и выберите **создать новый идентификатор приложения Facebook**.



3. В появившемся диалоговом окне введите адрес электронной почты и щелкните **создать идентификатор приложения**.



4. Выполните указания мастера.
5. Введите необходимые сведения для проверки, а затем выберите **пропустить быстрое начало** в правом верхнем углу.
6. В левой области следующего отображаемого окна разверните **Параметры** и выберите **базовый**.
7. В области справа скопируйте и сохраните **идентификатор приложения** и **секрет приложения**.

8. В левой области в разделе **Параметры** выберите **Дополнительно**.
9. В области справа установите для параметра **Разрешить доступ через API к параметрам приложения** ползунок значение **Да**.

10. На странице внизу справа выберите **сохранить изменения**.

### Включить программу Messenger

1. В левой области выберите **панель мониторинга**.
2. В области справа прокрутите вниз до поля **Messenger** и выберите **настроить**. Запись Messenger отобразится в разделе **PRODUCTS** (Продукты) в области слева.

### Добавление страниц и создание маркеров

1. В левой области под записью Messenger выберите **Параметры**.
2. В области справа прокрутите вниз до пункта **маркеры доступа** и выберите **Добавить или**

## **УДАЛИТЬ СТРАНИЦЫ.**

Access Tokens

Create New Page

Generate a Page access token to start using the platform APIs. You will be able to generate an access token for a Page if:

1. You are one of the Page admins, and
2. The app has been granted the Page's permission to manage and access Page conversations in Messenger.

Note: If your app is in dev mode, you can still generate a token but will only be able to access people who manage the app or Page.

### **No page permissions granted**

You'll need to connect pages and grant them the required permissions in order for tokens to be generated.

[Add or Remove Pages](#)  ⓘ

3. В списке, который находится в следующем окне, выберите страницы, которые нужно использовать с приложением.

4. Нажмите кнопку **Готово**.

5. Чтобы создать токен для этой страницы, выберите **создать токен**.

### **Включение веб-перехватчиков**

Чтобы отправлять сообщения и другие события из бота в Facebook Messenger, необходимо включить интеграцию веб-перехватчиков. Оставьте ожидающие установки Facebook шаги. Вы обновите их позже.

1. В браузере откройте новое окно и перейдите на [портал Azure](#).

2. В списке ресурсов выберите регистрацию ресурса Bot и в соответствующем колонке выберите **каналы**.

3. На правой панели выберите значок **Facebook**.

4. В мастере введите данные Facebook, сохраненные на предыдущих шагах. Если информация правильная, в мастере внизу вы увидите *URL-адрес обратного вызова и токен проверки*. Скопируйте и сохраните эти значения.

## Configure Facebook Channel



Enter Facebook Messenger or Workplace by Facebook Credentials  
Step-by-step instructions to add the bot to Facebook Messenger.  
Step-by-step instructions to add the bot to Workplace by Facebook.

<p>Facebook App ID *</p> <p>Facebook app ID can be found in your Facebook account settings</p>	<p>fb stored value</p> 	
<p>Facebook App Secret *</p> <p>Facebook app secret can be found in your Facebook account settings</p>	<p>fb stored value</p> 	
<p>Page ID *</p> <p>Page ID can be found in your Facebook account</p>	<p>Page Access Token *</p> <p>Access Token can be found in your Facebook account</p>	<p>fb stored value</p> 
<p><a href="#">Add a new page</a></p>		

Callback URL and Verify Token for Facebook  
[What do I do with my Callback URL and Verify Token?](#)

<p>Callback URL: (copy and paste in Facebook)</p> <p>https://facebook.botframework.com/api/v1/bots/DeployEchoBot</p>	<p>copy</p> 
<p>Verify Token: (copy and paste in Facebook)</p> <p>qjew2f29553Re95kD00C0ICDGtEELCK54r2ja8sf7b94hc</p>	<p>copy</p> 

Want to add your bot to Facebook Messenger App Directory?  
[Learn how](#)

5. Щелкните **Сохранить**.

6. Вернитесь к параметрам Facebook, чтобы завершить процесс настройки.

7. Введите URL-адрес обратного вызова и проверьте значения маркера, полученные из портал Azure.

8. В разделе Настройка веб-перехватчиков включите следующие подписки: **\_ Доставка сообщений, сообщения, параметры обмена сообщениями и обратная передача сообщений.**

### Отправка на проверку

На странице базовых параметров приложения Facebook требуется указать URL-адрес политики конфиденциальности и условий использования. На странице [Code of Conduct](#) (Правила поведения) содержатся ссылки на сторонние ресурсы для создания политики конфиденциальности. На странице [Terms of Use](#) (Условия использования) содержится пример условий, помогающий создать соответствующий документ с условиями использования.

После завершения работы с ботом в Facebook будет реализован собственный механизм [рассмотрения](#) для приложений, которые публикуются в Messenger. Программа-робот будет протестирована на соответствие [политикам платформы](#) Facebook.

### Предоставления общего доступа к приложению и публикация страницы

#### NOTE

До публикации приложение находится в [режиме разработки](#). Функции подключаемого модуля и API будут работать только для администраторов, разработчиков и тест-инженеров.

После успешной проверки на панели мониторинга приложения в разделе рассмотрения приложения выберите значение Public (общедоступное). Убедитесь, что страница Facebook, связанная с этим ботом, опубликована. Состояние появляется в параметрах страницы.

# Подключение бота к Facebook Workplace

## NOTE

16 декабря 2019 г. в Workplace by Facebook изменилась модель обеспечения безопасности для пользовательских интеграций. Интеграции, созданные ранее с помощью Microsoft Bot Framework версии 4, необходимо обновить до 28 февраля 2020 г. для использования адаптеров Facebook для Bot Framework в соответствии с приведенными ниже инструкциями.

Далее Facebook будет считать допустимыми для продолжения использования (до 31 декабря 2020 г.) только интеграции с ограниченным доступом к данным Workplace (разрешения с низким уровнем конфиденциальности), которые успешно прошли проверки безопасности RFI и по которым разработчик не позднее 15 января 2020 г. связался со службой поддержки [Direct Support](#) с запросом на продолжение работы приложения.

Адаптеры Bot Framework доступны для ботов, созданных с помощью [JavaScript](#) и [Node.js](#), а также [C#](#) и [.NET](#).

Facebook Workplace — это ориентированная на бизнес версия Facebook, которая позволяет сотрудникам легко подключаться и совместно работать. Она содержит видео в реальном времени, веб-каналы новостей, группы, приложение для обмена сообщениями, реакции, функции поиска и популярные записи. Она также поддерживает следующие возможности:

- Аналитика и интеграция. Панель мониторинга с функциями аналитики, интеграции и единого входа, а также поставщиками удостоверений, которые используются компаниями для интеграции Workplace с существующими ИТ-системами.
- Группы из нескольких компаний. Общие пространства, в которых сотрудники из разных организаций могут совместно работать.

Откройте [центр справки Workplace](#), чтобы изучить сведения о Facebook Workplace и [документацию для разработчиков Workplace](#), с подробными рекомендациями.

Чтобы использовать Facebook Workplace с ботом, создайте учетную запись Workplace и настраиваемую интеграцию для подключения бота.

## Создание учетной записи Workplace Premium

1. Отправьте приложение в [Workplace](#) от имени компании.
2. Когда приложение будет утверждено, вы получите сообщение электронной почты с приглашением присоединиться. Это может занять некоторое время.
3. В приглашении по электронной почте выберите **Приступая к работе**.
4. Укажите данные профиля.

## TIP

Назначьте себя системным администратором. Помните, что только системные администраторы могут создавать настраиваемые интеграции.

5. Выберите **Предварительный просмотр профиля** и проверьте правильность сведений.
6. Получите доступ к **бесплатной пробной версии**.
7. Создайте **пароль**.
8. Выберите **пригласить** сотрудников, чтобы пригласить служащих для входа. Приглашенные сотрудники станут участниками, как только они войдут, следуя приведенным здесь инструкциям.

## Создание пользовательской интеграции

Создайте [настраиваемую интеграцию](#) для Workplace, как описано ниже. При создании пользовательской интеграции создается приложение с определенными разрешениями и страницей типа [Bot](#)

(отображается только в сообществе вашего рабочего места).

1. На панели администратора откройте вкладку **Интеграции**.
2. Выберите **создать собственное пользовательское приложение**.

The screenshot shows the Microsoft Workplace Admin Panel. On the left, there's a sidebar with various management options like Setup, Reporting, People, Groups, and Administrators. The 'Integrations' option is selected and highlighted with a red box. The main area is titled 'Integrations' with the sub-instruction 'Build integrations that extend the functionality of Workplace using APIs.' Below this, there are sections for 'Added to Workplace' (with a note to select an integration for more info) and 'Added By Your Company' (with a note that no integrations have been installed yet). There's also a 'Custom Integrations' section where a 'Cafe Bot' entry is listed, showing it's enabled. A blue button at the bottom right of this section says 'Create Custom Integration'.

3. Выберите для приложения отображаемое имя и изображение профиля. Такие сведения будут предоставлены страницей типа **Bot**.
4. Установите для параметра **Allow API Access to App Settings** (Разрешить доступ API к параметрам приложения) значение Yes (Да).
5. Скопируйте и безопасно сохраните код приложения, секрет приложения и маркер приложения, которые вы здесь видите.

This screenshot shows the 'Edit Integration' dialog box. It has fields for 'Name' (set to 'Cafe Bot'), 'Update Logo' (with a placeholder image), 'Description' (set to 'Cafe Bot'), and 'App Enabled' (set to 'Yes'). To the right, there are boxes for 'App ID' (containing '37848932759751285'), 'API Version' (set to 'v3.1'), and 'App Secret' (showing several asterisks). Below these are buttons for 'Show' and 'Reset Access Token'. Underneath the dialog, there's a 'Grant Permissions' section with several checkboxes for managing accounts, impersonating accounts, managing groups, and reading messages and group content. A blue button 'Create Custom Integration' is visible on the right.

6. Итак, вы завершили создание пользовательской интеграции. Вы можете найти страницу типа **Bot** в сообществе вашего рабочего места, как показано ниже.

This screenshot shows a web browser window with the URL 'https://workplace.facebook.com/SomeTestBot-8402821602859423/'. The page title is 'TestBotV3 - Home'. The main content area features a large image of a dark blue cube icon and the text 'SomeTestBot'. Below this are links for 'Home', 'Photos', and 'About'. To the right, there's a 'Send Message' button and a 'BOTs' section showing another instance of 'SomeTestBot'. The bottom of the page includes standard footer links for English (US), Español, Português (Brasil), Français (France), Deutsch, Premium Privacy Policy, Acceptable Use Policy, Report Issue, Cookies, and More.

## Включение адаптера Facebook в код бота

Исходный код бота следует обновить, включив в него адаптер для взаимодействия с Workplace by Facebook. Эти адаптеры доступны для ботов, созданных с помощью [JavaScript и Node.js](#), а также [C# и .NET](#).

## Указание учетных данных Facebook

В appsettings.js файл Bot добавьте **идентификатор приложения Facebook, секрет приложения Facebook и маркер доступа к странице**, скопированные из рабочей области Facebook ранее. Вместо традиционного идентификатора страницы используйте номера, следующие за именем интеграции на странице About. Выполните эти инструкции, чтобы обновить исходный код бота для [JavaScript и Node.js](#) или [C# и .NET](#).

## Отправка приложения рабочей области для проверки

Подробные сведения см. в разделе **о подключении бота к Facebook Messenger** и в [документации для разработчика Workplace](#).

## Сделайте приложение рабочей области общедоступным и опубликуйте страницу

Подробные сведения см. в разделе **о подключении бота к Facebook Messenger**.

## Указание версии API

Если вы получите уведомление от Facebook о том, что определенная версия API Graph теперь считается нерекомендуемой, перейдите на [страницу для разработчиков Facebook](#). Перейдите к **параметрам приложения** Bot и перейдите в раздел **Параметры > Расширенное > обновление API версия**, а затем переключите **обновление всех вызовов** до версии 4,0.

## Подключение бота к Facebook с помощью адаптера Facebook

Чтобы подключить бота к Facebook Workplace, используйте адаптер Facebook для Bot Framework. Для подключения к Facebook Messenger можно использовать канал Facebook или адаптер Facebook. Адаптеры Facebook доступны для ботов, созданных с помощью [JavaScript и Node.js](#), а также [C# и .NET](#).

Из этой статьи вы узнаете, как подключить бота к Facebook с помощью адаптера. В этой статье описано, как изменить образец Echo Bot, чтобы подключить его к Facebook.

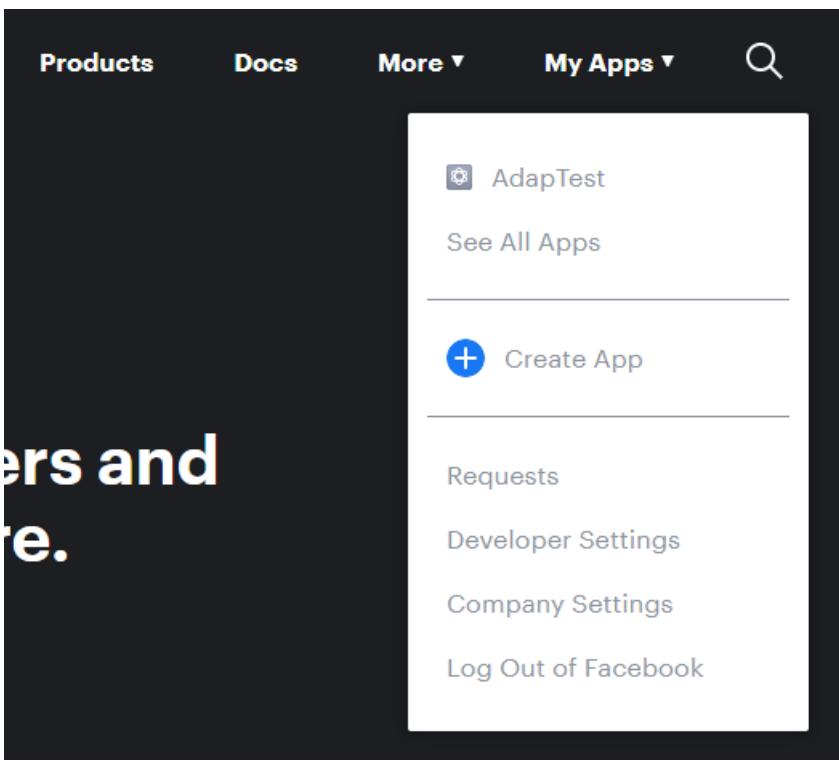
Ниже приводятся инструкции для реализации адаптера Facebook на C#. Инструкции по использованию адаптера для JavaScript, который входит в состав библиотек BotKit, см. в [документации по BotKit для Facebook](#).

### Предварительные требования

- [Пример кода EchoBot](#).
- Учетная запись Facebook для разработчиков. Если у вас нет учетной записи, [создайте ее](#).

### Создание приложения и страницы Facebook, сбор учетных данных

1. Войдите на сайт <https://developers.facebook.com>. В главном меню выберите **Мои приложения > создать приложение**.



2. В появившемся диалоговом окне введите отображаемое имя нового приложения, а затем выберите **создать идентификатор приложения**.

## Create a New App ID

Get started integrating Facebook into your app or website

Display Name

My Bot App|

### Настройка Messenger и связывание страницы Facebook

1. После создания приложения отобразится список доступных для настройки продуктов. Щелкните **настроить** рядом с продуктом Messenger .
2. Теперь необходимо связать новое приложение со страницей Facebook, — чтобы создать страницу, если у вас нет нужной страницы, выберите в разделе **маркеры доступа** пункт **создать новую страницу** . Выберите **Добавить или удалить страницы**, выберите страницу, которую необходимо связать с приложением, и нажмите кнопку **Далее**. Оставьте **диалоговое окно Управление и доступ к страницам в параметрах программы Messenger** включенным и нажмите кнопку **Готово**.

### Submit for Login Review

Some of the permissions below have not been approved for use by Facebook.

[Submit for review now](#) or [learn more](#).

## What is My Bot App allowed to do?

 My Bot App might not work properly if you turn off these options.

Manage and access Page conversations on Messenger  
My Bot App

YES

[Cancel](#)

[Back](#)

[Done](#)

1. После связывания страницы выберите **создать маркер**, чтобы создать маркер доступа к странице. Запишите этот маркер, так как он понадобится позже при настройке приложения бота.

### Получение секрета приложения

1. В меню слева выберите **Параметры**, а затем — **базовый**, чтобы вернуться на страницу основные параметры приложения.
2. На странице Основные параметры установите флагок **Показывать рядом с секретом приложения**. Запишите этот секрет, так как он потребуется позже при настройке приложения бота.

### Подключение адаптера Facebook к боту

Теперь, когда у вас есть приложение, страница и учетные данные Facebook, можно настроить приложение бота.

#### Установка пакета NuGet для адаптера Facebook

Добавьте пакет NuGet `Microsoft.Bot.Builder.Adapters.Facebook`. Подробные сведения об использовании NuGet см. в руководстве по [установке пакетов и управлении ими в Visual Studio](#).

#### Создание класса адаптера Facebook

Создайте новый класс, наследующий от `FacebookAdapter` класса. Этот класс выступает в качестве адаптера для канала Facebook и включает возможности обработки ошибок (аналогично `BotFrameworkAdapterWithErrorHandler` классу, уже указанному в примере, который используется для обработки других запросов от службы Azure Bot).

```

public class FacebookAdapterWithErrorHandler : FacebookAdapter
{
    public FacebookAdapterWithErrorHandler(IConfiguration configuration, ILogger<BotFrameworkHttpAdapter>
logger)
        : base(configuration, logger)
    {
        OnTurnError = async (turnContext, exception) =>
    {
        // Log any leaked exception from the application.
        logger.LogError(exception, $"[OnTurnError] unhandled error : {exception.Message}");

        // Send a message to the user
        await turnContext.SendActivityAsync("The bot encountered an error or bug.");
        await turnContext.SendActivityAsync("To continue to run this bot, please fix the bot source
code.");

        // Send a trace activity, which will be displayed in the Bot Framework Emulator
        await turnContext.TraceActivityAsync("OnTurnError Trace", exception.Message,
"https://www.botframework.com/schemas/error", "TurnError");
    };
    }
}

```

#### **Создание контроллера для обработки запросов Facebook**

Создайте новый контроллер, который будет выполнять запросы из Facebook на новой `api/facebook` конечной точке вместо конечной точки по умолчанию, `api/messages` используемой для запросов от каналов обслуживания Azure Bot. Добавив к боту дополнительную конечную точку, вы сможете с помощью одного бота принимать запросы одновременно от каналов службы Bot и Facebook.

```

[Route("api/facebook")]
[ApiController]
public class FacebookController : ControllerBase
{
    private readonly FacebookAdapter _adapter;
    private readonly IBot _bot;

    public FacebookController(FacebookAdapter adapter, IBot bot)
    {
        _adapter = adapter;
        _bot = bot;
    }

    [HttpPost]
    [HttpGet]
    public async Task PostAsync()
    {
        // Delegate the processing of the HTTP POST to the adapter.
        // The adapter will invoke the bot.
        await _adapter.ProcessAsync(Request, Response, _bot);
    }
}

```

#### **Включение адаптера Facebook в файл startup.cs бота**

Добавьте следующую строку в `ConfigureServices` метод в файле `Startup.cs`. Это действие регистрирует адаптер Facebook и делает его доступным для нового класса контроллера. Добавленные на предыдущем шаге параметры конфигурации применяются адаптером автоматически.

```
services.AddSingleton<FacebookAdapter, FacebookAdapterWithErrorHandler>();
```

После добавления `ConfigureServices` метод должен выглядеть следующим образом.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

    // Create the default Bot Framework adapter (used for Azure Bot Service channels and Emulator).
    services.AddSingleton<IBotFrameworkHttpAdapter, BotFrameworkAdapterWithErrorHandler>();

    // Create the Facebook adapter
    services.AddSingleton<FacebookAdapter, FacebookAdapterWithErrorHandler>();

    // Create the bot as a transient. In this case the ASP controller is expecting an IBot.
    services.AddTransient<IBot, EchoBot>();
}
```

#### Получение URL-адреса для бота

Теперь, когда вы подключили адаптер в проекте бота, необходимо передать в Facebook правильную конечную точку этого приложения, чтобы бот мог получать сообщения. Этот URL-адрес также потребуется для завершения настройки приложения бота.

Чтобы выполнить этот шаг, [разверните бота в Azure](#) и запишите URL-адрес этого развертывания.

#### NOTE

Если вы не готовы к развертыванию программы Bot в Azure или хотите отладить программу Bot при использовании адаптера Facebook, можно использовать такое средство, как [ngrok](#) (которое уже установлено, если ранее использовался эмулятор Bot Framework) для туннелирования на интерфейсе Bot, запущенном локально, и предоставить вам общедоступный URL-адрес для этого.

Если вы хотите создать туннель и получить для бота URL-адрес с помощью ngrok, выполните следующую команду в окне терминала. Здесь предполагается, что локальный бот работает на порту 3978. Если это не так, измените номера портов в команде.

```
ngrok.exe http 3978 -host-header="localhost:3978"
```

#### Добавление параметров приложения Facebook в файл конфигурации бота

Добавьте указанные ниже параметры в `appsettings.js` в файле в проекте Bot. Параметры `FacebookAppSecret` и `FacebookAccessToken` заполняются значениями, которые вы записали при создании и настройке приложения Facebook. **Фацебукверифитокен** должен быть случайной строкой, которую вы создаете и будете использовать для проверки подлинности конечной точки Bot при вызове Facebook.

```
"FacebookVerifyToken": "",  
"FacebookAppSecret": "",  
"FacebookAccessToken": ""
```

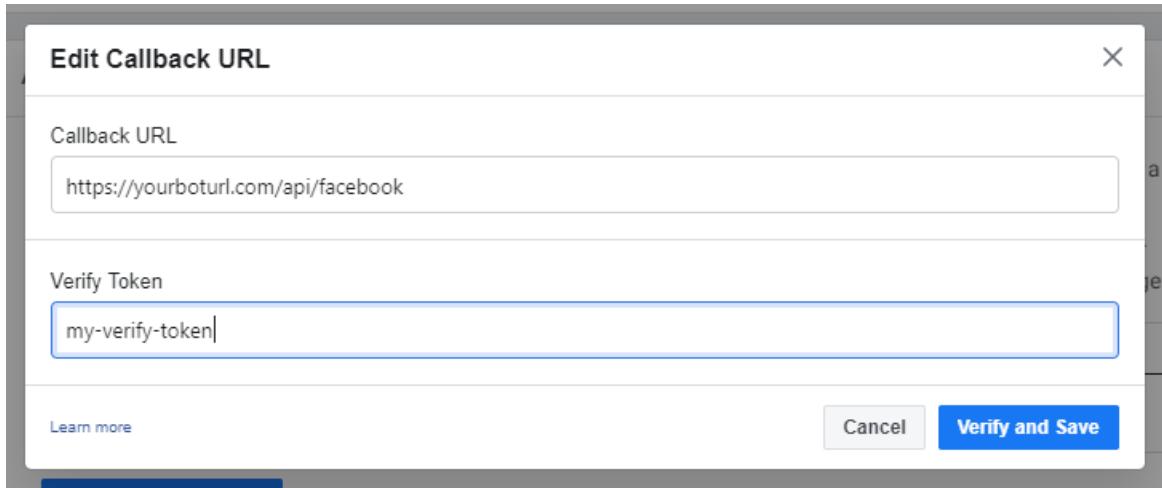
Теперь, когда вы заполнили указанные выше параметры, следует повторно развернуть бота или просто перезапустить его, если вы используете туннель ngrok к локальной среде.

#### Завершение настройки приложения Facebook

Заключительным этапом является настройка конечной точки Messenger приложения Facebook, чтобы бот мог получать из нее сообщения.

- На панели мониторинга приложения выберите **Messenger** в меню слева и щелкните **Параметры**.
- В разделе **веб-перехватчики** выберите **Добавить URL обратного вызова**.
- В текстовом поле **URL обратного вызова** введите URL-адрес бота и конечную точку

`api/facebook`, которую вы указали в только что созданном контроллере. Например, `https://yourboturl.com/api/facebook`. В текстовом поле **Проверка токена** введите созданный ранее токен проверки и используемый в `appsettings.js` приложения-робота в файле.



4. Выберите **проверить и сохранить**. Убедитесь, что бот запущен, так как Facebook выполнит запрос к конечной точке этого приложения и проверит его с помощью значения, указанного в поле **Маркер подтверждения**.
5. После проверки URL-адреса обратного вызова выберите **Добавить подписки**, которые теперь отображаются. Во всплывающем окне выберите следующие подписки и нажмите кнопку **сохранить**.

- messages
- messaging\_postbacks
- messaging\_optins
- messaging\_deliveries

#### Subscription Fields

<input checked="" type="checkbox"/> messages	<input checked="" type="checkbox"/> messaging_postbacks	<input checked="" type="checkbox"/> messaging_optins
<input checked="" type="checkbox"/> message_deliveries	<input type="checkbox"/> message_reads	<input type="checkbox"/> messaging_payments
<input type="checkbox"/> messaging_pre_checkouts	<input type="checkbox"/> messaging_checkout_updates	<input type="checkbox"/> messaging_account_linking
<input type="checkbox"/> messaging_referrals	<input type="checkbox"/> message_echoes	<input type="checkbox"/> messaging_game_plays
<input type="checkbox"/> standby	<input type="checkbox"/> messaging_handovers	<input type="checkbox"/> messaging_policy_enforcement
<input type="checkbox"/> message_reactions		

[Learn more](#)

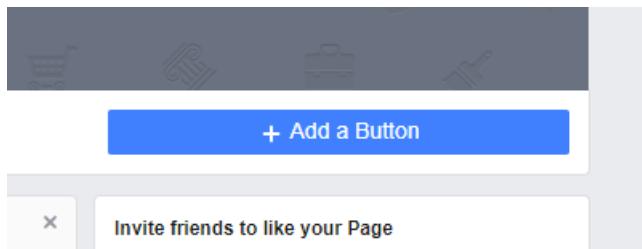
[Cancel](#)

[Save](#)

#### Тестирование работы бота с адаптером в Facebook

Теперь можно проверить, правильно ли бот подключен к Facebook, отправив сообщение через страницу Facebook, которую вы связали с новым приложением Facebook.

1. Перейдите к нужной странице Facebook.
2. Нажмите **кнопку Добавить**.



3. Выберите **контакт** и **отправьте сообщение**, а затем нажмите кнопку **Далее**.

Add a button to your Page

**Preview**

Like Follow Share ... Send Message

**Step 1:** Which button do you want people to see?

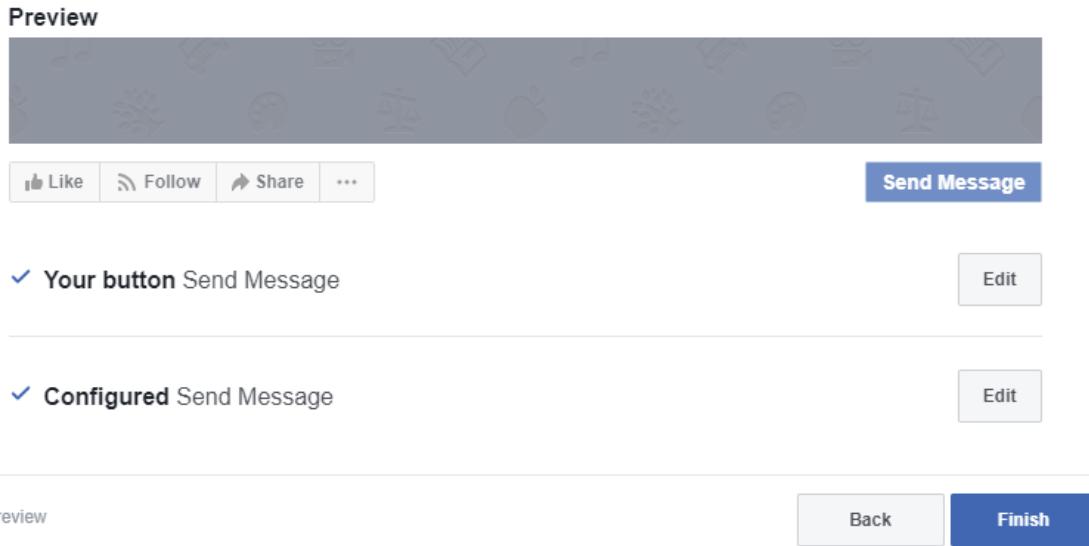
The button at the top of your Page helps people take an action. People see it on your Page and in search results when your Page appears. You can edit it at any time.

<input type="checkbox"/> Make a booking with you	▼
<input checked="" type="checkbox"/> Contact you	^
<input type="radio"/> Contact Us	<input type="radio"/> Sign Up
<input checked="" type="radio"/> Send Message	<input type="radio"/> Send Email
<input type="radio"/> Call Now	
<input type="checkbox"/> Learn more about your business	▼
<input type="checkbox"/> Shop with you	▼
<input type="checkbox"/> Download your app or play your game	▼

Step 1 of 2

Cancel Next

4. Когда появится вопрос, **куда вы хотите нажать эту кнопку для отправки людей?** выберите Messenger, а затем нажмите кнопку **Готово**.



5. Наведите указатель мыши на кнопку **Новая Отправка сообщения**, которая теперь отображается на странице Facebook, и выберите пункт **проверить** во всплывающем меню. Это действие запускает новый диалог с приложением через Facebook Messenger, в котором вы можете проверить обмен сообщениями с ботом. Каждый раз, когда ваш бот получает сообщение, он отправляет вам ответ с копией текста вашего сообщения.

Вы также можете протестировать эту функцию, используя пример программы- [робота для адаптера Facebook](#), заполнив `appsettings.json` файле с теми же значениями, которые описаны выше.

## См. также раздел

- **Пример кода.** См. пример бота [Facebook-events](#), чтобы узнать об обмене данными между ботом и Facebook Messenger.

# Подключение бота к веб-чату

27.10.2020 • 2 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Вы можете настроить взаимодействие бота с другими пользователями с помощью программы для обмена сообщениями GroupMe.

## TIP

Описание функций, поддерживаемых в каждом канале, см. в статье [Разделенные на категории действия по каналам](#).

## Регистрация учетной записи GroupMe

Если у вас нет учетной записи GroupMe, [зарегистрируйтесь](#), чтобы создать учетную запись.

## Создание приложения GroupMe

[Создайте приложение GroupMe](#) для вашего бота.

Используйте следующий URL-адрес обратного вызова: <https://groupme.botframework.com/Home/Login>

## Create Application

Application Name

Callback URL

Callback URL must be https, localhost, or a deep link.

Developer Name

Developer Email

Developer Phone Number

Developer Company

Developer Address

I agree to abide by the [Terms of Use](#) and the [Brand Standards](#)

[Save](#)

[Cancel](#)

## Получение учетных данных

1. В поле **URL-адрес перенаправления** скопируйте значение параметра `client_id` = .
2. Скопируйте значение параметра **Маркер доступа**.

## {YOUR BOT NAME}

Details    Settings    Delete

Settings

Redirect URL    [https://oauth.groupme.com/oauth/authorize?client\\_id=Your Client Id](https://oauth.groupme.com/oauth/authorize?client_id=Your Client Id)

Callback URL    https://groupme.botframework.com/Home/Login

Your Access Token

Use the access token string to authenticate as yourself when making API requests.

Access Token    Your Access Token

## Отправка учетных данных

1. На странице dev.botframework.com вставьте скопированное значение параметра `client_id` в поле **Идентификатор клиента**.
2. Вставить значение **Маркера доступа** в поле **Маркер доступа**.
3. Выберите команду **Сохранить**.

Access Token

Access token can be found in your GroupMe account settings

Client ID

Client ID can be found in your GroupMe account settings

# Подключение бота к Kik

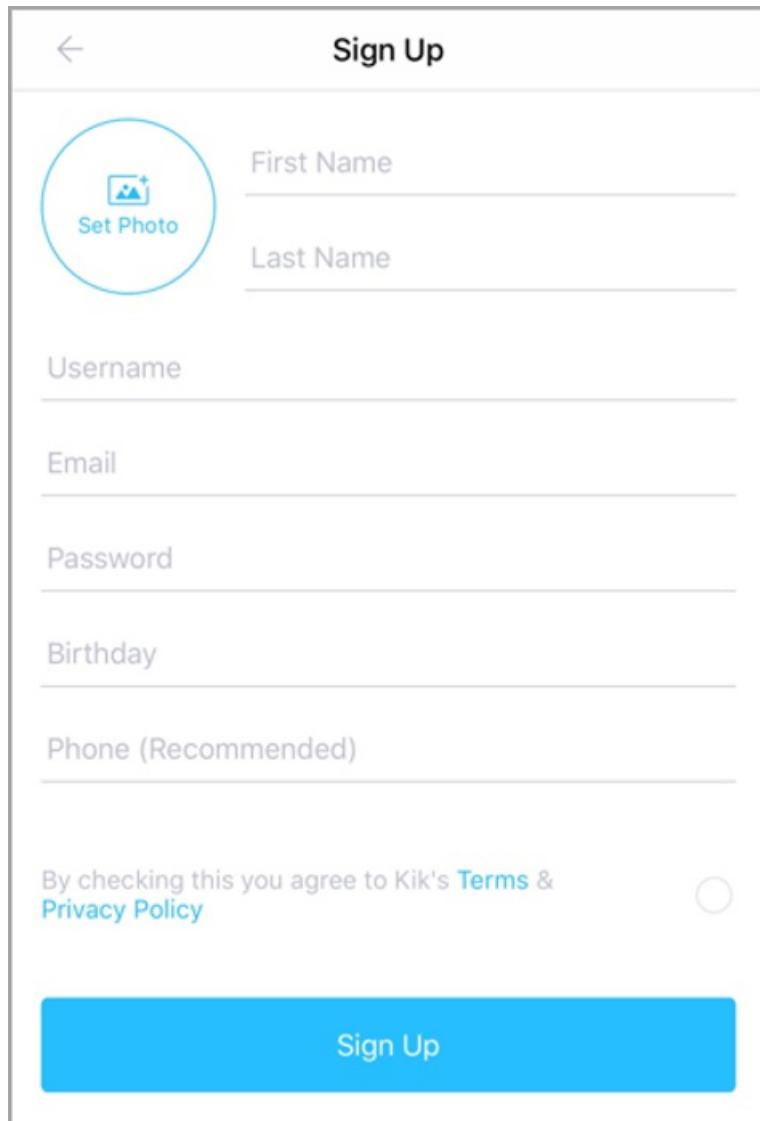
27.10.2020 • 2 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Можно настроить бот для общения с людьми, использующими приложение обмена сообщениями Kik.

## Установите Kik на телефон

Если Kik не установлен на телефоне, это можно сделать с помощью магазина приложений для вашей платформы или на [веб-сайте Kik](#). Вам необходимо использовать существующую учетную запись Kik или зарегистрировать новую.



The screenshot shows the 'Sign Up' page of the Kik application. At the top, there is a back arrow icon and the text 'Sign Up'. On the left side, there is a circular profile picture placeholder with a 'Set Photo' button. Below it are fields for 'First Name' and 'Last Name', both with placeholder text. Further down are fields for 'Username', 'Email', 'Password', and 'Birthday', each with placeholder text. A field for 'Phone (Recommended)' follows. At the bottom of the form, there is a statement: 'By checking this you agree to Kik's [Terms](#) & [Privacy Policy](#)', with an unchecked checkbox next to it. A large blue 'Sign Up' button is at the bottom of the form.

## Войдите на портал разработчика с мобильного телефона

Используйте мобильный телефон для [входа на портал Kik](#). Когда отобразится запрос *Open this page in "Kik"?* (Открыть эту страницу в Kik?), щелкните **Open** (Открыть).

# Build Your Bot

First there were websites, then there were  
apps.

Now, there are bots.



Scan To Create Your Bot

1. Open Kik    2. [Scan Code](#)

## Установка бота

Присвойте имя своему боту.



## Получение учетных данных

На вкладке "Конфигурация" скопируйте **Имя** и **Ключ API**.

A screenshot of the "Configuration" tab in the Kik Dev application. The top navigation bar includes links for "Reporting", "Configuration", "Bot Shop Settings", "Bot Shop", "Docs", and a user profile icon. The main content area is titled "Account Settings". It features a circular "Edit" button next to a placeholder "yourbotname" for the display name. Below it is a "Display Name" input field containing "yourbotname" with "Cancel" and "Save" buttons. There's also a "Admins" section with an "Add an admin..." input field and an "Add" button. A note "larsliden" is listed below. The "API Key" section shows a generated API key (redacted), its generation date ("Generated: 14:20 Apr 13, 2016"), and a "Regenerate" button.

## Отправка учетных данных

Bot Name

API Key

Submit Kik Credentials

Нажмите **Отправить учетные данные Kik**.

## Включение бота

Проверка функции **Включить этот бот на Kik**. Затем нажмите **Настройка Kik завершена**.

Сделав это, бот будет успешно настроен для общения с пользователями в приложении Kik.

# Подключение бота к LINE

27.03.2021 • 6 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Вы можете настроить бот для общения с людьми, использующими приложение LINE.

## Вход в консоль LINE

Войдите в [консоль разработчика LINE](#) с учетной записью LINE, используя действие *Log in with Line* (Вход в LINE).

### NOTE

Если у вас нет приложения LINE, [скачайте его](#) и перейдите к параметрам, чтобы зарегистрировать адрес электронной почты.

### Регистрация в качестве разработчика

Если вы впервые имеете дело с консолью разработчика LINE, введите здесь имя и адрес электронной почты, чтобы создать учетную запись разработчика.

The screenshot shows the LINE Developers Console interface. On the left, there's a sidebar with 'Welcome Username' and a gear icon. Below it, 'Providers' is selected in green, followed by 'Has not provider' and 'Tools'. The main area has a title 'Provider List' and a 'Create New Provider' button. A central message says 'Welcome to LINE Developers Console!' with a sub-instruction: 'Let's develop an app that connects people with people using your development technology and LINE Platform! A provider is a service provider (company / individual), and we begin by creating a provider.' Below this, three steps are shown: 'STEP 1' (a folder icon with two green stars), 'STEP 2' (a folder icon with a green arrow and a speech bubble icon), and 'STEP 3' (a smartphone icon with green code brackets). Under each step is a descriptive text and a 'Create New Provider' button at the bottom.

## Создание поставщика

Для начала создайте поставщик для бота, если у вас он еще не настроен. Поставщиком называется физическое или юридическое лицо, которое предоставляет некоторое приложение.

Welcome Username

Providers

Has not provider

Tools

## Create new provider

Enter channel information

Confirm

Done

Provider name

Contoso

Max: 100 characters

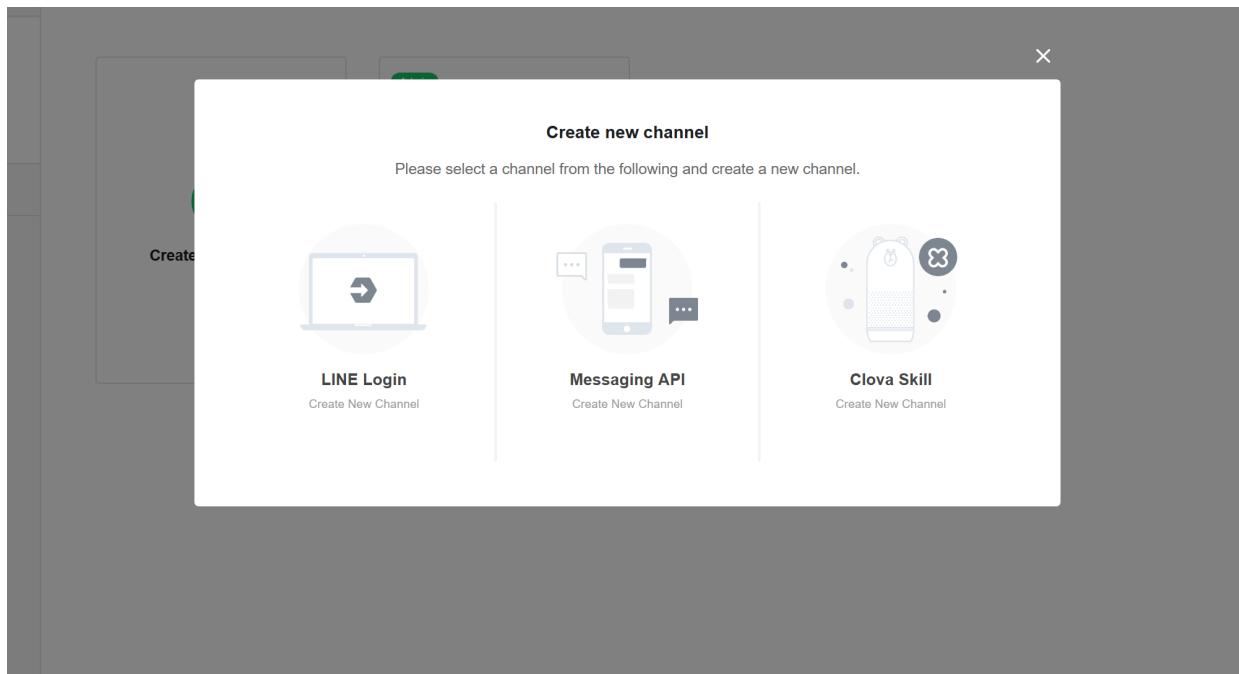
Confirm

© LINE Corporation   Terms and policies   About trademarks   LINE    LINE Corp    LINE Partner    LINE@ 

English 

## Создание канала API обмена сообщениями

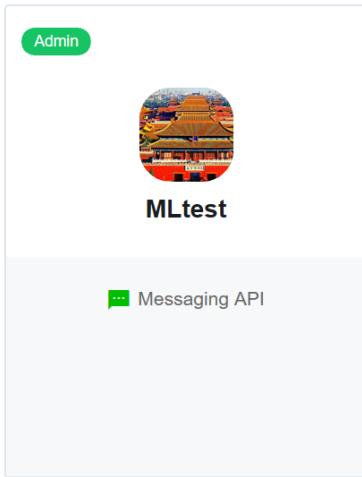
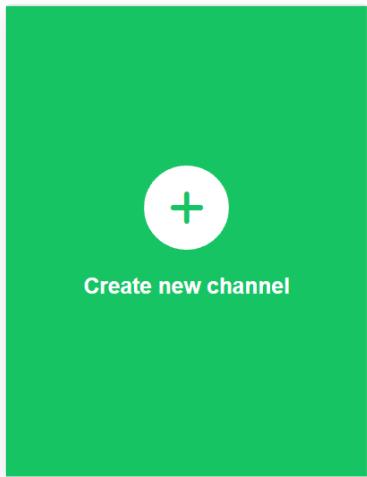
Теперь создайте канал API обмена сообщениями



Создайте канал API обмена сообщениями, щелкнув зеленый квадратик.



ML



Имя канала не может содержать строку LINE и некоторые похожие буквосочетания. Заполните обязательные поля и подтвердите параметры канала.

	<p><b>App name</b></p> <input type="text" value="Enter app name"/> <p>Max: 20 characters</p> <p>Note: The app name cannot be changed for seven days.</p> <hr/> <p><b>App description</b></p> <input type="text" value="Enter app description"/> <p>Max: 500 characters</p> <hr/> <p><b>Plan</b></p> <p><input checked="" type="radio"/> <b>Developer Trial</b> A trial plan which lets you create a bot that can send push messages and have up to 50 friends. Note: You cannot upgrade or buy a premium ID for a Developer Trial plan.</p> <p><input type="radio"/> <b>Free</b> A plan which lets you create a bot with an unlimited number of friends. Push messages cannot be sent with this plan. Note: You can upgrade this plan at any time.</p> <hr/> <table border="1" style="width: 100%;"> <thead> <tr> <th style="width: 50%;">Category</th> <th style="width: 50%;">Subcategory</th> </tr> </thead> <tbody> <tr> <td>Select category</td> <td>Select subcategory</td> </tr> </tbody> </table> <hr/> <p><b>Email address</b> </p> <input type="text" value="example@line.me"/> <p>Max: 100 characters</p>	Category	Subcategory	Select category	Select subcategory
Category	Subcategory				
Select category	Select subcategory				

## Получение необходимых значений из настроек канала

Когда вы подтвердите настройки канала, откроется страница со следующим содержимым.

Welcome  
Username


Providers

Provider List

Contoso

Tools

Contoso

 Create new channel

  
Contoso

New Admin

Messaging API

Щелкните здесь канал, который вы создали для доступа к параметрам канала, и прокрутите экран вниз до раздела **Basic information (Базовая информация)** > **Channel secret (Секрет канала)**. Сохраните эти сведения, они вам скоро пригодятся. Убедитесь, что в списке доступных функций присутствует **PUSH\_MESSAGE**.

The screenshot shows the LINE Developers portal interface. At the top, there's a navigation bar with links for Products, Documents, News, FAQ, Community, and Blog. On the right side of the header are a search icon and a user profile icon.

The main content area displays the following channel settings:

- Channel ID**: 1968854556
- Channel secret**: 7e53a5215b0455fe6ca72e2926b5091d (with a blue "Issue" button to its right)
- App type**: BOT
- Plan**: For Developer (with a "Change plan" link to its right). A note below says: "To verify your plan after making a change, reload the page."
- Available features**: REPLY\_MESSAGE, PUSH\_MESSAGE
- Email address**: bots-line-email@contoso.com (with an "Edit" button to its right)

Теперь прокрутите еще дальше до пункта **Messaging settings** (Параметры обмена сообщениями). Здесь вы увидите поле **Channel access token** (Маркер доступа к каналу) с кнопкой *issue* (Выдать). Нажмите эту кнопку, чтобы получить маркер доступа, и также сохраните его.

### Messaging settings

Channel access token (long-lived) [?](#)  
Pq/2emFE0qD0U8cyu5fivlV1ZvHqD9CWLxqnqvwovhIL07ZmiXUGHMWmjsgtqXKsb9zSL08uwQP0432yKXwiliovgrihlhcc68  
[Issue](#)

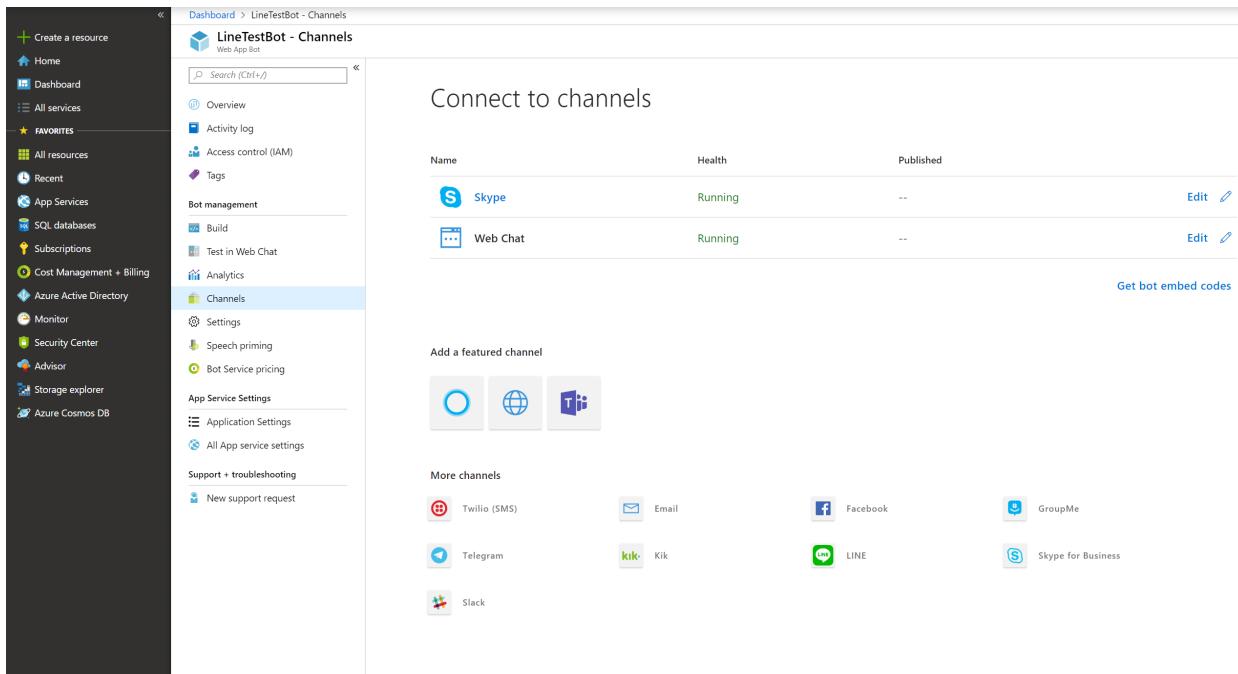
Use webhooks [?](#)  
Enabled [Edit](#)

Webhook URL Requires SSL [?](#)  
line.botframework.com/api/DR7rrK8EV0f [Verify](#) [Edit](#)

Allow bot to join group chats [?](#)  
Enabled [Edit](#)

## Подключение канала LINE к боту Azure

Войдите на [портал Azure](#), найдите нужный бот и щелкните для него **Каналы**.



Dashboard > LineTestBot - Channels

LineTestBot - Channels

Search (Ctrl+Shift+F)

All services

FAVORITES

All resources

Recent

App Services

SQL databases

Subscriptions

Cost Management + Billing

Azure Active Directory

Monitor

Security Center

Advisor

Storage explorer

Azure Cosmos DB

Create a resource

Home

Overview

Activity log

Access control (IAM)

Tags

Bot management

Build

Test in Web Chat

Analytics

Channels

Settings

Speech priming

Bot Service pricing

Name

Health

Published

S Skype Running -- Edit

Web Chat Running -- Edit

Add a featured channel

Twilio (SMS) Email Facebook GroupMe

Telegram Kik LINE Skype for Business

Slack

Get bot embed codes

Теперь выберите канал LINE и вставьте сохраненные ранее значения секрета канала и маркера доступа в соответствующие поля. Не забудьте сохранить изменения.

Скопируйте URL-адрес пользовательского веб-перехватчика, который вам предоставит Azure.

## Настройка параметров веб-перехватчика LINE

Теперь вернитесь к консоли разработчика LINE и вставьте URL-адрес веб-перехватчика Azure в поле **Message settings (Параметры сообщений) > Webhook URL (URL-адрес веб-перехватчика)**, а затем щелкните **Verify** (Проверить) для проверки подключения. Если вы только что создали канал в Azure, он может заработать только через несколько минут.

Теперь включите **Message settings (Параметры сообщений) > Use webhooks (Использовать веб-перехватчики)**.

### IMPORTANT

В консоли разработчика LINE следует сначала задать URL-адрес веб-перехватчика, и только затем выбрать **Use webhooks = Enabled** (Использовать веб-перехватчики = Включено). Если вы включите веб-перехватчики при пустом значении URL-адреса, состояние этого параметра не изменится, даже если в пользовательском интерфейсе будет указано иное.

Когда вы добавите URL-адрес веб-перехватчика и включите веб-перехватчики, обязательно перезагрузите страницу и убедитесь, что все изменения внесены правильно.

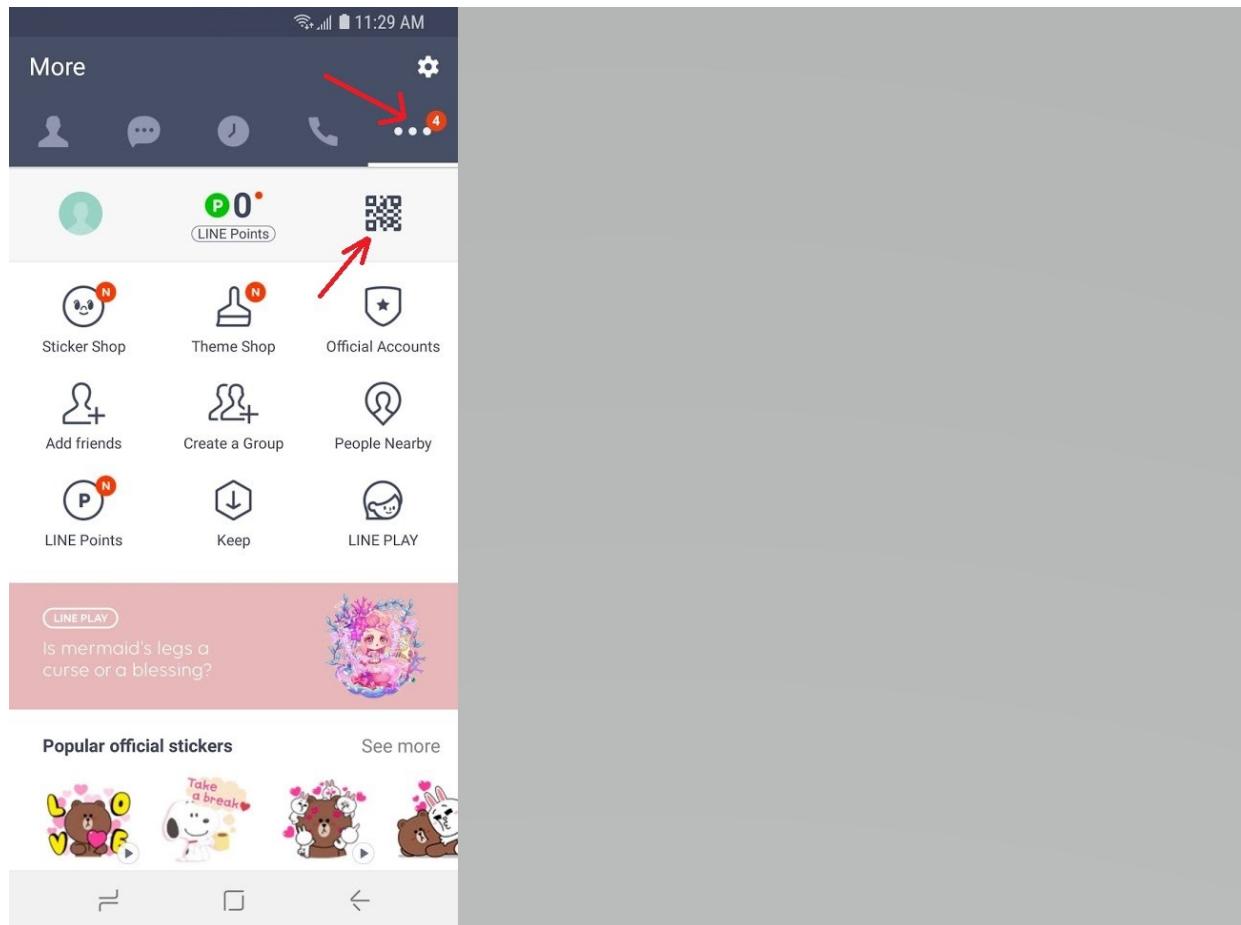
# Тестирование бота

После успешного завершения этих шагов бот сможет взаимодействовать с пользователями в приложении LINE, и теперь его можно протестировать.

## Добавление бота в мобильное приложение LINE

В консоли разработчика LINE перейдите на страницу параметров, где вы найдете QR-кода вашего бота.

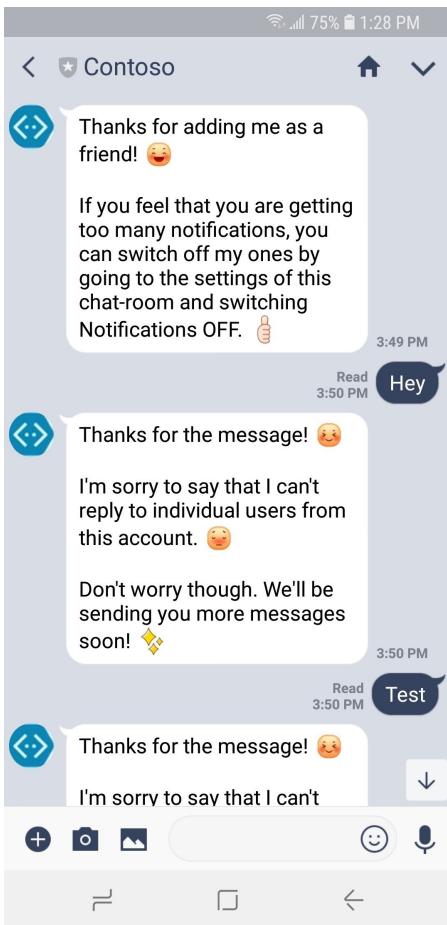
В мобильном приложении LINE перейдите на самую правую вкладку навигации, где изображены три точки [ ... ] и щелкните значок QR-кода.



Наведите сканер QR-кодов на QR-код, представленный в консоли разработчика. Теперь вы сможете взаимодействовать с ботом через мобильное приложение LINE, чтобы проверить работу бота.

## Автоматические сообщения

Начав тестирование бота, вы можете заметить неожиданные сообщения от бота, которые вы не указывали в действии `conversationUpdate`. Диалог может выглядеть примерно так:



Чтобы избежать отправки таких сообщений, следует отключить автоматическое ответные сообщения.

**LINE Developers** Products Documents News FAQ Community Blog

Using LINE@ features

Message text for LINE@ features are set on the LINE@ Manager.

Auto-reply messages [?](#)

Enabled  Disabled

[Update](#) [Cancel](#)

Greeting messages [?](#)

Disabled

[Set message](#) [Edit](#)

QR code of your bot

При необходимости вы можете сохранить эти сообщения. В этом случае, возможно, стоит нажать кнопку "задать сообщение" и изменить его.

The screenshot shows the LINE@ Manager interface. On the left, there's a sidebar with a user icon for 'Contoso @contosoBot' and a message count of '1'. Below the sidebar are several menu items: 'Compose message', 'Post to Timeline (Home)', 'Message' (which is expanded to show 'Message lists', 'Compose', 'Auto reply message', 'Keyword reply message', and 'Greeting message'), and 'Greeting message' (which is highlighted in green). The main content area is titled 'Greeting message' and contains a sub-instruction: 'This message will be sent automatically to users when they add you as a friend.' Below this is a text editor window with the placeholder text 'Thanks for adding me as a friend! (happy)' and a note: 'If you feel that you are getting too many notifications, you can switch off my ones by going to the settings of this chat-room and switching Notifications OFF. (ok)'. There's also an 'Emoji' button and a character limit indicator '207/500'. At the bottom of the text editor are six buttons: 'Text', 'Sticker', 'Photo', 'Coupon', 'Prize drawing page', and 'Polls & Surveys'. A note at the very bottom says: 'You can send up to five messages at once. Please select the messages you would like to send.'

## Устранение неполадок

- Если бот совсем не отвечает на сообщения, перейдите к нему на портале Azure и выберите действие "Тестирование в веб-чате".
  - Если здесь бот работает normally, но по-прежнему не отвечает в LINE, перезагрузите страницу консоли разработчика LINE и повторно выполните приведенные выше инструкции по настройке веб-перехватчика. Перед включением веб-перехватчиков необходимо настроить **URL-адрес веб-перехватчика**.
  - Если бот не работает даже в веб-чате, выполните отладку для устранения этой проблемы, а затем вернитесь и завершите настройку канала LINE.

# Подключение бота к Microsoft Teams

27.10.2020 • 2 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Боты, используемые в рабочей среде, следует добавлять в Microsoft Teams как часть приложения. Ознакомьтесь со сведениями о [создании бота](#) и [тестировании и отладке бота Microsoft Teams](#) в документах Microsoft Teams.

## Тестирование бота в Microsoft Teams

### IMPORTANT

Добавлять бота по GUID рекомендуется только для тестирования. В противном случае функциональность бота будет существенно ограничена. Боты, используемые в рабочей среде, следует добавлять в Teams как часть приложения. Ознакомьтесь со сведениями о [создании бота](#) и [тестировании и отладке бота Microsoft Teams](#) в документах Microsoft Teams.

Чтобы добавить канал Microsoft Teams, откройте бот на [портале Azure](#), щелкните колонку **Каналы** и выберите **Teams**.

The screenshot shows the Azure portal interface for managing a bot. On the left, there's a sidebar with various options like Overview, Activity log, Access control (IAM), Tags, Bot management, Build, Test in Web Chat, Analytics, and Channels. The 'Channels' option is highlighted with a red box. The main area is titled 'Connect to channels' and shows a table with one entry: 'Web Chat' under the 'Name' column and 'Running' under the 'Health' column. Below the table, there's a section for 'Add a featured channel' with four icons: a blue circle, a globe, a purple square with a white 'T', and a blue square with a white 'S'. The purple square with the 'T' is also highlighted with a red box.

Затем щелкните **Сохранить**.

## Configure MSTeams



Добавив канал Teams, перейдите на страницу **Каналы** и щелкните **Get bot embed code** (Получить код внедрения бота).

## Connect to channels

Name	Health	Published	
Microsoft Teams	Running	--	<a href="#">Edit</a>
Web Chat	Running	--	<a href="#">Edit</a>
<a href="#">Get bot embed codes</a>			

- Скопируйте часть кода с *https* в диалоговом окне **Получение кода внедрения бота**. Например,  
`https://teams.microsoft.com/l/chat/0/0?users=28:b8a22302e-9303-4e54-b348-343232`.
- В браузере вставьте этот адрес и выберите приложение Microsoft Teams (клиентское или веб-приложение), которое используется для добавления бота в Teams. Вы увидите бота как контакт, с которым вы можете обмениваться сообщениями в Microsoft Teams.

## Дополнительные сведения

См. подробнее о [Microsoft Teams](#).

# Подключение бота к Skype

27.10.2020 • 3 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Skype обеспечивает связь с пользователями через обмен сообщениями, телефон и видеозвонки. Расширьте эти функциональные возможности, создав боты, которые могут обнаруживать пользователи и с которыми они могут взаимодействовать через интерфейс Skype.

## NOTE

Начиная с 31 октября 2019 г. канал Skype не принимает новые запросы на публикацию ботов. Это означает, что вы можете разрабатывать боты с использованием канала Skype, но бот будет доступен только 100 пользователям. Вы не сможете опубликовать бота для большего числа пользователей. Текущие боты в Skype будут работать без прерываний. Узнайте больше о том, [почему некоторые функции недоступны в Skype](#).

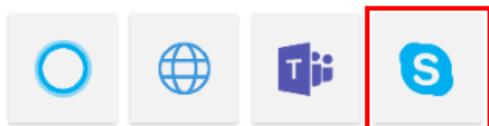
Чтобы добавить канал Skype, откройте бот на [портале Azure](#), щелкните колонку **Каналы** и выберите Skype.

## Connect to channels

Name	Health	Published	
 Web Chat	Running	--	<a href="#">Edit</a> 

[Get bot embed codes](#)

Add a featured channel



Откроется страница параметров **Configure Skype** (Настройка Skype).

# Configure



[Web control](#)

[Messaging](#)

[Calling](#)

[Groups](#)

[Publish](#)

## Web control

The Web Control lets you embed the bot in your own website.

[Get embed code](#)

Необходимо настроить параметры в разделах **Web control** (Веб-элемент управления), **Messaging** (Сообщения), **Calling** (Звонки), **Groups** (Группы) и **Publish** (Публикации). Рассмотрим подробнее каждый из разделов.

## Веб-элемент управления

Чтобы внедрить бот на веб-сайт, нажмите кнопку **Get embed code** (Получить код внедрения) в разделе **Web control** (Веб-элемент управления). Вы будете перенаправлены на страницу Skype для разработчиков. Следуйте инструкциям на этой странице, чтобы получить код внедрения.

## Обмен сообщениями

В этом разделе отправка и получение сообщений в боте.

## Вызов

В этом разделе настраивается возможность вызова Skype в боте. Вы можете указать, включен ли **вызов** для вашего бота и следует ли использовать интерактивные речевые ответы или мультимедиа в режиме реального времени.

## Группы

В этом разделе определяется то, можно ли бот добавить в группу, настраивается ли его поведение в группе для обмена сообщениями. Кроме того, этот раздел позволяет включить групповые видеозвонки для ботов вызова.

## Публикация

В этом разделе настраиваются параметры публикации бота. Все поля со звездочкой (\*) являются обязательными.

Для ботов в **предварительной версии** число контактов ограничено до 100. Если вам нужно больше контактов, отправьте бот для проверки. Когда вы щелкнете **Отправить на проверку**, бот автоматически станет доступным для поиска в Skype (если он будет принят). Если ваш запрос невозможно утвердить, вы получите уведомление о том, что нужно изменить для его утверждения.

**TIP**

Если вы хотите отправить бота для проверки, не забывайте о [контрольном списке для сертификации в Skype](#), которому бот должен соответствовать.

После завершения настройки, щелкните **Save** (Сохранить) и примите **условия предоставления услуг**. Канал Skype теперь добавлен к вашему боту.

## Дальнейшие действия

- [Skype для бизнеса](#)

# Подключение бота к Skype для бизнеса (предварительная версия)

27.03.2021 • 2 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Skype для бизнеса Online обеспечивает связь со своими сотрудниками и партнерами по бизнесу посредством мгновенных сообщений, телефонных звонков и видеосвязи. Расширьте эту функцию, создав боты, которых пользователи могут открывать и взаимодействовать через интерфейс Skype для бизнеса.

## IMPORTANT

**Канал Skype для бизнеса в Bot Framework был объявлен нерекомендуемым с 30 июня 2019 г.**

Канал Skype для бизнеса не принимает подключение новых ботов с 30 июня 2019 г. Существующие боты продолжали работу вплоть до 31 октября 2019 г. В настоящее время канал считается устаревшим, и его нельзя использовать в рабочих средах. Microsoft Teams — это предпочтительное средство связи от Майкрософт.

Узнайте, как [подключить бота к Microsoft Teams](#).

# Подключение бота к Slack

27.03.2021 • 17 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье показано, как добавить канал временного резерва в Bot. Выберите один из следующих способов настройки **приложения резервного времени**:

- подключение бота с помощью портала службы Azure Bot;
- использование адаптера Slack.
- [Портал службы Azure Bot](#)
- [Адаптер Slack](#)

## Создание приложения Slack для бота

1. В браузере [временной резервпри](#) входе.

2. Перейдите на панель [приложения](#).

App Name	Workspace	Distribution Status
botapp	mssc	Not distributed

3. Щелкните **Создать новое приложение**.

4. В поле **имя приложения** введите имя приложения резервного времени.

5. В поле **Группа резервного времени разработки** введите имя команды разработчиков. Если вы еще не являетесь членом команды разработчиков Slack, [создайте ее или присоединитесь к ней](#).

6. Нажмите кнопку **Create App** (Создать приложение).

### Добавить новый URL-адрес перенаправления

1. На левой панели выберите пункт меню "разрешения OAuth &".

2. На панели справа щелкните **Добавить новый URL-адрес перенаправления**.

3. В поле введите <https://slack.botframework.com/>.

4. Нажмите кнопку **Добавить**.

5. Нажмите кнопку Save URLs (Сохранить URL-адреса).

The screenshot shows the 'OAuth & Permissions' page in the Slack App Builder. On the left, a sidebar lists various app settings like 'Basic Information', 'Collaborators', and 'Features'. The 'OAuth & Permissions' section is currently selected. In the main area, there's a section for 'OAuth Tokens & Redirect URLs' with a note about automatic token generation. Below it, the 'Redirect URLs' section displays a single URL ('https://slack.botframework.com') with edit and delete options. At the bottom right of the main area is a prominent green 'Save URLs' button.

### Подписка на события бота

Выполните следующие действия для подписки на шесть определенных событий бота. При этом приложение будет получать уведомления о действиях пользователя по указанному вами URL-адресу.

#### TIP

Дескриптор бота — это его имя. Чтобы найти маркер Bot, перейдите на страницу <https://dev.botframework.com/bots>, выберите робот и запишите имя робота.

1. На левой панели выберите элемент подписки на **события**.
2. На панели справа установите для параметра **включить события** значение **вкл**.
3. В поле **URL-адрес запроса** введите значение  
`https://slack.botframework.com/api/Events/{YourBotHandle}`, где `{YourBotHandle}` обозначает дескриптор бота без фигурных скобок.

4. В разделе **Subscribe to Bot Events** (Подписаться на события ботов) щелкните **Add Bot User Event** (Добавить пользовательское событие бота).

5. В списке событий выберите следующие шесть типов событий:

- `member_joined_channel`
- `member_left_channel`
- `message.channels`
- `message.groups`
- `message.im`
- `message.mpim`

Event Name	Description	Required Scope
<code>member_joined_channel</code>	A user joined a public or private channel	<code>channels:read</code> or <code>groups:read</code>
<code>member_left_channel</code>	A user left a public or private channel	<code>channels:read</code> or <code>groups:read</code>
<code>message.channels</code>	A message was posted to a channel	<code>channels:history</code>
<code>message.groups</code>	A message was posted to a private channel	<code>groups:history</code>
<code>message.im</code>	A message was posted in a direct message channel	<code>im:history</code>
<code>message.mpim</code>	A message was posted in a multiparty direct message channel	<code>mpim:history</code>

6. В нижней части экрана нажмите кнопку **сохранить изменения**.

При добавлении событий в резервный временной диапазон выводится список областей, которые необходимо запросить. Необходимые области будут зависеть от событий, на которые вы подписаны, и от того, как вы планируете реагировать на них. Сведения о поддерживаемых резервных диапазонах см. в разделе [области и разрешения](#). См. также [Основные сведения об областях OAuth для программы-роботы](#).

## NOTE

По отношению к Июнь 2020. канал резервного времени поддерживает области разрешений версии 2, позволяющие Bot более детально указывать возможности и разрешения. Все вновь настроенные каналы временного резерва будут использовать области версии 2. Чтобы переключить программу-робот на области v2, удалите и повторно создайте конфигурацию канала резервного времени в колонке портал Azure каналы.

## Добавление и настройка интерактивных сообщений (необязательно)

Если ваш бот будет использовать функции Slack, например кнопки, выполните следующие действия.

1. Выберите вкладку **интерактивные & ярлыки** и включите **интерактивность**.
2. Введите `https://slack.botframework.com/api/Actions` в качестве **URL-адреса запроса**.
3. Нажмите кнопку **Save changes** (Сохранить изменения).

The screenshot shows the 'Interactivity & Shortcuts' configuration page. It includes sections for 'Interactivity' (with an 'On' toggle switch), 'Request URL' (set to `https://slack.botframework.com/api/Actions`), 'Shortcuts' (with a 'Create New Shortcut' button), and 'Select Menus' (with an 'Options Load URL' field set to `https://my.app.com/slack/options-load-endpoint`). At the bottom are 'Discard Changes' and 'Save Changes' buttons, with 'Save Changes' being highlighted by a red box.

## Добавление канала резервного времени в Bot

1. На панели слева выберите элемент **Основные сведения**.
2. На панели справа перейдите к разделу **учетные данные приложения**. Отобразятся **идентификатор клиента, секрет клиента и секрет подписи**, необходимые для настройки канала ленты времени создания резервной копии. Скопируйте и сохраните эти учетные данные в надежном месте.

## App Credentials

These credentials allow your app to access the Slack API. They are secret. Please don't share your app credentials with anyone, include them in public code repositories, or store them in insecure ways.

App ID	Date of App Creation
A010FRKDZJQ	March 20, 2020
Client ID	
[REDACTED]	
Client Secret	Show Regenerate
*****	Show Regenerate
You'll need to send this secret along with your client ID when making your oauth.v2.access request.	
Signing Secret	
*****	Show Regenerate
Slack signs the requests we send you using this secret. Confirm that each request comes from Slack by verifying its unique signature.	
Verification Token	
[REDACTED]	Regenerate
This deprecated Verification Token can still be used to verify that requests come from Slack, but we strongly recommend using the above, more secure, signing secret instead.	

3. Откройте программу Bot в [портал Azure](#).
4. На панели слева выберите **каналы**,
5. На правой панели выберите значок **временной резерв**.
6. Вставьте в соответствующие поля учетные данные приложения резервного копирования, сохраненные на предыдущих шагах.
7. **URL-адрес целевой страницы** является необязательным. Его можно опустить или изменить.

Enter your Slack credentials  
[Step-by-step instructions to add the bot to Slack](#)

Client ID *	[REDACTED]
Client Secret *	*****
Signing Secret *	*****
Landing Page URL (optional)	Users will be redirected to this URL after adding your bot to Slack
<input type="button" value="Cancel"/> <input type="button" value="Save"/> <input checked="" type="checkbox"/> Disabled	

8. Выберите команду **Сохранить**. Следуйте инструкциям, чтобы авторизовать доступ приложения Slack к команде разработчиков Slack.
9. На странице Настройка временного резерва убедитесь, что ползунок на кнопке Сохранить установлен в значение **включено**. Теперь программа Bot настроена для взаимодействия с пользователями в временной резерв.

### Создание кнопки Add to Slack (Добавить в Slack)

Slack предоставляет HTML-код, с помощью которого можно упростить поиск вашего бота в разделе *Add the Slack button* (Создание кнопки "Добавить в Slack") на [этой странице](#). Чтобы использовать этот HTML-код с ботом, замените значение href (начинается с `https://`) URL-адресом, найденным в параметрах

канала Slack вашего бота. Выполните следующие действия, чтобы получить URL-адрес на замену.

1. В <https://dev.botframework.com/bots> щелкните элемент Bot.
2. Выберите Channels (Каналы), щелкните правой кнопкой мыши запись с именем Slack и выберите пункт Copy link (Копировать ссылку). Этот URL-адрес теперь находится в буфере обмена.
3. Вставьте его из буфера обмена в HTML-код, предоставленный для кнопки Slack. Этот URL-адрес заменяет значение href, предоставленное Slack для этого бота.

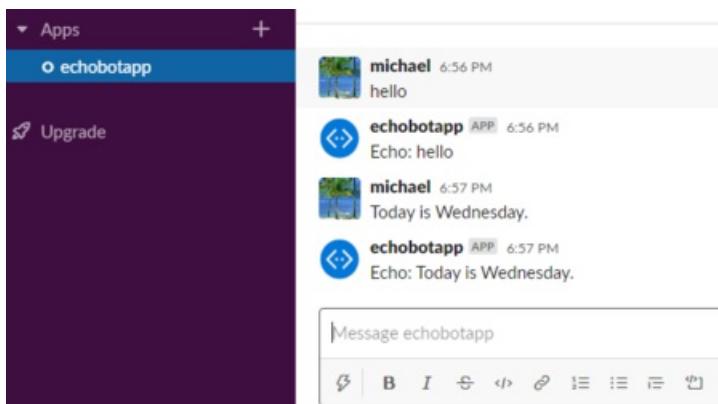
Авторизованные пользователи могут нажать кнопку Add to Slack (Добавить в Slack), предоставленную этим измененным HTML, для доступа к боту в Slack.

#### NOTE

Ссылка, вставленная в значение href элемента HTML, содержит области, которые могут быть уточнены по мере необходимости. Полный список доступных областей см. в разделе [области и разрешения](#).

## Тестирование приложения в временной резерв

1. Войдите в рабочее пространство временного резерва, в которое вы установили приложение (<http://<your work space>-group.slack.com/>). Он будет отображаться в разделе **приложения** на левой панели.
2. На панели слева выберите свое приложение.
3. На панели справа напишите сообщение и отправьте его в приложение. При использовании эхоробота приложение возвращает сообщение, как показано на рисунке ниже.



Эту функцию также можно протестировать с помощью [примера бота для адаптера Slack](#), заполнив файл appSettings.json теми же значениями, что и в описанных выше шагах. В файле README для этого примера есть дополнительные действия для предоставления ссылок, получения вложений и отправки интерактивных сообщений.

# Подключение бота к Telegram

27.10.2020 • 2 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

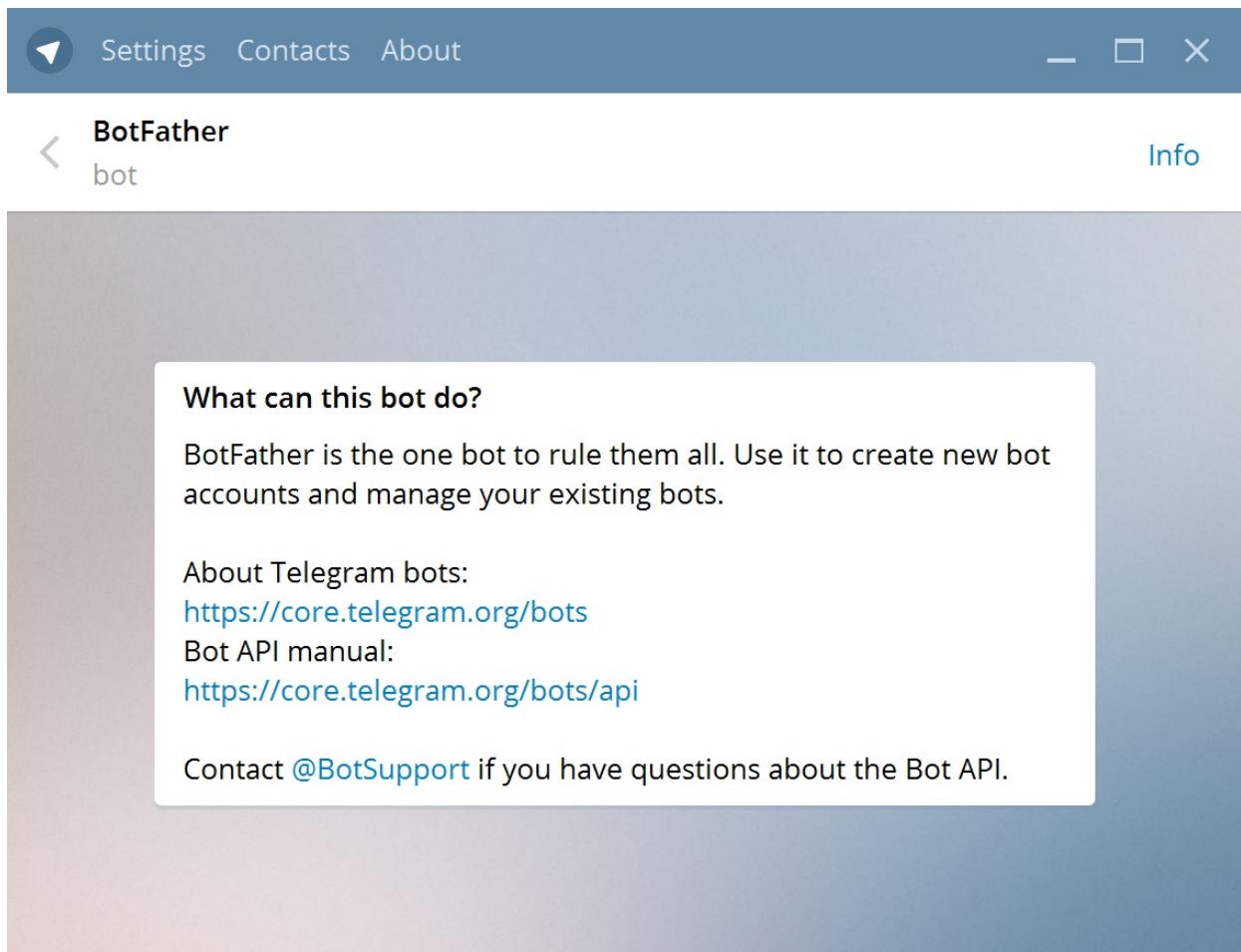
Вы можете настроить взаимодействие бота с другими пользователями с помощью программы для обмена сообщениями Telegram.

## TIP

Описание функций, поддерживаемых в каждом канале, см. в статье [Разделенные на категории действия по каналам](#).

## Получение доступа к BotFather для создания бота Telegram

[Создайте бот Telegram](#) с помощью BotFather.



Start

## Создание бота Telegram

Чтобы создать бот Telegram, отправьте команду `/newbot`.



Settings Contacts About

— □ ×



BotFather

bot

Info

<https://core.telegram.org/bots>

You can control me by sending these commands:

[/newbot](#) - create a new bot  
[/token](#) - generate authorization token  
[/revoke](#) - revoke bot access token  
[/setname](#) - change a bot's name  
[/setdescription](#) - change bot description  
[/setabouttext](#) - change bot about info  
[/setuserpic](#) - change bot profile photo  
[/setinline](#) - change inline settings  
[/setinlinefeedback](#) - change inline feedback settings  
[/setcommands](#) - change bot commands list  
[/setjoininggroups](#) - can your bot be added to groups?  
[/setprivacy](#) - what messages does your bot see in groups?  
[/deletebot](#) - delete a bot



/newbot

create a new bot



/newbot



Send

#### Указание понятного имени

Присвойте боту Telegram понятное имя.



BotFather

bot

Info

/revoke - revoke bot access token  
/setname - change a bot's name  
/setdescription - change bot description  
/setabouttext - change bot about info  
/setuserpic - change bot profile photo  
/setinline - change inline settings  
/setinlinefeedback - change inline feedback settings  
/setcommands - change bot commands list  
/setjoininggroups - can your bot be added to groups?  
/setprivacy - what messages does your bot see in groups?  
/deletebot - delete a bot  
/cancel - cancel the current operation

1:44 PM

/newbot 1:44 PM ✓

Alright, a new bot. How are we going to call it? Please choose a name for your bot.

1:44 PM



Delightful Bot



Send

#### Указание имени пользователя

Присвойте боту Telegram уникальное имя пользователя.



/setinline - change inline settings  
/setinlinefeedback - change inline feedback settings  
/setcommands - change bot commands list  
/setjoininggroups - can your bot be added to groups?  
/setprivacy - what messages does your bot see in groups?  
/deletebot - delete a bot  
/cancel - cancel the current operation

1:44 PM

/newbot 1:44 PM ✓✓

Alright, a new bot. How are we going to call it? Please choose a name for your bot.

1:44 PM

Delightful Bot 1:44 PM ✓✓

Good. Now let's choose a username for your bot. It must end in `bot`. Like this, for example: TetrisBot or tetris\_bot.

1:44 PM



DelightfulBot



Send

### Копирование маркера доступа

Скопируйте маркер доступа бота Telegram.



BotFather  
bot

Info

Good. Now let's choose a username for your bot. It must end in `bot`. Like this, for example: TetrisBot or tetris\_bot.

1:44 PM

Delightful Bot 1:44 PM ✓

Done! Congratulations on your new bot. You will find it at [telegram.me/DelightfulBot](https://telegram.me/DelightfulBot). You can now add a description, about section and profile picture for your bot, see [/help](#) for a list of commands.

Use this token to access the HTTP API:  
183547168:AAHB5Ne2yzV5qfOvgAAgW0DHWRG0OjQLDEg

For a description of the Bot API, see this page:  
<https://core.telegram.org/bots/api>

1:45 PM

DelightfulBot 1:45 PM ✓



## Ввод маркера доступа бота Telegram

Перейдите к разделу **Каналы** бота на портале Azure и нажмите кнопку **Telegram**.

### NOTE

Пользовательский интерфейс портала Azure будет выглядеть немного иначе, когда вы подключите бота к Telegram.

Web App Bot

- Overview
- Activity log
- Access control (IAM)
- Tags

### Bot management

- Build
- Test in Web Chat
- Analytics
- Channels
- Settings
- Speech priming
- Bot Service pricing

### App Service Settings

Search (Ctrl+ /)

Add a featured channel

More channels

- Direct Line Speech
- Email
- Facebook
- GroupMe
- Kik
- LINE
- Slack
- Twilio (SMS)
- Telegram

The 'Telegram' icon is highlighted with a red box.

Вставьте ранее скопированный маркер в поле **Маркер доступа** и нажмите кнопку **Сохранить**.

Web App Bot

- Overview
- Activity log
- Access control (IAM)
- Tags

### Bot management

- Build
- Test in Web Chat
- Analytics
- Channels
- Settings
- Speech priming
- Bot Service pricing

### App Service Settings

Search (Ctrl+ /)

## Configure Telegram

Enter your Telegram credentials  
[Step-by-step instructions to add the bot to Telegram.](#)

AccessToken \*

AccessToken can be found in your Telegram account setting

Cancel Save Disabled Delete Channel

Бот настроен для взаимодействия с пользователями в Telegram.

Saved.

[Overview](#)[Activity log](#)[Access control \(IAM\)](#)[Tags](#)**Bot management**[Build](#)[Test in Web Chat](#)[Analytics](#)[Channels](#)[Settings](#)[Speech priming](#)[Bot Service pricing](#)**App Service Settings**[Configuration](#)[All App service settings](#)

Enter your Telegram credentials

[Step-by-step instructions to add the bot to Telegram.](#)

Access Token \*

[Cancel](#)[Save](#)[Enabled](#)[Delete Channel](#)

# Подключение программы-робота к телефонии (Предварительная версия)

27.03.2021 • 2 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Канал телефонии использует мощь Microsoft Bot Framework в сочетании со службами связи Azure и службами Microsoft Speech Services для обеспечения возможности телефонных вызовов в роботе.

Дополнительные сведения см. в [проекте телефонии Bot Framework](#).

# Подключение бота к Twilio

27.03.2021 • 12 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Вы можете настроить взаимодействие бота с пользователями с помощью облачной платформы обмена данными Twilio.

## Вход в учетную запись Twilio или ее создание для отправки и получения SMS-сообщений

Если у вас нет учетной записи Twilio, [создайте ее](#).

## Создание приложения TwiML

[Создайте приложение TwiML](#) в соответствии с инструкциями.

Create TwiML App

Properties

FRIENDLY NAME	My TwiML app
Voice	
REQUEST URL	HTTP POST
Messaging	
REQUEST URL	HTTP POST
<button>Save</button> <button>Cancel</button>	

В разделе **Свойства**, введите FRIENDLY NAME (Понятное имя). В этом руководстве для примера используется имя "My TwiML app". Поле REQUEST URL (URL-адрес запроса) под Voice можно оставить пустым. В разделе **Messaging** (Обмен сообщениями) для параметра Request URL (URL-адрес запроса) введите значение <https://sms.botframework.com/api/sms>.

## Выбор или добавление номера телефона

Следуйте предоставленным [здесь](#) инструкциям, чтобы добавить идентификатор проверенного вызывающего объекта через сайт консоли. После завершения вы увидите проверенный номер в поле Active Numbers (Активные номера) раздела Manage Numbers (Управление номерами).

### Active Numbers

CLICK + TO BUY NEW NUMBER			
Number	Voice URL	CAPABILITIES	
+1 610 609 8967 Havertown, PA	(610) 609-8967		
CONFIGURATION			Voice POST: <a href="https://demo.twilio.com/welcome/voice/">https://demo.twilio.com/welcome/voice/</a> Messaging POST: <a href="https://demo.twilio.com/welcome/sms/reply/">https://demo.twilio.com/welcome/sms/reply/</a>

## Указание приложений для голосовой связи и обмена

## сообщениями

Щелкните число и перейдите к разделу **Настройка**. Для голосовой связи и обмена сообщениями укажите в поле **CONFIGURE WITH** (Настройка с помощью) значение "TwiML App", а в поле **TWIML APP** (Приложение TwiML) выберите "My TwiML app". Завершив настройку, щелкните **Сохранить**.

(610) 609-8967

FRIENDLY NAME (610) 609-8967

SID PNb7228f7f793c08cc22169b60d3202182

PHONE NUMBER +16106098967

LOCATION Havertown, PA US

CAPABILITIES Voice, SMS, MMS

Voice

Configure With TwiML App

Twiml App My TwiML app

Messaging

Configure With TwiML App

Twiml App My TwiML app

Save Cancel Release this Number

Вернитесь к разделу **Управление номерами**. Вы увидите, что для голосовой связи и обмена сообщениями теперь в конфигурации указано приложение TwiML.

## Active Numbers

+ CLICK + TO BUY NEW NUMBER

Number Voice URL

NUMBER FRIENDLY NAME CAPABILITIES

+1 610 609 8967 (610) 609-8967

CONFIGURATION

Voice: TwiML App: My TwiML app  
Messaging: TwiML App: My TwiML app

## Получение учетных данных

Вернитесь к [домашней странице консоли](#), где на панели мониторинга проекта вы найдете идентификатор безопасности учетной записи и маркер проверки подлинности, как показано ниже.

Dashboard / UPGRADE

ProgrammableSMS Dashboard

Project Info

We've customized your dashboard based on the products you selected. Use the product getting started guides to get up and running.  
We can't wait to see what you build!

PROJECT NAME ProgrammableSMS edit  
ACCOUNT SID ACea27a63cc0192f429e7763062440749e  
AUTH TOKEN hide 7b2439a5536ae778724e52c36a35568f  
Owner 1 manage  
2FA Disabled edit

Programmable SMS Invite Your Team

# Отправка учетных данных

В отдельном окне браузера откройте сайт Bot Framework, расположенный по адресу <https://dev.botframework.com/>.

- Щелкните **My bots** (Мои боты) и выберите бот, который нужно подключить к Twilio. Это действие перенесет вас на портал Azure.
- Выберите **Каналы** в разделе **Управление ботами**. Щелкните значок Twilio (SMS).
- Введите номер телефона, идентификатор безопасности учетной записи и маркер проверки подлинности, которые вы записали ранее. Завершив настройку, щелкните **Сохранить**.

Home > 1stWebAppBot - Channels

## 1stWebAppBot - Channels

Web App Bot

Search (Ctrl+ /)

Overview

Activity log

Access control (IAM)

Tags

Bot management

Build

Test in Web Chat

Analytics

Channels

Settings

Speech priming

Bot Service pricing

App Service Settings

Application Settings

All App service settings

Support + troubleshooting

New support request

Enter your Twilio credentials

Step-by-step instructions to add the bot to Twilio.

Phone Number

+5(555)555-5555

Account Sid

Account SID can be found in your Twilio account settings

Auth Token

Auth token can be found in your Twilio account settings

Twilio (SMS) configured but still not working?

Look at Twilio logs

Cancel

Save

Disabled

Как только вы выполните эти действия, ваш бот будет настроен для взаимодействия с пользователями с помощью Twilio.

## Подключение бота к Twilio с помощью адаптера Twilio

Для подключения бота к Twilio можно использовать не только канал, доступный в службе Azure Bot, но и адаптер Twilio. Из этой статьи вы узнаете, как подключить бота к Twilio с помощью адаптера. Здесь представлена пошаговая инструкция, которая позволит изменить пример EchoBot для подключения его к Twilio.

### NOTE

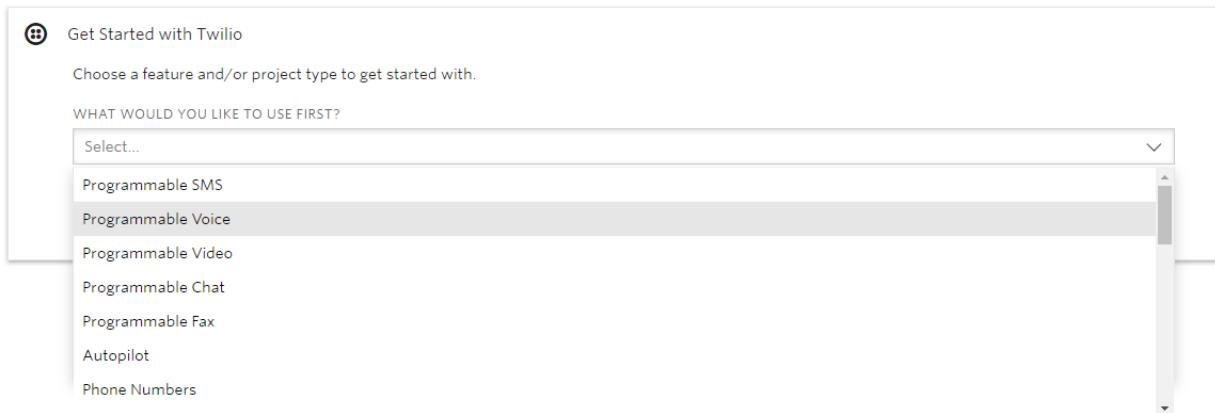
Ниже приводятся инструкции для реализации адаптера Twilio на C#. Инструкции по использованию адаптера для JS, который входит в состав библиотек BotKit, см. в [документации по BotKit для Twilio](#).

### Предварительные требования

- [Пример кода EchoBot](#).
- Учетная запись Twilio. Если у вас нет учетной записи Twilio, [создайте ее](#).

### Получение номера Twilio и сбор учетных данных учетной записи

1. Войдите в [Twilio](#). В правой части страницы вы увидите параметры ACCOUNT SID (ИД безопасности учетной записи) и AUTH TOKEN (Маркер аутентификации) для вашей учетной записи. Запишите их, так как они понадобятся позднее при настройке приложения бота.
2. Выберите элемент **Programmable Voice** (Программируемый голос) из раздела **Get Started with Twilio** (Начало работы с Twilio).



3. На следующей странице нажмите кнопку **Get your first Twilio number** (Получить первый номер Twilio). Во всплывающем окне отобразится новый номер. Вы можете принять его щелчком по кнопке **Choose this number** (Выбрать этот номер), или найти другой номер, следуя инструкциям на экране.
4. Завершив выбор номера, запишите его, так как он потребуется позже при настройке приложения бота.

### Подключение адаптера Twilio к боту

Теперь, когда у вас есть номер Twilio и учетные данные, можно настроить приложение бота.

#### Установка пакета NuGet для адаптера Twilio

Добавьте пакет NuGet [Microsoft.Bot.Builder.Adapters.Twilio](#). Подробные сведения об использовании NuGet см. в руководстве по [установке пакетов и управлении ими в Visual Studio](#).

#### Создание класса адаптера Twilio

Создайте новый класс, наследующий от класса `*twilioadapter_`. Этот класс выступает в качестве адаптера для канала Twilio и включает возможности обработки ошибок (аналогично классу `*ботфрамворкадаптервиссеррорхандлер*`, который уже есть в примере, используемом для обработки других запросов от службы Azure Bot).

```

public class TwilioAdapterWithErrorHandler : TwilioAdapter
{
    public TwilioAdapterWithErrorHandler(IConfiguration configuration, ILogger<BotFrameworkHttpAdapter>
logger)
        : base(configuration, logger)
    {
        OnTurnError = async (turnContext, exception) =>
        {
            // Log any leaked exception from the application.
            logger.LogError(exception, $"[OnTurnError] unhandled error : {exception.Message}");

            // Send a message to the user
            await turnContext.SendActivityAsync("The bot encountered an error or bug.");
            await turnContext.SendActivityAsync("To continue to run this bot, please fix the bot source
code.");

            // Send a trace activity, which will be displayed in the Bot Framework Emulator
            await turnContext.TraceActivityAsync("OnTurnError Trace", exception.Message,
"https://www.botframework.com/schemas/error", "TurnError");
        };
    }
}

```

#### **Создание контроллера для обработки запросов Twilio**

Создайте контроллер, который будет выполнять запросы из приложения Twilio, в новой конечной точке api/twilio вместо api/messages, которая используется по умолчанию для запросов от каналов службы Azure Bot. Добавив к боту дополнительную конечную точку, вы сможете с помощью одного бота принимать запросы одновременно от каналов службы Bot и Twilio.

```

[Route("api/twilio")]
[ApiController]
public class TwilioController : ControllerBase
{
    private readonly TwilioAdapter _adapter;
    private readonly IBot _bot;

    public TwilioController(TwilioAdapter adapter, IBot bot)
    {
        _adapter = adapter;
        _bot = bot;
    }

    [HttpPost]
    [HttpGet]
    public async Task PostAsync()
    {
        // Delegate the processing of the HTTP POST to the adapter.
        // The adapter will invoke the bot.
        await _adapter.ProcessAsync(Request, Response, _bot, default);
    }
}

```

#### **Включение адаптера Twilio в файл startup.cs бота**

Добавьте следующую строку в метод *ConfigureServices* в файле startup.cs. Это действие регистрирует адаптер Twilio и делает его доступным для нового класса контроллера. Добавленные на предыдущем шаге параметры конфигурации применяются адаптером автоматически.

```
services.AddSingleton<TwilioAdapter, TwilioAdapterWithErrorHandler>();
```

После добавления метод *ConfigureServices* должен выглядеть следующим образом.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

    // Create the default Bot Framework Adapter (used for Azure Bot Service channels and Emulator).
    services.AddSingleton<IBotFrameworkHttpAdapter, BotFrameworkAdapterWithErrorHandler>();

    // Create the Twilio Adapter
    services.AddSingleton<TwilioAdapter, TwilioAdapterWithErrorHandler>();

    // Create the bot as a transient. In this case the ASP Controller is expecting an IBot.
    services.AddTransient<IBot, EchoBot>();
}
```

#### Получение URL-адреса для бота

Теперь, когда вы настроили адаптер в проекте бота, необходимо найти правильную конечную точку и передать ее в Twilio, чтобы бот мог получать сообщения. Этот URL-адрес также потребуется для завершения настройки приложения бота.

Чтобы выполнить этот шаг, [разверните бота в Azure](#) и запишите URL-адрес этого развертывания.

#### NOTE

Если вы не готовы к развертыванию Bot в Azure или хотите отладить программу Bot при использовании адаптера Twilio, вы можете использовать такое средство, как [ngrok](#) (которое уже было установлено, если ранее использовался эмулятор Bot Framework) для туннелирования к запущенному на локальном компьютере интерфейсу Bot и предоставит вам общедоступный URL-адрес для этого.

Если вы хотите создать туннель и получить для бота URL-адрес с помощью ngrok, выполните следующую команду в окне терминала. Здесь предполагается, что локальный бот работает на порту 3978. Если это не так, измените номера портов в команде.

```
ngrok.exe http 3978 -host-header="localhost:3978"
```

#### Добавление параметров приложения Twilio в файл конфигурации бота

Добавьте перечисленные ниже параметры в файл appSettings.json в проекте бота. Вы заполняете `_*` Твilioнумбер `**`, **твilioаккаунтсид** и **твilioаутстокен**, используя значения, собранные при создании номера Twilio. В поле `TwilioValidationUrl` введите URL-адрес бота и конечную точку `api/twilio`, которую вы указали в только что созданном контроллере. Например,

```
https://yourboturl.com/api/twilio .
```

```
"TwilioNumber": "",  
"TwilioAccountSid": "",  
"TwilioAuthToken": "",  
"TwilioValidationUrl": ""
```

Теперь, когда вы заполнили указанные выше параметры, следует повторно развернуть бота или просто перезапустить его, если вы используете туннель ngrok к локальной среде.

#### Завершение настройки номера Twilio

Заключительным этапом является настройка конечной точки отправки сообщений в Twilio, чтобы бот мог получать из нее сообщения.

1. Перейдите на [страницу Active Numbers \(Активные номера\)](#) в Twilio.
2. Щелкните номер телефона, который вы создали на предыдущем шаге.

3. В разделе **Messaging** (Сообщения) заполните раздел **A MESSAGE COMES IN** (Поступает сообщение), выбрав **Webhook** (Веб-перехватчик) из раскрывающегося списка и заполнив текстовое поле значением конечной точки бота, которое вы указали в параметре **TwilioValidationUrl** на предыдущем шаге, например <https://yourboturl.com/api/twilio>.

## Messaging

CONFIGURE WITH

Webhooks, TwiML Bins, Functions, Studio, or Proxy

A MESSAGE COMES IN

Webhook

<https://yourboturl.com/api/twilio>

4. Нажмите кнопку **Сохранить**.

### Тестирование работы бота с адаптером в Twilio

Теперь вы можете проверить, правильно ли подключен бот к Twilio, отправив SMS-сообщение на свой номер Twilio. Каждый раз, когда ваш бот получает сообщение, он отправляет вам ответ с копией текста вашего сообщения.

Эту функцию также можно протестировать с помощью [примера бота для адаптера Twilio](#), заполнив файл appSettings.json теми же значениями, что и в описанных выше шагах.

# Подключение бота к WeChat

27.03.2021 • 5 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Вы можете настроить взаимодействие бота с пользователями с помощью официальной платформы учетных записей WeChat.

## Скачивание адаптера WeChat для Bot Framework

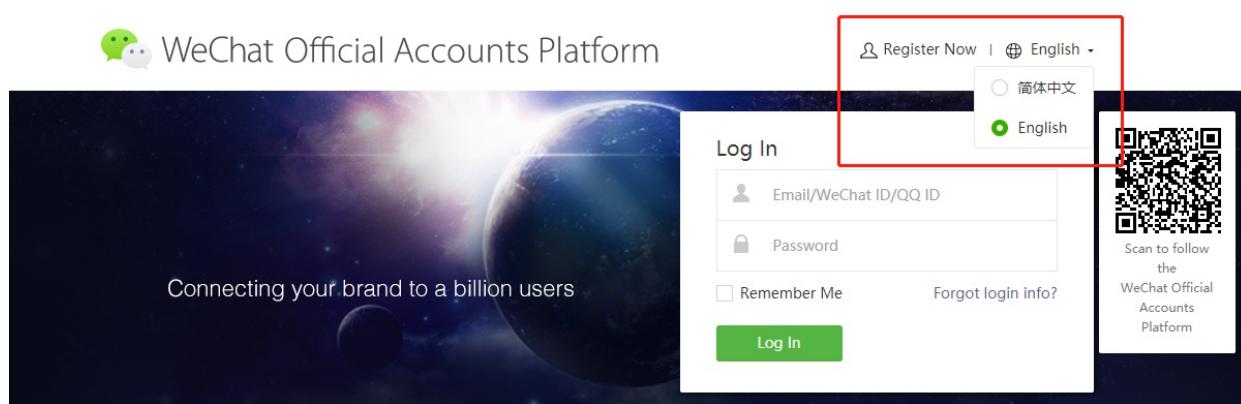
WeChat для Microsoft Bot Framework представляет собой адаптер с открытым исходным кодом, который размещен на сайте GitHub. Скачать адаптер WeChat для Bot Framework можно [здесь](#).

## Создание учетной записи WeChat

Чтобы настроить бота для общения через WeChat, необходимо создать официальную учетную запись WeChat на [платформе официальных учетных записей WeChat](#), а затем подключить бота к приложению. В настоящее время поддерживается только учетная запись службы.

### Изменение предпочтаемого языка

Вы можете изменить предпочтаемый язык интерфейса, прежде чем выполнять вход.



Notices • 关于小程序恶意对抗平台规则的违规行为公告 NEW • 公众号文章内图片跳转的调整通知 NEW View more >

### Account Types



#### Service Account

Gives companies and organizations powerful business services and user management capabilities to quickly reach and better serve users.



#### Subscription Account

Provides media and individuals a new way to post stories and information, innovating author-reader interaction and content management.



#### Mini Program

Empowers developers and businesses to revolutionize their services on WeChat with convenient development and an excellent user experience.



#### WeChat Work (Formerly "Enterprise Account")

This professional office management tool helps companies optimize operations and management, offering the same WeChat communication experience you've come to love, a robust suite of free office productivity features, and interoperability with WeChat messaging, Mini Programs, and WeChat Pay.

## Регистрация учетной записи службы

Реальная учетная запись службы должна быть проверена с помощью WeChat. Вы не сможете включить веб-перехватчик до проверки учетной записи. Следуйте [этим](#) инструкциям, чтобы создать учетную запись службы. Для ускорения процесса просто щелкните ссылку на регистрацию в верхней части страницы, выберите Service Account (Учетная запись службы) и следуйте инструкциям.

The screenshot shows the 'Register' page of the WeChat Official Accounts Platform. At the top, there is a message: 'Select an account type in order to register'. Below this, there are four options:

- Subscription Account**: Broadcast content to all of your followers. Best suited for media and individual publishers.
- Service Account**: Offer online services and manage users. Available to businesses and organizations.
- Mini Program**: Distribute services from within WeChat while maintaining an excellent user experience. Available to businesses and organizations with service offerings.
- WeChat Work**: Formerly. Helping businesses with internal communication and office efficiency. Suitable for registration of corporate customer.

At the bottom of the registration form, there is a note: 'Don't know what account type to select? please consult the differentiation of the account type'.

At the very bottom of the page, there are links to 'About Tencent', 'Service Agreement', and 'Tencent Customer Service', followed by a copyright notice: 'Copyright © 2012-2019 Tencent. All Rights Reserved.'

## Учетная запись песочницы

Если вы просто хотите протестировать интеграцию бота с WeChat, вместо новой учетной записи службы можно использовать учетную запись песочницы. Дополнительные сведения об учетной записи песочницы см. в [этой статье](#).

## Включение адаптера WeChat для бота

Проект Bot представляет собой обычный проект на основе пакета SDK для Bot Framework версии 4. Прежде чем открывать проект, необходимо попробовать запустить бота. Скачайте [адаптер WeChat для Bot Framework](#).

### Предварительные требования

- Пакет SDK для .NET Core (версия 2.2. x)

### Добавление ссылки на источник адаптера WeChat

Используйте прямую ссылку на проект адаптера WeChat или добавьте ~/BotFramework-WeChat/libraries/csharp\_dotnetcore/outputpackages в качестве локального источника NuGet.

### Включение адаптера WeChat в файл Startup.cs бота

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);

    // Create the storage we'll be using for User and Conversation state. (Memory is great for testing purposes.)
    services.AddSingleton<IStorage, MemoryStorage>();

    // Create the User state. (Used in this bot's Dialog implementation.)
    services.AddSingleton<UserState>();

    // Create the Conversation state. (Used by the Dialog system itself.)
    services.AddSingleton<ConversationState>();

    // Load WeChat settings.
    var wechatSettings = new WeChatSettings();
    Configuration.Bind("WeChatSettings", wechatSettings);
    services.AddSingleton<WeChatSettings>(wechatSettings);

    // Configure hosted service.
    services.AddSingleton<IBackgroundTaskQueue, BackgroundTaskQueue>();
    services.AddHostedService<QueuedHostedService>();
    services.AddSingleton<WeChatHttpAdapter>();

    // The Dialog that will be run by the bot.
    services.AddSingleton<MainDialog>();

    // Create the bot as a transient. In this case the ASP Controller is expecting an IBot.
    services.AddTransient<IBot, EchoBot>();
}
```

### Обновление контроллера бота

```
[Route("api/messages")]
[ApiController]
public class BotController : ControllerBase
{
    private readonly IBot _bot;
    private readonly WeChatHttpAdapter _weChatHttpAdapter;
    private readonly string Token;
    public BotController(IBot bot, WeChatHttpAdapter weChatAdapter)
    {
        _bot = bot;
        _weChatHttpAdapter = weChatAdapter;
    }

    [HttpPost("/WeChat")]
    [HttpGet("/WeChat")]
    public async Task PostWeChatAsync([FromQuery] SecretInfo secretInfo)
    {
        // Delegate the processing of the HTTP POST to the adapter.
        // The adapter will invoke the bot.
        await _weChatHttpAdapter.ProcessAsync(Request, Response, _bot, secretInfo);
    }
}
```

### Настройка appsettings.json

Вам необходимо настроить файл appsettings.json, как описано ниже, прежде чем запускать бота.

```

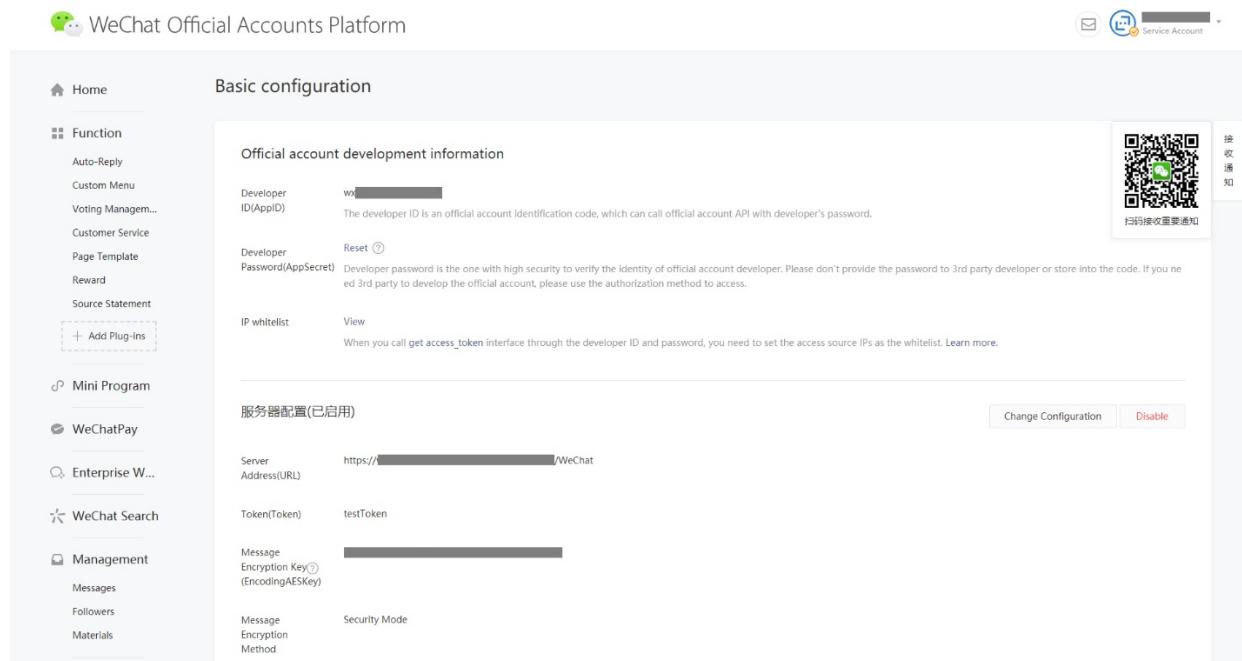
"WeChatSettings": {
    "UploadTemporaryMedia": true,
    "PassiveResponseMode": false,
    "Token": "",
    "EncodingAESKey": "",
    "AppId": "",
    "AppSecret": ""
}

```

### Учетная запись службы

Если у вас уже есть учетная запись службы и вы готовы развернуть бота, получите значения **AppID** (идентификатор приложения), **AppSecret** (секрет приложения), **EncodingAESKey** (ключ шифрования) и **Token** (токен) на панели навигации слева, как показано ниже.

Не помню, что вам нужно настроить список разрешений IP-адресов, иначе WeChat не будет принимать ваш запрос.



The screenshot shows the 'Basic configuration' section of the WeChat Official Accounts Platform. On the left sidebar, under 'Function', 'Auto-Reply', 'Custom Menu', 'Voting Managem...', 'Customer Service', 'Page Template', 'Reward', and 'Source Statement' are listed. Under 'Mini Program', 'WeChatPay' is selected. Under 'Management', 'Messages', 'Followers', and 'Materials' are listed. The main content area is titled 'Official account development information'. It contains fields for 'Developer ID(AppID)' (wx[REDACTED]), 'Developer Password(AppSecret)' (Reset [?] [REDACTED]), and 'IP whitelist' (View). A QR code is displayed with the text 'Scan to receive important notifications'. Below this is a section titled '服务器配置(已启用)' (Server Configuration Enabled) with fields for 'Server Address(URL)' (https://[REDACTED]/WeChat), 'Token(Token)' (testToken), 'Message Encryption Key(?) (EncodingAESKey)' ([REDACTED]), and 'Message Encryption Method' (Security Mode). Buttons for 'Change Configuration' and 'Disable' are at the bottom right of this section.

### Учетная запись песочницы

У учетной записи песочницы нет параметра **EncodingAESKey**, то есть сообщения от WeChat не шифруются. Для нее просто оставьте поле **EncodingAESKey** пустым. В этом случае нужно указать только три параметра: **appId** (идентификатор приложения), **appsecret** (секрет приложения) и **Token** (токен).

### Test Account Management

WeChat number: gh\_... Sign out

**Test account information**

appId	Wx...
Appsecret	2e...

**Interface configuration information**

Please fill in the interface configuration information. This information requires you to have your own server resources. The URL you fill in needs to correctly respond to the Token verification sent by WeChat. Please read the [message interface usage guide](#).

URL	<input type="text"/>
Token	<input type="text"/>

**submit**

### Запуск бота и настройка URL-адреса конечной точки

Теперь переходите к настройке серверной части бота. Но сначала, прежде чем сохранить параметры, нужно запустить бота, чтобы WeChat отправил запрос для проверки URL-адреса. Укажите конечную точку в формате `https://your_end_point/WeChat` или настройте те же параметры, которые вы указали в файле `BotController.cs`.

### Test Account Management

WeChat number: gh\_... Sign out

**Test number information**

appId	Wx...
Appsecret	1a5dc4f1ebb...

**Modify configuration information**

Please fill in the interface configuration information. This information requires you to have your own server resources. The URL you fill in needs to correctly respond to the Token verification sent by WeChat. Please read the [message interface usage guide](#).

URL	<code>http://.../WeChat</code>
Token	testToken

### Подписка для официальной учетной записи

С помощью QR-кода вы можете оформить подписку в WeChat для тестовой учетной записи.

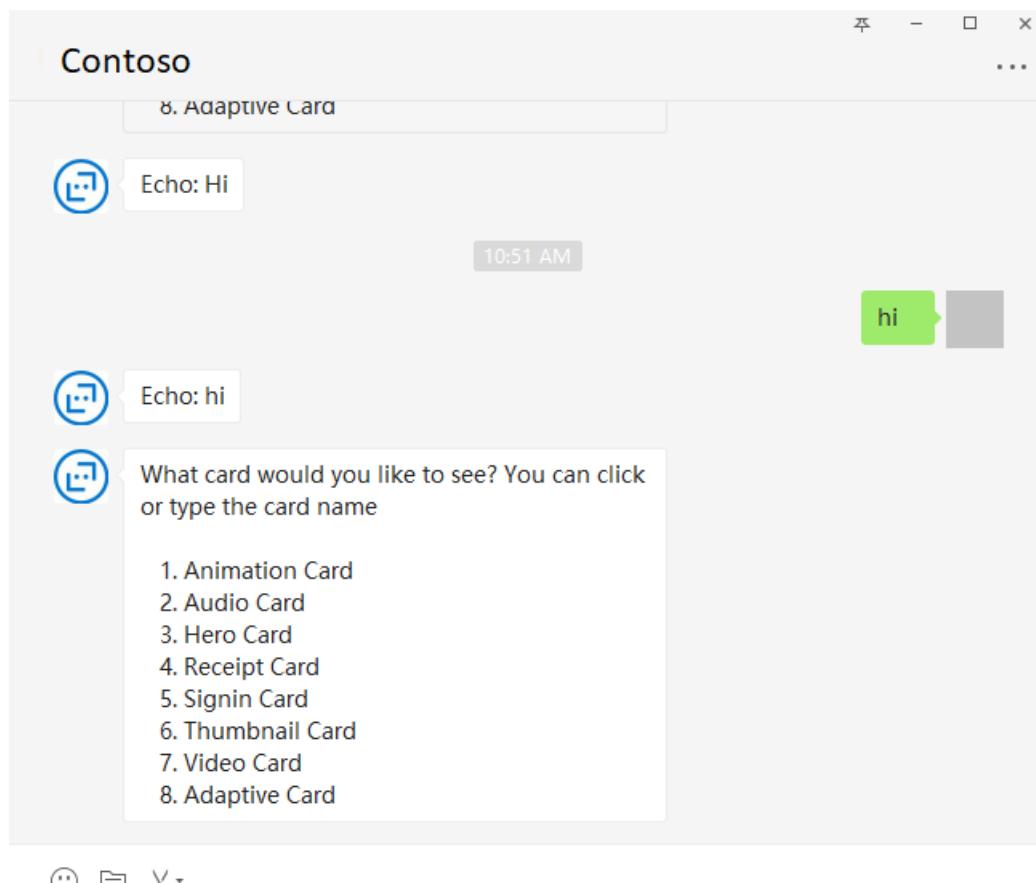
**Test account QR code**

User list (up to 100)			
Serial number	Nickname	WeChat number	Operation
No data			

Please use WeChat to scan and follow the test account

## Тестирование через WeChat

Итак, все готово и вы можете проверить работу в клиенте WeChat. Например, используйте пример бота из папки tests. Этот бот интегрирован с адаптером WeChat, а также ботами Echo и Cards.



Send (S)

# Подключение бота к веб-чату

27.03.2021 • 8 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

При создании программы-[робота](#) с помощью службы Bot канал Web Chat настраивается автоматически. Канал веб-чата включает [элемент управления "веб-чат"](#), который предоставляет пользователям возможность взаимодействовать с Bot непосредственно на веб-странице.

The screenshot shows the Azure Bot Framework portal interface. On the left, there's a sidebar with navigation links: Overview, Activity log, Access control (IAM), Tags, Bot management (with sub-links like Test in Web Chat, Analytics, Channels, Settings, Speech priming, and Bot Service pricing), and Support + troubleshooting (with New support request). The main area is titled 'PromptsTestBot' and shows the 'Bot Channels Registration' blade. It includes fields for Resource group (ChannelsPromptResourceGroup), Bot Service pricing tier (F0), Subscription (Visual Studio Enterprise with MSDN), Subscription ID (redacted), and Messaging endpoint (https://aspnetcore-channelsprompt...). Below these are sections for Resources (Documents, Samples, Feedback) and a 'Delete' button at the top right.

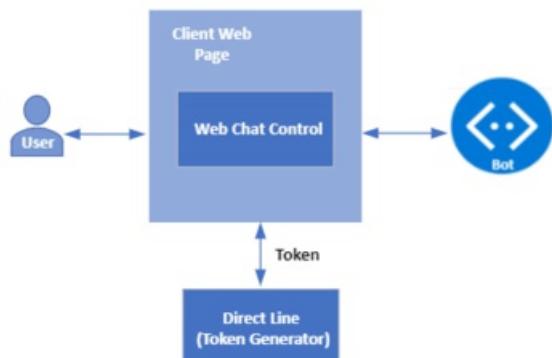
Канал веб-чата на портале Bot Framework содержит все необходимое для внедрения элемента управления "веб-чат" на веб-страницу. Для использования элемента управления "Веб-чат" нужно всего лишь получить секретный ключ бота и внедрить элемент управления на веб-странице.

## Вопросы безопасности веб-обсуждений

При использовании проверки подлинности службы Azure Bot для Web Chat важно учитывать несколько моментов, связанных с безопасностью. Обратитесь к [вопросам безопасности](#).

## Внедрение элемента управления "веб-чат" в веб-страницу

На следующем рисунке показаны компоненты, используемые при внедрении элемента управления "веб-чат" на веб-страницу.



## IMPORTANT

Как показано на предыдущем рисунке, необходимо использовать прямую линию (с расширенной проверкой подлинности) для снижения угроз безопасности при подключении к роботу с помощью элемента управления "веб-чат". Дополнительные сведения см. в разделе [Прямая строка Расширенная проверка подлинности](#).

### Получение секретного ключа бота

1. Откройте бот на [портале Azure](#) и выберите колонку **Каналы**.
2. Щелкните **Изменить** для канала **Веб-чат**.

#### Connect to channels

Name	Health	Published	
Direct Line	Running	--	Edit
Web Chat	Running	--	<b>Edit</b>

[Get bot embed codes](#)

3. В разделе **Secret keys** (Секретные ключи) щелкните **Show** (Показать) для первого ключа.

The screenshot shows the 'Configure Web Chat' page. At the top, there are icons for adding a new site, switching sites, and disabling the feature. Below this, there's a 'Default Site' dropdown set to 'Default Site'. The 'Secret keys' section contains two fields, both of which have their 'Show' buttons highlighted with red boxes. The first field's value is 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'. Below each field is a 'Regenerate' button. Underneath the secret keys, there's an 'Embed code' section with an 'iframe' example. At the bottom, there's a 'Copy' button, a link to 'Learn more about advanced customization options for Web Chat', and a 'Preview' section with a note about preview features. A 'Done' button is at the very bottom.

4. Скопируйте **секретный ключ и код внедрения**.

5. Нажмите кнопку **Done**(Готово).

### Параметры внедрения для разработки

#### Вариант 1. Обмен секретом для маркера и создание внедрения

Используйте этот вариант, если можно выполнить запрос между серверами для обмена секрета веб-чата на временный токен, а также если вы хотите усложнить внедрение бота на веб-сайтах для других разработчиков. Этот вариант не помешает другим разработчикам внедрить ваш бот на своих веб-сайтах, но усложнит эту задачу.

Для обмена секрета на токен и создания внедрения сделайте следующее:

1. Выдайте запрос GET к <https://webchat.botframework.com/api/tokens> И передайте секрет веб-чата с помощью заголовка `Authorization`. Заголовок `Authorization` использует схему `BotConnector` И включает в себя ваш секрет, как показано в следующем примере запроса.

- Ответ на ваш запрос GET будет содержать токен (в кавычках), который можно использовать, чтобы начать общение путем рендеринга элемента управления "Веб-чат" в пределах iframe. Токен действителен только для одного диалога. Чтобы запустить другой диалог, необходимо создать токен.
- В **коде внедрения** `iframe`, скопированном в канале Web Chat на портале Bot Framework (см. сведения о [получении секретного ключа бота](#) выше), измените параметр `s=` на `t=` и замените `YOUR_SECRET_HERE` своим токеном.

#### NOTE

Токены будут автоматически продлены до истечения срока действия.

#### Пример запроса

```
requestGET https://webchat.botframework.com/api/tokens
Authorization: BotConnector YOUR_SECRET_HERE
```

#### NOTE

Обратите внимание, что для Azure для государственных организаций URL-адрес обмена маркерами отличается.

```
requestGET https://webchat.botframework.azure.us/api/tokens
Authorization: BotConnector YOUR_SECRET_HERE
```

#### Пример ответа

```
"IIBSpLnn8sA.dBB.MQBhAFMAZwBXAHoANGBQAGcAZABKAEcAMwB2ADQASABjAFMAegBuAHYANwA.bbguxy0v0gE.cccJjh-TFDs.ruXQyivVZIcgvosGaFs_4jRj1AyPnDt1wk1HMBb5Fuw"
```

#### Пример iframe (с использованием токена)

```
<iframe src="https://webchat.botframework.com/embed/YOUR_BOT_ID?t=YOUR_TOKEN_HERE"></iframe>
```

#### NOTE

Обратите внимание, что для Azure для государственных организаций пример IFRAME выглядит иначе.

```
<iframe src="https://webchat.botframework.azure.us/embed/YOUR_BOT_ID?t=YOUR_TOKEN_HERE"></iframe>
```

#### Пример HTML-кода

```

<!DOCTYPE html>
<html>
<body>
    <iframe id="chat" style="width: 400px; height: 400px;" src='''></iframe>
</body>
<script>

    var xhr = new XMLHttpRequest();
    xhr.open('GET', "https://webchat.botframework.com/api/tokens", true);
    xhr.setRequestHeader('Authorization', 'BotConnector ' + 'YOUR SECRET HERE');
    xhr.send();
    xhr.onreadystatechange = processRequest;

    function processRequest(e) {
        if (xhr.readyState == 4 && xhr.status == 200) {
            var response = JSON.parse(xhr.responseText);
            document.getElementById("chat").src="https://webchat.botframework.com/embed/<botname>?t="+response
        }
    }

</script>
</html>

```

#### **Вариант 2. Внедрение элемента управления "Веб-чат" на веб-сайте с помощью секрета**

Используйте этот вариант, если вы хотите разрешить другим разработчикам с легкостью внедрять бот на их веб-сайтах.

#### **WARNING**

При использовании этого параметра секретный ключ канала веб-чата отображается на веб-странице клиента.  
Используйте этот параметр только в целях разработки, а не в рабочей среде.

Чтобы внедрить робот в веб-страницу, указав секрет в `iframe` теге, выполните действия, описанные ниже.

1. Скопируйте **код внедрения** `iframe` в канале Web Chat на портале Bot Framework (см. сведения о [получении секретного ключа бота](#) выше).
2. В этом **коде внедрения** замените YOUR\_SECRET\_HERE значением **секретного ключа**, скопированным с той же страницы.

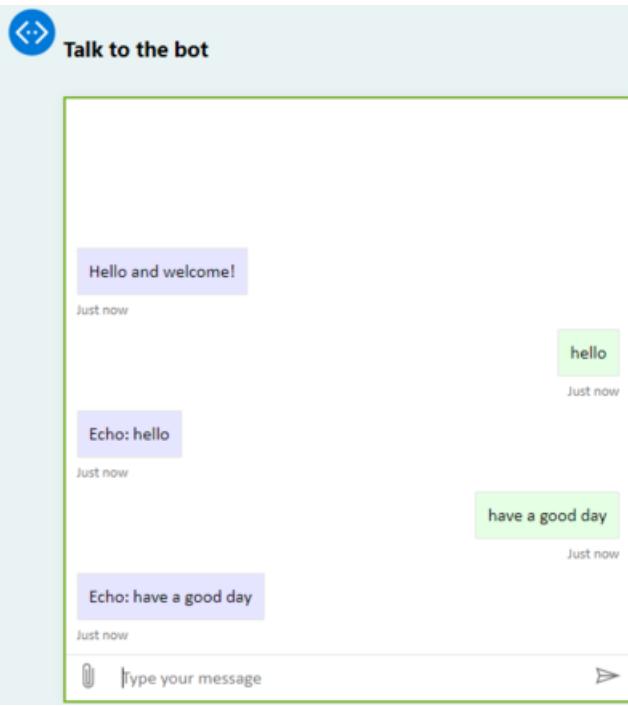
Пример `iframe` (с использованием секрета)

```
<iframe style="height:480px; width:402px" src="https://webchat.botframework.com/embed/YOUR_BOT_ID?
s=YOUR_SECRET_HERE"></iframe>
```

#### **NOTE**

Обратите внимание, что для Azure для государственных организаций пример IFRAAME выглядит иначе.

```
<iframe style="height:480px; width:402px" src="https://webchat.botframework.azure.us/embed/YOUR_BOT_ID?
s=YOUR_SECRET_HERE"></iframe>
```



## Параметр внедрения в производство

**Не отключайте секрет, отключайте свой секрет для маркера и создайте внедрение**

Этот параметр не предоставляет секретный ключ канала Web Chat на веб-странице клиента, так как он необходим в рабочей среде.

Клиент должен предоставить маркер для взаимодействия с Bot. Чтобы узнать о различиях между секретами и маркерами и узнать о рисках, связанных с использованием секретов, перейдите на страницу [Прямая проверка подлинности](#).

На следующей веб-странице клиента показано, как использовать маркер с веб-чатом. Если вы используете Azure gov, измените URL-адреса с общедоступного на государственные.

```

<!DOCTYPE html>
<html>
  <head>
    <script src="https://cdn.botframework.com/botframework-webchat/latest/webchat.js"></script>
  </head>
  <body>
    <h2>Web Chat bot client using Direct Line</h2>

    <div id="webchat" role="main"></div>

    <script>

      // "styleSet" is a set of CSS rules which are generated from "styleOptions"
      const styleSet = window.WebChat.createStyleSet({
        bubbleBackground: 'rgba(0, 0, 255, .1)',
        bubbleFromUserBackground: 'rgba(0, 255, 0, .1)',

        botAvatarImage: '<your bot avatar URL>',
        botAvatarInitials: 'BF',
        userAvatarImage: '<your user avatar URL>',
        userAvatarInitials: 'WC',
        rootHeight: '100%',
        rootWidth: '30%'
      });

      // After generated, you can modify the CSS rules
      styleSet.textContent = {
        ...styleSet.textContent,
        fontFamily: "'Comic Sans MS', 'Arial', sans-serif",
        fontWeight: 'bold'
      };

      const res = await fetch('https:<YOUR_TOKEN_SERVER/API>', { method: 'POST' });
      const { token } = await res.json();

      window.WebChat.renderWebChat(
        {
          directLine: window.WebChat.createDirectLine({ token }),
          userID: 'WebChat_UserId',
          locale: 'en-US',
          username: 'Web Chat User',
          locale: 'en-US',
          // Passing 'styleSet' when rendering Web Chat
          styleSet
        },
        document.getElementById('webchat')
      );
    </script>
  </body>
</html>

```

Примеры создания маркера см. в следующих статьях:

- [Демонстрация единого входа](#)
- [Прямой маркер линии](#)

## Дополнительные ресурсы

- [Общие сведения о веб-чате](#)
- [Настройка веб-чата](#)
- [Включение распознавания речи в веб-чате](#)
- [Применение Web Chat с расширением Службы приложений Direct Line](#)
- [Подключение бота к каналу Direct Line Speech](#)

- Добавление функции единого входа в Web Chat

# Подключение бота к Webex Teams с помощью адаптера Slack

27.03.2021 • 10 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Из этой статьи вы узнаете, как подключить бота к Webex с помощью адаптера, предоставляемого в пакете SDK. В этой статье описано, как изменить пример EchoBot для его подключения к приложению Webex.

## NOTE

Ниже приводятся инструкции для реализации адаптера Webex на C#. Инструкции по использованию реализации на JavaScript, которая входит в состав библиотек BotKit, см. в [документации по BotKit для Webex](#).

## Предварительные требования

- [Пример кода EchoBot](#).
- Доступ к команде Webex с достаточными разрешениями для создания приложений и управления ими в <https://developer.webex.com/my-apps>. Если у вас нет доступа к команде Webex, вы можете бесплатно создать учетную запись на сайте [www.webex.com](http://www.webex.com).

## Создание приложения для бота Webex

1. Войдите на [панель мониторинга разработчика Webex](#) и нажмите кнопку Create a new app (Создать приложение).
2. На следующем экране выберите создание бота Webex, щелкнув Create a bot (Создать бот).
3. На следующем экране введите имя бота, имя пользователя и описание бота, а также выберите значок или передайте собственное изображение.

## New Bot

### Bot Name\*

Name of your bot in 100 characters or less.

### Bot Username\*

The username users will use to add your bot to a space. Cannot be changed later.

@webex.bot

myadapterbot@webex.bot is available

### Icon\*

Upload your own or select from our defaults. Must be 512x512px in JPEG or PNG format.

[Edit](#)

### Description\*

Provide some details about what your bot does, how it benefits users, and how an end user can get started in 1000 characters or less. Bullets and links Markdown supported. If your app is listed on the Webex App Hub, this field will be used as the listing's description.



This is a bot that will be connected to the Microsoft Bot Framework using the [Webex Team adapter](#).

[Supported markdown](#)

By creating this app, you accept the [Terms of Service](#) and [Privacy Statement](#).

[Cancel](#)[Add Bot](#)

Нажмите кнопку Add bot (Добавить бота).

- На следующей странице вы получите маркер доступа для нового приложения Webex. Запишите его значение, так как оно потребуется при настройке бота.

**Bot's Access Token**

Non-expiring (good for 100 years) access token for your bot.

`NTBIY2I0ZjltNzRjMi00OTIxLTljZDQtNmEzNGQxMDg5C`[Copy](#)**Bot ID**

Unique system generated ID for your bot.

`Y2IzY29zcGFyazovL3VzL0FQUExJQ0FUSU9OL2M3M`[Copy](#)**Bot Name\***

Name of your bot in 100 characters or less.

`My Adapter Bot`[Edit](#)**Bot Username\***

The username users will use to add your bot to a space. Cannot be changed later.

`myadapterbot@webex.bot`

## Подключение адаптера Webex к боту

Прежде чем выполнять настройку приложения Webex, необходимо подключить адаптер Webex к боту.

### Установка пакета NuGet для адаптера Webex

Добавьте пакет NuGet [Microsoft.Bot.Builder.Adapters.Webex](#). Подробные сведения об использовании NuGet см. в руководстве по [установке пакетов и управлении ими в Visual Studio](#).

### Создание класса адаптера Webex

Создайте новый класс, наследующий от класса `*вебексадаптер_`. Этот класс будет использоваться в качестве адаптера для канала Webex. Он включает возможности обработки ошибок (подобно классу `*ботфрамворкадаптервисеррорхандлер*`, уже указанному в примере, который используется для обработки запросов от службы Azure Bot).

```

public class WebexAdapterWithErrorHandler : WebexAdapter
{
    public WebexAdapterWithErrorHandler(IConfiguration configuration, ILogger<WebexAdapter> logger)
        : base(configuration, null, logger)
    {
        OnTurnError = async (turnContext, exception) =>
        {
            // Log any leaked exception from the application.
            logger.LogError(exception, $"[OnTurnError] unhandled error : {exception.Message}");

            // Send a message to the user
            await turnContext.SendActivityAsync("The bot encountered an error or bug.");
            await turnContext.SendActivityAsync("To continue to run this bot, please fix the bot source
code.");

            // Send a trace activity, which will be displayed in the Bot Framework Emulator
            await turnContext.TraceActivityAsync("OnTurnError Trace", exception.Message,
"https://www.botframework.com/schemas/error", "TurnError");
        };
    }
}

```

### **Создание контроллера для обработки запросов Webex**

Мы создадим контроллер, который будет выполнять запросы из приложения Webex, в новой конечной точке api/webex вместо api/messages, которая используется по умолчанию для запросов от каналов службы Azure Bot. Добавив к боту дополнительную конечную точку, вы сможете с помощью одного бота принимать запросы одновременно от каналов (или других адаптеров) службы Bot и Webex.

```

[Route("api/webex")]
[ApiController]
public class WebexController : ControllerBase
{
    private readonly WebexAdapter _adapter;
    private readonly IBot _bot;

    public WebexController(WebexAdapter adapter, IBot bot)
    {
        _adapter = adapter;
        _bot = bot;
    }

    [HttpPost]
    public async Task PostAsync()
    {
        // Delegate the processing of the HTTP POST to the adapter.
        // The adapter will invoke the bot.
        await _adapter.ProcessAsync(Request, Response, _bot, default(CancellationToken));
    }
}

```

### **Включение адаптера Webex в файл Startup.cs бота**

Добавьте следующую строку в метод *ConfigureServices* в файле startup.cs, который будет регистрировать адаптер WebEx и сделать его доступным для нового класса контроллера. Описанные в следующем шаге параметры конфигурации применяются адаптером автоматически.

```
services.AddSingleton<WebexAdapter, WebexAdapterWithErrorHandler>();
```

После добавления метод *ConfigureServices* должен выглядеть следующим образом.

```

// This method gets called by the runtime. Use this method to add services to the container.
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers().AddNewtonsoftJson();

    // Create the default Bot Framework Adapter.
    services.AddSingleton<IBotFrameworkHttpAdapter, BotFrameworkAdapterWithErrorHandler>();

    // Create the default Bot Framework Adapter.
    services.AddSingleton<WebexAdapter, WebexAdapterWithErrorHandler>();

    // Create the bot as a transient. In this case the ASP Controller is expecting an IBot.
    services.AddTransient<IBot, EchoBot>();
}

```

## Добавление параметров адаптера Webex в файл конфигурации бота

- Добавьте четыре указанных ниже параметра в файл appSettings.json в проекте бота.

```

{
    "WebexPublicAddress": "",
    "WebexAccessToken": "",
    "WebexSecret": "",
    "WebexWebhookName": ""
}

```

- Заполните параметр `_вебексакцесстокен*` в маркере доступа WebEx Bot, который был создан при создании приложения WebEx Bot на предыдущих шагах. Остальные три параметра пока оставьте пустыми. Информацию для них мы соберем на следующих шагах.

## Завершение настройки приложения Webex и бота

### Создание и обновление веб-перехватчика Webex

Теперь, когда вы создали приложение Webex и подключили адаптер в проекте бота, осталось лишь настроить веб-перехватчик Webex и направить его на конечную точку бота, а затем оформить для приложения подписку, чтобы бот получал нужные сообщения и вложения. Для этого процесса бот должен работать, чтобы решение Webex успешно проверило допустимость URL-адреса конечной точки.

- Чтобы выполнить этот шаг, [разверните бота в Azure](#) и запишите URL-адрес этого развертывания. Конечной точкой Webex для обмена сообщениями является URL-адрес бота, который совпадает с URL-адресом развернутого приложения (или конечной точки ngrok) с добавленным префиксом /api/webex (например, <https://yourbotapp.azurewebsites.net/api/webex>).

#### NOTE

Если вы не готовы к развертыванию Bot в Azure или хотите отладить программу Bot при использовании адаптера WebEx, вы можете использовать такое средство, как [ngrok](#) (которое уже было установлено, если ранее использовался эмулятор Bot Framework) для туннелирования к запущенному на локальном компьютере интерфейсу Bot и предоставит вам общедоступный URL-адрес для этого.

Если вы хотите создать туннель и получить для бота URL-адрес с помощью ngrok, выполните следующую команду в окне терминала. Здесь предполагается, что локальный бот работает на порту 3978. Если это не так, измените номера портов в команде.

```
ngrok.exe http 3978 -host-header="localhost:3978"
```

- Перейдите по адресу <https://developer.webex.com/docs/api/v1/webhooks>.
- Щелкните ссылку для метода POST <https://webexapis.com/v1/webhooks> (с описанием *создать веб-перехватчик*). Будет отображена форма, позволяющая отправить запрос в конечную точку.

Method	Description
<code>GET</code> <a href="https://webexapis.com/v1/webhooks">https://webexapis.com/v1/webhooks</a>	List Webhooks
<code>POST</code> <a href="https://webexapis.com/v1/webhooks">https://webexapis.com/v1/webhooks</a>	Create a Webhook
<code>GET</code> <a href="https://webexapis.com/v1/webhooks/{webhookId}">https://webexapis.com/v1/webhooks/{webhookId}</a>	Get Webhook Details
<code>PUT</code> <a href="https://webexapis.com/v1/webhooks/{webhookId}">https://webexapis.com/v1/webhooks/{webhookId}</a>	Update a Webhook
<code>DELETE</code> <a href="https://webexapis.com/v1/webhooks/{webhookId}">https://webexapis.com/v1/webhooks/{webhookId}</a>	Delete a Webhook

- Заполните форму следующими сведениями:

- Имя** — имя веб-перехватчика, например сообщения веб- *перехватчик*.
- TargetUrl** — полный URL-адрес конечной точки WebEx для ленты, например <https://yourbotapp.azurewebsites.net/api/webex> ).
- Resource** -messages.
- созданное **событие** .
- Фильтр** . Оставьте поле пустым.
- секрет** — секретный ключ для защиты веб-перехватчика. Позже вы добавите его в программу-робот [appsettings.json](#) .

Header

Content-Type: application/json

Authorization: Bearer [REDACTED]

This limited-duration personal access token is hidden for your security.

Body

name Required	e.g. My Awesome Webhook
targetUrl Required	e.g. https://example.com/mywebhook
resource Required	e.g. messages
event Required	e.g. created
filter	e.g. roomId=Y2lzY29zcGFyazovL3VzL1JPT
secret	e.g. 86dacc007724d8ea666f88fc77d918a

Run

- Нажмите кнопку Run (Выполнить), чтобы создать веб-перехватчик и получить сообщение об успешном выполнении.

#### Заполнение оставшихся параметров приложения бота

Заполните оставшиеся три параметра в файле appSettings.json для бота (значение WebexAccessToken вы уже указали на предыдущем шаге).

- WebexPublicAddress (это полный URL-адрес конечной точки бота для Webex).

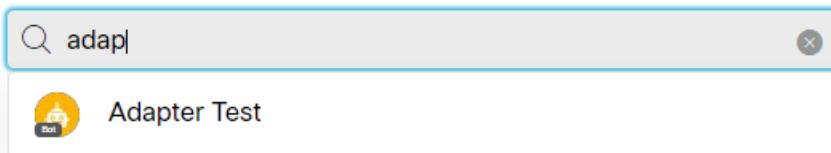
- WebexSecret (секрет, указанный при создании веб-перехватчика на предыдущем шаге).
- WebexWebhookName (имя веб-перехватчика, которое вы указали на предыдущем шаге).

## Повторное развертывание бота в команде Webex

Теперь, когда вы завершили настройку параметров бота в файле appSettings.json, следует повторно развернуть бота или просто перезапустить его, если вы используете туннель ngrok к локальной конечной точке. Теперь настройка приложения Webex и бота завершена. Вы можете войти в команду Webex по адресу <https://www.webex.com> и начать чат с ботом, отправляя ему сообщения точно так же, как другому человеку.

### Contact a person

Start a conversation between you and just one other person.



# Дополнительные каналы

27.03.2021 • 2 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Кроме каналов, перечисленных в этих документах, есть и дополнительные каналы, доступные как адаптеры. Они предоставляются через наши [платформы](#) (Botkit) или [репозитории сообщества](#).

Представляются следующие дополнительные каналы:

- [Webex Teams](#);
- [WebSocket](#) и [Webhooks](#);
- [Google Hangouts](#) и [Google Assistant](#) (доступно через сообщество);
- [Amazon Alexa](#) (доступно через сообщество).

## Доступные сейчас адаптеры

Полный список доступных адаптеров см. в разделе [адаптеры платформы](#). Вы заметите, что некоторые каналы доступны как адаптеры. Именно вы решаете, когда использовать канал, а когда — адаптер.

### Когда следует использовать адаптер

1. Служба не поддерживает нужный канал.
2. Требования к безопасности и соответствуя требованиям в развертывании определяют, что нельзя полагаться на внешнюю службу.
3. Глубина функций, необходимая для конкретного канала, может не поддерживаться.

### Когда следует использовать канал

1. Необходима Межканальная совместимость. Программа Bot должна работать на нескольких доступных каналах.
2. Встроенная поддержка: Корпорация Майкрософт сохраняет, обновляет и без труда обслуживает каждый канал каждый раз, когда сторонние программы делают обновления.
3. Предоставляет доступ к дополнительным эксклюзивным каналам Майкрософт, таким как быстрый рост Microsoft Teams.
4. Если вы хотите использовать интерфейс графического интерфейса пользователя, чтобы включить дополнительные каналы для программы-робота.

# Общие сведения о переносе

27.03.2021 • 21 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Пакет SDK для Bot Framework версии 4 основан на отзывах клиентов и учебных интерфейсах предыдущих пакетов SDK. В нем представлены верные уровни абстракции при использовании гибкой архитектуры компонентов Bot. Помимо прочего, это позволяет создать простой робот, а затем расширить его сложность, используя модульность и расширяемость пакета SDK для Bot версии 4.

## NOTE

При создании пакета SDK Bot Framework версии 4 разработчики стремились обеспечить возможность реализации комплексных задач, не усложняя используемые подходы.

Открытый подход к разработке пакета SDK Bot Framework версии 4 позволил принять в ней активное участие членам сообщества. При первой подаче запроса на вытягивание происходит автоматическая оценка того, нужно ли вам подписать [лицензионное соглашение с участником](#) (CLA). Это однократная процедура, позволяющая работать с разными репозиториями. Как правило, устанавливается определенный срок, чтобы обозначить задачи, которые планируется выполнить.

## Что будет с ботами, созданными с помощью пакета SDK версии 3

Пакет SDK Bot Framework версии 3 станет недоступным для работы, но существующие рабочие нагрузки ботов версии 3 продолжат работу. Дополнительные сведения см. в разделе: [Поддержка пакета SDK Bot Framework версии 3 на все время существования](#).

Настоятельно рекомендуем начать перенос ботов с версии 3 в версию 4. Чтобы упростить перенос, мы подготовили соответствующую документацию и окажем расширенную поддержку инициативам по переносу по стандартным каналам.

Если вы не можете выполнить миграцию бота из версии 3 в версию 4, вы все равно можете использовать дополнительные функции пакета SDK версии 4. Вы можете преобразовать бот версии 3 в навык и создать бот на основе пакета SDK версии 4, который будет выполнять роль потребителя навыков и передавать сообщения в бот версии 3. См. сведения о [преобразовании бота версии 3 в навык](#).

## Преимущества

- Более разнообразная, гибкая и открытая архитектура, позволяющая создавать сложные структуры диалога.
- "Доступность": добавлены сценарии с поддержкой новых каналов.
- Увеличен штат профильных специалистов, занятых в разработке. Новый конструктор с графическим пользовательским интерфейсом позволяет пользователям, которые не связаны с разработкой, участвовать в работе над структурами диалога.
- Повышение темпов разработки. Доступны новые средства разработчика для отладки и тестирования.
- Анализ производительности. Добавлены новые функции телеметрии для оценки и повышения качества диалога.
- Аналитика. Улучшены возможности Cognitive Services.

## Аргументы в пользу миграции

- Гибкое и оптимизированное управление диалогом.
  - Доступен адаптер бота для обработки действий.
  - Выполнен рефакторинг управления состоянием.
  - Добавлена новая библиотека диалогов.
  - Доступно ПО промежуточного слоя для составных и расширяемых структур. Доступны понятные и согласованные обработчики для настройки поведения.
- Поддержка .NET Core 2.
  - повышение производительности.
  - Кроссплатформенная поддержка (Windows, Mac, Linux).
- Единообразная программная модель для разных языков программирования.
- Улучшена документация.
- Bot Inspector обеспечивает расширенные возможности отладки.
- Виртуальный помощник
  - Комплексное решение, которое упрощает создание ботов, с такими компонентами и функциями, как основные намерения для реализации общения, интеграция со средством Dispatch, QnA Maker, Application Insights и автоматизированное развертывание.
  - Наращиваемые навыки. Возможность создавать решения для общения, комбинируя доступные для повторного использования возможности ведения беседы, называемые навыками.
- Платформа тестирования. Не требующие настройки возможности тестирования с новой архитектурой независимого транспортного адаптера.
- Данные телеметрии. Получите важные сведения о работоспособности и поведении Bot с помощью анализа искусственного интеллекта
- Ожидается (в предварительной версии).
  - Адаптивные диалоги. Позволяют разработчикам создавать диалоги, которые могут изменяться динамически в ходе беседы.
  - Создание языка. Позволяет определять несколько вариантов фразы.
- Планируется реализация.
  - Декларативный подход к разработке, задающий уровень абстракции для разработчиков.
  - Конструктор диалогов с графическим пользовательским интерфейсом.
- Служба Azure Bot
  - Канал Direct Line Speech. Объединяет Bot Framework и службу "Речь" корпорации Майкрософт. Этот канал обеспечивает двустороннюю потоковую передачу речи и текста между клиентом и приложением бота.

## Изменения

Пакет SDK Bot Framework версии 4 поддерживает ту же базовую службу Bot Framework, что и версия 3. Но в версии 4 выполнен рефакторинг кода предыдущей версии пакета SDK для повышения гибкости и уровня контроля над процессом создания ботов. Это включает следующие действия.

- Добавлен адаптер бота.
  - Адаптер — это компонент стека обработки действий.
  - Он выполняет аутентификацию Bot Framework и инициализирует контекст каждого шага.
  - Адаптер управляет входящим и исходящим трафиком между каналом и обработчиком шагов вашего бота, инкапсулируя вызовы службы Bot Framework Connector.
  - Дополнительные сведения см. в статье [Принципы работы бота](#).
- Выполнен рефакторинг управления состоянием.

- Данные о состоянии больше не предоставляются в боте автоматически.
- Управление состоянием теперь осуществляется через соответствующие объекты и методы доступа к свойствам.
- Дополнительные сведения см. в статье [Управление состоянием](#).
- Добавлена новая библиотека диалогов.
  - Диалоги версии 3 необходимо переписать для новой библиотеки диалогов.
  - Дополнительные сведения см. в статье [Библиотека диалогов](#).

## Что необходимо выполнить при миграции

- Обновить логику настройки.
- Перенести критически важные данные о состоянии пользователя.
  - Примечание. Важные данные о состоянии пользователя не следует хранить в службе "Состояние бота". Используйте для этого контролируемое вами отдельное хранилище.
- Перенести бота и логику диалога (дополнительные сведения см. в посвященных языку разделах).

### Список задач в рамках миграции

Следующие списки задач помогут оценить объем рабочей нагрузки для миграции. В столбце **Количество** замените значение своими фактическими числовыми значениями. В столбце **Размер** укажите такие значения, как *Небольшой*, *Средний*, *Большой*, руководствуясь своей оценкой.

- [C#](#)
- [JavaScript](#)

ШАГ	V3	ВЕРСИЯ 4	ВХОЖДЕНИЯ	СЛОЖНОСТЬ	РАЗМЕР
Получение входящего действия	IDialogContext.Activity	ITurnContext.Activity	count	Малый	
Создание действия и отправка его пользователю	оборт. Креатерепли ("Text") Идиалогконтекст. onasync	Мессажефактори. Text ("Text") Итурнконтекст. Сендактивитасинк	count	Малый	
Управление данными о состоянии	UserData, ConversationData и PrivateConversationData context.UserData.SetValue context.UserData.TryGetValue botDataStore.LoadAsync	UserState, ConversationState и PrivateConversationState с методами доступа к свойствам	context.UserData.SetValue "значение" context.UserData.TryGetValue "значение" botDataStore.LoadAsync "значение"	Средняя– высокая (см. раздел <a href="#">Управление данными о состоянии</a> )	
Обработка запуска диалога	Реализация IDialog.StartAsync	Сделайте этот шаг первым в каскадном диалоге.	count	Малый	
Отправка действия	IDialogContext.PostAsync.	Вызов ITurnContext.SendActivityAsync.	count	Малый	

ШАГ	V3	ВЕРСИЯ 4	ВХОЖДЕНИЯ	СЛОЖНОСТЬ	РАЗМЕР
Ожидание ответа пользователя	Вызов IDialogContext.Wait с помощью параметра IAwaitable	Возврат await ITurnContext.PromptAsync для начала диалога запроса. Затем получите результаты на следующем шаге каскадного диалога.	count	Средняя (в зависимости от потока)	
Обработка продолжения диалога	IDialogContext.Wait	Добавление дополнительных шагов в каскадный диалог или реализация Dialog.ContinueDialogAsync	count	большой	
Обозначение завершения обработки до следующего сообщения пользователя	IDialogContext.Wait	Возврат Dialog.EndOfTurn.	count	Средний	
Запуск дочернего диалога	IDialogContext.Call	Возврат await BeginDialogAsync method контекста шага. Если дочерний диалог вернет значение, оно будет доступно на следующем шаге каскадного диалога в свойстве Resultproperty.	count	Средний	
Замена текущего диалога новым	IDialogContext.Forward	Возврат await ITurnContext.ReplaceDialogAsync.	count	большой	
Информирование о завершении текущего диалога	IDialogContext.Done	Возврат await метода EndDialogAsync контекста шага.	count	Средний	

ШАГ	V3	ВЕРСИЯ 4	ВХОЖДЕНИЯ	СЛОЖНОСТЬ	РАЗМЕР
Выход из диалога	IDialogContext.Fail	Создание исключения, которое будет зарегистрировано на другом уровне бота, и завершение шага с состоянием Cancelled, либо вызов шага или CancelAllDialogsAsync контекста диалога.	count	Малый	

- [C#](#)
- [JavaScript](#)

Пакет SDK Bot Framework версии 4 использует тот же базовый REST API, что и пакет SDK версии 3. Но в версии 4 выполнен рефакторинг кода предыдущей версии пакета SDK для повышения гибкости и улучшения контроля над ботами.

Мы рекомендуем перейти на .NET Core, что позволяет существенно повысить производительность. Но некоторые боты версии 3 используют внешние библиотеки, у которых нет аналогов на .NET Core. В этом случае можно использовать пакет SDK для Bot Framework версии 4 с платформой .NET Framework версии 4.6.1 или более поздней. Пример доступен в [этом репозитории](#).

При переносе проекта из версии 3 в версию 4 можно выбрать один из следующих вариантов: преобразование в .NET Framework или перенос нового проекта на .NET Core.

#### .NET Framework

- Обновление и установка пакетов NuGet
- Обновление файла global.asax.cs
- Обновление класса MessagesController.
- Преобразование диалогов.

Дополнительные сведения см. статью [Перенос бота .NET версии 3 в бот .NET Framework версии 4](#).

#### .NET Core

- Создание проекта на основе шаблона

#### NOTE

Пакет [VSIX](#) включает версии .net Core 2.1 и .net Core 3.1 шаблонов C#. При создании ботов в Visual Studio 2019 следует использовать шаблоны .NET Core 3.1. В текущих примерах ботов используются шаблоны .NET Core 3.1. Примеры, использующие шаблоны .NET Core 2.1, можно найти в ветви [4.7-archive](#) репозитория BotBuilder-Samples. Сведения о развертывании .NET Core 3.1 программы-роботы в Azure см. в статье [развертывание программы Bot в Azure](#).

- Установите дополнительные пакеты NuGet при необходимости.
- Настройте бота согласно требованиям, обновите файла Startup.cs и класс контроллера.
- Обновление класса бота
- Скопируйте и обновите диалоги и модели.

Дополнительные сведения см. статье [Перенос бота .NET версии 3 в бот .NET Framework версии 4](#).

## Дополнительные ресурсы

Перечисленные ниже ресурсы содержат дополнительные сведения, которые помогут с миграцией.

- [C#](#)
- [JavaScript](#)

В следующих разделах описаны различия между пакетами SDK Bot Framework для .NET версий 3 и 4, критические изменения в обеих версиях, а также приведены пошаговые инструкции по переходу с версии 3 на версию 4.

РАЗДЕЛ	ОПИСАНИЕ
<a href="#">Различия между версиями 3 и 4 пакета SDK для .NET</a>	Различия между версиями 3 и 4 пакета SDK
<a href="#">Краткий справочник по миграции для .NET</a>	Критические изменения в версиях 3 и 4 пакета SDK
<a href="#">Перенос бота .NET версии 3 в .NET Framework версии 4</a>	Перенос бота из версии 3 в версию 4 с использованием одного и того же типа проекта
<a href="#">Перенос бота .NET версии 3 в бот .NET Core версии 4</a>	Перенос бота из версии 3 в версию 4 с помощью нового проекта .NET Core

## Примеры кода

Ниже приведены примеры кода, которые позволяют ознакомиться с пакетом SDK Bot Framework версии 4 или приступить к новому проекту.

ПРИМЕРЫ	ОПИСАНИЕ
<a href="#">Примеры переноса бота из пакета SDK Bot Framework версии 3 в версию 4</a>	Примеры миграции из пакета SDK Bot Framework версии 3 в версию 4
<a href="#">Примеры Bot Builder для .NET</a>	Примеры Bot Builder для C# и .NET Core
<a href="#">Примеры Bot Builder для JavaScript</a>	Примеры Bot Builder для JavaScript (Node.js)
<a href="#">Все примеры Bot Builder</a>	Все примеры Bot Builder

## Получение справки

Указанные ниже ресурсы содержат дополнительные сведения и могут помочь с разработкой ботов.

### [Дополнительные ресурсы по Bot Framework](#)

## Ссылки

Указанные ниже ресурсы содержат дополнительные и справочные сведения.

РАЗДЕЛ	ОПИСАНИЕ
<a href="#">Новые возможности Bot Framework</a>	Основные возможности и усовершенствования Bot Framework и Azure Bot
<a href="#">Принципы работы бота</a>	Внутренний механизм бота

<b>РАЗДЕЛ</b>	<b>ОПИСАНИЕ</b>
Управление состоянием	Абстракции, упрощающие управление состоянием
Библиотека диалогов	Основные концепции управления диалогом
Отправка и получение текстовых сообщений	Основной способ взаимодействия бота с пользователями
Отправка мультимедиа	Отправка во вложении мультимедийных материалов, таких как изображения, файлы, видео и аудиозаписи
Реализация процесса общения	Постановка вопросов как основной способ взаимодействия бота с пользователями
Сохранение данных пользователя и диалога	Отслеживание диалога без отслеживания состояния
Сложный поток беседы	Управление сложным процессом общения
Повторное использование диалогов	Создание независимых диалогов для особых сценариев
Обработка прерываний	Обработка прерываний для повышения эффективности бота
Схема действий	Схема действий, предпринимаемых в ходе беседы людьми и автоматическим программным обеспечением

# Различия между версиями 3 и 4 пакета SDK для .NET

27.10.2020 • 18 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Версия 4 пакета SDK Bot Framework поддерживает ту же базовую службу Bot Framework, что и версия 3. Но в версии 4 выполнен рефакторинг кода предыдущей версии пакета SDK для повышения гибкости и улучшения контроля над ботами. Основные изменения в пакете SDK:

- Общие сведения об адаптере бота. Адаптер — это компонент стека обработки действий.
  - Адаптер выполняет аутентификацию Bot Framework.
  - Адаптер управляет входящим и исходящим трафиком между каналом и обработчиком шагов вашего бота, инкапсулируя вызовы в Bot Framework Connector.
  - Адаптер инициализирует контекст для каждого шага.
  - Дополнительные сведения см. в статье [Принципы работы бота](#).
- Выполнен рефакторинг управления состоянием.
  - Данные о состоянии автоматически больше не предоставляются в боте.
  - Управление состоянием теперь осуществляется через соответствующие объекты и методы доступа к свойствам.
  - Дополнительные сведения см. в статье [Управление состоянием](#).
- Новая библиотека диалогов.
  - Диалоги версии 3 потребовалось переписать для новой библиотеки диалогов.
  - Диалоги с возможностью оценки Scoreable больше не существуют. Вы можете проверить наличие глобальных команд, прежде чем передавать средства управления в диалоги. В зависимости от архитектуры бота версии 4 это можно сделать в обработчике сообщений или в родительском диалоге. Пример см. в статье [Обработка прерываний со стороны пользователя](#).
  - Дополнительные сведения см. в статье [Библиотека диалогов](#).
- Поддержка ASP.NET Core.
  - Шаблоны создания ботов C# предназначены для платформы ASP.NET Core.
  - Вы все еще можете использовать ASP.NET для своих ботов, но в версии 4 мы хотим уделить основное внимание поддержке платформы ASP.NET Core.
  - Дополнительные сведения об этой платформе см. в статье [Введение в ASP.NET Core](#).

## Обработка действий

При создании адаптера для бота вы также указываете делегата обработчика сообщений, который будет получать входящие действия от каналов и пользователей. Адаптер создает объект контекста шага для каждого полученного действия. Он передает объект контекста шага обработчику шагов в боте, а затем удаляет объект после завершения шага.

Обработчик шагов может получать действия разных типов. Как правило, диалогам в боте необходимо перенаправлять только действия *сообщений*. Если вы наследуете бота из `ActivityHandler`, обработчик шагов в нем будет пересыпать все сообщения о действиях в `OnMessageActivityAsync`. Переопределите этот метод, чтобы добавить логику обработки сообщений. Дополнительные сведения о типах действий см. в статье о [Bot Framework — Activity](#).

### Шаги обработки

При обработке сообщений используйте контекст шага, чтобы получить данные о входящем действии и отправить действия пользователю:

ЗАДАЧА	ПРОЦЕСС
Получение входящего действия	Получите свойство <code>Activity</code> контекста шага.
Создание действия и отправка его пользователю	Вызовите метод <code>SendActivityAsync</code> контекста шага. Дополнительные сведения см. в статьях <a href="#">Отправка и получение текстовых сообщений</a> и <a href="#">Добавление мультимедиа в сообщения</a> .

Класс `MessageFactory` предоставляет некоторые вспомогательные методы для создания и форматирования действий.

#### Элементы с возможностью оценки больше не используются

Обрабатывайте такие элементы в цикле сообщений бота. Описание такого процесса для диалогов в версии 4 см. в статье [Обработка прерываний со стороны пользователя](#).

Составные деревья отправки с возможностью оценки и составные последовательные диалоги, например *исключение по умолчанию*, также больше не используются. Чтобы включить эти возможности, вы можете реализовать их в обработчике шагов бота.

## Управление данными о состоянии

В версии 3 вы могли хранить данные о беседе в службе "Состояние бота", которая входит в обширный набор служб для Bot Framework. Но после 31 марта 2018 г. эта служба более недоступна. Начиная с версии 4, применяются те же рекомендации по управлению состоянием, что и для любого веб-приложения, и его можно реализовать несколькими способами. Обычно проще всего кэшировать состояние в памяти в том же процессе, но в производственных приложениях состояние следует сохранять в более устойчивом хранилище, например в базе данных SQL или NoSQL или в больших двоичных объектах.

В версии 4 для управления состоянием не используются свойства `UserData`, `ConversationData` и `PrivateConversationData`, а также контейнеры данных. Управление состоянием теперь осуществляется через соответствующие объекты и методы доступа к свойствам, как описано в статье [Управление состоянием](#).

Версия 4 определяет классы `UserState`, `ConversationState` и `PrivateConversationState`, которые управляют данными состояния для бота. Вам нужно создать метод доступа к свойству состояния для каждого свойства, которое нужно сохранить (а не просто считывать и записывать его в предопределенный контейнер данных).

#### Настройка состояния

Если это возможно, состояние необходимо настроить с использованием одноэлементных экземпляров (в `Startup.cs` для .NET Core или в `Global.asax.cs` для .NET Framework).

- Инициализируйте один или несколько объектов `IStorage`. Вы получите резервное хранилище для данных вашего бота. Пакет SDK версии 4 предоставляет несколько [уровней хранения](#). Вы также можете реализовать собственные уровни, чтобы подключаться к хранилищам различных типов.
- Затем при необходимости создайте и зарегистрируйте объекты [управления состоянием](#). Вам доступны те же области, что и в версии 3, и при необходимости вы можете создать другие.
- Затем создайте и зарегистрируйте [методы доступа к свойствам состояния](#) для нужных боту свойств. В объекте управления состоянием каждый метод доступа к свойству должен иметь уникальное имя.

## Использование состояния

Вы можете использовать внедрение зависимостей для получения к ним доступа при каждом создании бота. (В ASP.NET новый экземпляр бота или контроллера сообщений создается для каждого шага.) Используйте методы доступа к свойствам состояния, чтобы получить и обновить свои свойства, а объекты управления состоянием, чтобы записать какие-либо изменения в хранилище. Так как вам необходимо учитывать особенности параллелизма, ниже описано, как выполнять некоторые стандартные задачи.

ЗАДАЧА	ПРОЦЕСС
Создание метода доступа к свойству состояния	Вызовите процедуру <code>BotState.CreateProperty&lt;T&gt;</code> . <code>BotState</code> является абстрактным базовым классом для беседы, закрытой беседы и состояния пользователя.
Получение текущего значения свойства	Вызовите процедуру <code>IStatePropertyAccessor&lt;T&gt;.GetAsync</code> . Если значение не задано ранее, для его создания будет использоваться фабричный параметр по умолчанию.
Обновление текущего кэшированного значения свойства	Вызовите процедуру <code>IStatePropertyAccessor&lt;T&gt;.SetAsync</code> . Эта операция обновляет только кэш, но не уровень резервного хранилища.
Сохранение изменений состояний в хранилище	Вызовите <code>BotState.SaveChangesAsync</code> для любого объекта управления состоянием, в котором состояние было изменено до выхода из обработчика шагов.

## Управление параллелизмом

Возможно, вашему боту потребуется управлять параллелизмом состояния. Дополнительные сведения см. в разделе [Сохранение состояния](#) статьи [Управление состоянием](#) и в разделе [Управление параллелизмом с помощью тегов eTag](#) статьи [Запись данных напрямую в хранилище](#).

## Библиотека диалогов

Ниже приведены некоторые из основных изменений в диалогах:

- Библиотека диалогов теперь имеет вид отдельного пакета NuGet: [Microsoft.Bot.Builder.Dialogs](#).
- Классы диалогов теперь не нужно сериализовать. Управление состоянием диалога осуществляется через метод доступа к свойству состояния `DialogState`.
  - Свойство состояния диалога теперь сохраняется между шагами в отличие от самого объекта диалога.
- Интерфейс `IDialogContext` заменен классом `DialogContext`. В ходе выполнения шага вы создаете контекст диалога для *набора диалогов*.
  - Этот контекст диалога инкапсулирует стек диалога (старый кадр стека). Эта информация сохраняется в свойстве состояния диалога.
- Интерфейс `IDialog` заменен абстрактным классом `Dialog`.

## Определение диалогов

Версия 3 предоставляла гибкую возможность реализовать диалоги через интерфейс `IDialog`, но для этого требовалось создать собственный код для таких функций, как проверка. В версии 4 появились классы запроса, которые автоматически проверяют пользовательский ввод, применяют к нему ограничение по типу (например целое число) и повторно запрашивают данные у пользователя, пока они не будут соответствовать этим ограничениям. В большинстве случаев это позволяет разработчику

писать меньше кода.

Теперь вам доступно несколько способов определения диалогов:

ТИП ДИАЛОГА	ОПИСАНИЕ
Компонентный диалог, наследуемый от класса <code>ComponentDialog</code> .	Позволяет инкапсулировать код диалога без конфликтов именования с другими контекстами. См. статью <a href="#">Повторное использование диалогов</a> .
Каскадный диалог, экземпляр класса <code>WaterfallDialog</code> .	Разработан для оптимальной работы с диалогами запроса, которые предлагают пользователю ввести данные и проверяют их. Каскадный диалог автоматизирует большинство процессов, но требует применения определенной формы для кода диалога (см. статью <a href="#">Реализация процесса общения</a> ).
Настраиваемый диалог, полученный из абстрактного класса <code>Dialog</code> .	Дает максимальную гибкость в аспекте поведения диалогов, но также требует более глубоких знаний о реализации стека диалогов.

В версии 3, вы использовали `FormFlow` для выполнения фиксированного количества шагов для задачи. В версии 4 `FormFlow` заменен каскадным диалогом. Создавая каскадный диалог, вы определяете шаги диалога в конструкторе. Порядок выполняемых действий строго соответствует тому, в котором вы их объявили, и переход между ними происходит автоматически.

Можно также создать сложные потоки управления с помощью нескольких диалогов, как описано в статье [Создание сложного потока беседы с использованием ветвления и циклов](#).

Чтобы получить доступ к диалогу, вам необходимо разместить его экземпляр в *наборе диалогов*, а затем сгенерировать *контекст диалога* для такого набора. При создании набора диалогов вам нужно указать метод доступа к свойству состояния диалога. Это позволит платформе сохранять состояние диалога при переходе между шагами. Управление состоянием в версии 4 описано в статье [Управление состоянием](#).

## Использование диалогов

Ниже приведен список стандартных операций в версии 3, а также описывается их выполнение в каскадном диалоге. Обратите внимание, что каждый шаг каскадного диалога должен возвращать значение `DialogTurnResult`. В противном случае каскадный диалог может преждевременно завершиться.

ОПЕРАЦИЯ	ВЕРСИЯ 3	ВЕРСИЯ 4
Обработка запуска диалога	Реализуйте <code>IDialog.StartAsync</code>	Сделайте этот шаг первым в каскадном диалоге.
Отправка действия	Вызовите процедуру <code>IDialogContext.PostAsync</code> .	Вызовите процедуру <code>ITurnContext.SendActivityAsync</code> . Используйте свойство <code>Context</code> контекста шага, чтобы получить контекст шага.
Ожидание ответа пользователя	Используйте параметр <code>IAwaitable&lt;IMessageActivity&gt;</code> и вызовите <code>IDialogContext.Wait</code> .	Верните ожидание <code>ITurnContext.PromptAsync</code> , чтобы начать диалог с запросом. Затем получите результаты на следующем шаге каскадного диалога.

ОПЕРАЦИЯ	ВЕРСИЯ 3	ВЕРСИЯ 4
Обработка продолжения диалога	Вызовите процедуру <code>IDialogContext.Wait</code> .	Добавьте в каскадный диалог дополнительные шаги или реализуйте <code>Dialog.ContinueDialogAsync</code>
Обозначение завершения обработки до следующего сообщения пользователя	Вызовите процедуру <code>IDialogContext.Wait</code> .	Возвращается значение <code>Dialog.EndOfTurn</code> .
Запуск дочернего диалога	Вызовите процедуру <code>IDialogContext.Call</code> .	Верните ожидание для метода <code>BeginDialogAsync</code> контекста шага. Если дочерний диалог вернет значение, оно будет доступно на следующем шаге каскадного диалога с помощью свойства <code>Result</code> контекста шага.
Замена текущего диалога новым	Вызовите процедуру <code>IDialogContext.Forward</code> .	Верните ожидание для <code>ITurnContext.ReplaceDialogAsync</code> .
Информирование о завершении текущего диалога	Вызовите процедуру <code>IDialogContext.Done</code> .	Верните ожидание для метода <code>EndDialogAsync</code> контекста шага.
Выход из диалога	Вызовите процедуру <code>IDialogContext.Fail</code> .	Сгенерируйте исключение, которое будет зарегистрировано на другом уровне бота, завершите шаг с состоянием <code>Cancelled</code> либо вызовите шаг или <code>CancelAllDialogsAsync</code> контекста диалога. Обратите внимание, что в версии 4 исключения в диалоге распространяются по стеку C#, а не по стеку диалога.

Другие примечания о коде версии 4:

- Различные производные классы `Prompt` в версии 4 реализуют запросы пользователей в виде отдельных диалогов из двух шагов. Изучите, как правильно [реализовать последовательный поток беседы](#).
- Используйте `DialogSet.CreateContextAsync`, чтобы создать контекст диалога для текущего шага.
- Используйте свойство `DialogContext.Context` из диалога, чтобы получить контекст текущего шага.
- Каскадные шаги имеют параметр `WaterfallStepContext`, который является производным от `DialogContext`.
- Все конкретные классы диалогов и приглашений наследуют от абстрактного класса `Dialog`.
- Идентификатор присваивается при создании компонентного диалога. Каждый диалог в наборе диалогов должен иметь уникальный в пределах этого набора идентификатор.

#### Передача состояния в диалогах и между ними

В разделе [состояние диалоговой](#) статьи **Библиотека диалоговых окон** и в разделах [свойства контекста каскада](#) и [параметры диалоговых](#) окон статьи **о компонентах и каскадных диалоговых окнах** описывается управление состоянием диалога в v4.

**IAwaitable** больше не используется

Чтобы получить действие пользователя на шаге, получите его из контекста шага.

Чтобы отправить запрос пользователю и получить результат:

- Добавьте соответствующий экземпляр запроса к вашему набору диалогов.
- Вызовите запрос из шага в каскадном диалоге.
- Извлеките результат из свойства `Result` контекста шага на следующем шаге.

## FormFlow

В версии 3 FormFlow входит в состав пакета SDK для C#, но не в состав пакета SDK для JavaScript. В пакете SDK версии 4 FormFlow отсутствует, но доступна версия для сообщества для C#.

ИМЯ ПАКЕТА NUGET	РЕПОЗИТОРИЙ СООБЩЕСТВА GITHUB
Bot.Builder.Community.Dialogs.Formflow	<a href="https://github.com/BotBuilderCommunity/botbuilder-community-dotnet/libraries/Bot.Builder.Community.Dialogs/FormFlow">BotBuilderCommunity/botbuilder-community-dotnet/libraries/Bot.Builder.Community.Dialogs/FormFlow</a>

## Дополнительные ресурсы

- [Перенос бота с версии 3 в версию 4 пакета SDK для .NET](#)

# Краткий справочник по миграции для .NET

27.03.2021 • 11 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В пакете SDK Bot Builder для .NET версии 4 реализовано несколько важных изменений, связанных с процессом создания ботов. В этом руководстве описаны различия между способами выполнения задач в пакетах SDK версий 3 и 4.

- Способ передачи данных между каналами и ботом изменился. В версии 3 использовался объект *беседы* и метод *SendAsync* для обработки сообщения, а для загрузки разных зависимостей широко использовался Autofac. В версии 4 используются объекты *адаптера* и *TurnContext* для обработки сообщения, а для внедрения зависимостей можно использовать любую библиотеку.
- Кроме того, разграничены экземпляры диалогов и бота. В версии 3 диалоги встраивались в базовый пакет SDK, стек обрабатывался внутренними средствами, а дочерние диалоги загружались с использованием методов *вызыва* и *перенаправления*. В версии 4 диалоги передаются в экземпляры бота как аргументы, делая процесс создания более гибким. При этом разработчик контролирует стек диалогов, а дочерние диалоги загружаются с использованием методов *BeginDialogAsync* и *ReplaceDialogAsync*.
- Кроме того, в версии 4 реализован класс `ActivityHandler` для автоматизации обработки разных типов действий, таких как *сообщение*, *обновление беседы* и *событие*.

В результате изменен синтаксис для разработки ботов на .NET. В частности, это касается создания объектов бота, определения диалогов и создания логики обработки событий.

Ниже приведено сравнение конструкций в пакетах SDK Bot Framework для .NET версий 3 и 4.

## Обработка входящих сообщений

### Версия 3

```
[BotAuthentication]
public class MessagesController : ApiController
{
    public async Task<HttpResponseMessage> Post([FromBody]Activity activity)
    {
        if (activity.GetActivityType() == ActivityTypes.Message)
        {
            await Conversation.SendAsync(activity, () => new Dialogs.RootDialog());
        }

        return Request.CreateResponse(HttpStatusCode.OK);
    }
}
```

### версия 4

```
[Route("api/messages")]
[ApiController]
public class BotController : ControllerBase
{
    private readonly IBotFrameworkHttpAdapter Adapter;
    private readonly IBot Bot;

    public BotController(IBotFrameworkHttpAdapter adapter, IBot bot)
    {
        Adapter = adapter;
        Bot = bot;
    }

    [HttpPost]
    public async Task PostAsync()
    {
        await Adapter.ProcessAsync(Request, Response, Bot);
    }
}
```

## Отправка сообщения пользователю

### Версия 3

```
await context.PostAsync("Hello and welcome to the help desk bot.");
```

### версия 4

```
await turnContext.SendActivityAsync("Hello and welcome to the help desk bot.");
```

## Загрузка корневого диалога

### Версия 3

```
await Conversation.SendAsync(activity, () => new Dialogs.RootDialog());
```

### версия 4

```

// Create a DialogExtensions class with a Run method.
public static class DialogExtensions
{
    public static async Task Run(
        this Dialog dialog,
        ITurnContext turnContext,
        IStatePropertyAccessor<DialogState> accessor,
        CancellationToken cancellationToken)
    {
        var dialogSet = new DialogSet(accessor);
        dialogSet.Add(dialog);

        var dialogContext = await dialogSet.CreateContextAsync(turnContext, cancellationToken);

        var results = await dialogContext.ContinueDialogAsync(cancellationToken);
        if (results.Status == DialogTurnStatus.Empty)
        {
            await dialogContext.BeginDialogAsync(dialog.Id, null, cancellationToken);
        }
    }
}

// Call it from the ActivityHandler's OnMessageActivityAsync override
protected override async Task OnMessageActivityAsync(
    ITurnContext<IMessageActivity> turnContext,
    CancellationToken cancellationToken)
{
    // Run the Dialog with the new message Activity.
    await Dialog.Run(
        turnContext,
        ConversationState.CreateProperty<DialogState>("DialogState"),
        cancellationToken);
}

```

## Запуск дочернего диалога

### Версия 3

```
context.Call(new NextDialog(), this.ResumeAfterNextDialog);
```

или диспетчер конфигурации служб

```
await context.Forward(new NextDialog(), this.ResumeAfterNextDialog, message);
```

### версия 4

```
dialogContext.BeginDialogAsync("<child-dialog-id>", options);
```

или диспетчер конфигурации служб

```
dialogContext.ReplaceDialogAsync("<child-dialog-id>", options);
```

## Завершение диалога

### Версия 3

```
context.Done(ReturnValue);
```

#### версия 4

```
await context.EndDialogAsync(ReturnValue);
```

## Запрос пользователю на ввод данных

#### Версия 3

```
PromptDialog.Choice(  
    context,  
    this.OnOptionSelected,  
    Options, PromptMessage,  
    ErrorMessage,  
    3,  
    PromptStyle.PerLine);
```

#### версия 4

```
// In the dialog's constructor, register the prompt, and waterfall steps.  
AddDialog(new TextPrompt(nameof(TextPrompt)));  
AddDialog(new WaterfallDialog(nameof(WaterfallDialog)), new WaterfallStep[]  
{  
    FirstStepAsync,  
    SecondStepAsync,  
});  
  
// The initial child Dialog to run.  
InitialDialogId = nameof(WaterfallDialog);  
  
// ...  
  
// In the first step, invoke the prompt.  
private async Task<DialogTurnResult> FirstStepAsync(  
    WaterfallStepContext stepContext,  
    CancellationToken cancellationToken)  
{  
    return await stepContext.PromptAsync(  
        nameof(TextPrompt),  
        new PromptOptions { Prompt = MessageFactory.Text("Please enter your destination.") },  
        cancellationToken);  
}  
  
// In the second step, retrieve the Result from the stepContext.  
private async Task<DialogTurnResult> SecondStepAsync(  
    WaterfallStepContext stepContext,  
    CancellationToken cancellationToken)  
{  
    var destination = (string)stepContext.Result;  
}
```

## Сохранение информации в состоянии диалога

#### Версия 3

В версии 3 все диалоги и их поля автоматически сериализовались.

#### версия 4

```
// StepContext values are auto-serialized in V4, and scoped to the dialog.  
stepContext.values.destination = destination;
```

## Запись изменений состояния в уровень сохраняемости

### Версия 3

Данные о состоянии автоматически сохраняются по умолчанию в конце шага.

### версия 4

```
// You now must explicitly save state changes before the end of the turn.  
await this.conversationState.saveChanges(context, false);  
await this.userState.saveChanges(context, false);
```

## Создание и регистрация хранилища состояний

### Версия 3

```
// Autofac was used internally by the sdk, and state was automatic.  
Conversation.UpdateContainer(  
builder =>  
{  
    builder.RegisterModule(new AzureModule(Assembly.GetExecutingAssembly()));  
    var store = new InMemoryDataStore();  
    builder.Register(c => store)  
        .Keyed<IBotDataStore<BotData>>(AzureModule.Key_DataStore)  
        .AsSelf()  
        .SingleInstance();  
});
```

### версия 4

```

// Create the storage we'll be using for User and Conversation state.
// In-memory storage is great for testing purposes.
services.AddSingleton<IStorage, MemoryStorage>();

// Create the user state (used in this bot's Dialog implementation).
services.AddSingleton<UserState>();

// Create the conversation state (used by the Dialog system itself).
services.AddSingleton<ConversationState>();

// The dialog that will be run by the bot.
services.AddSingleton<MainDialog>();

// Create the bot as a transient. In this case the ASP.NET controller is expecting an IBot.
services.AddTransient<IBot, DialogBot>();

// In the bot's ActivityHandler implementation, call SaveChangesAsync after the OnTurnAsync completes.
public class DialogBot : ActivityHandler
{
    protected readonly Dialog Dialog;
    protected readonly BotState ConversationState;
    protected readonly BotState UserState;

    public DialogBot(ConversationState conversationState, UserState userState, Dialog dialog)
    {
        ConversationState = conversationState;
        UserState = userState;
    }

    public override async Task OnTurnAsync(ITurnContext turnContext, CancellationToken cancellationToken = default(CancellationToken))
    {
        await base.OnTurnAsync(turnContext, cancellationToken);

        // Save any state changes that might have occurred during the turn.
        await ConversationState.SaveChangesAsync(turnContext, false, cancellationToken);
        await UserState.SaveChangesAsync(turnContext, false, cancellationToken);
    }

    protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
    {
        // Run the dialog, passing in the message activity for this turn.
        await Dialog.Run(turnContext, ConversationState.CreateProperty<DialogState>("DialogState"),
cancellationToken);
    }
}

```

## Перехват ошибки, связанной с диалогом

### Версия 3

```

// Create a custom IPostToBot implementation to catch exceptions.
public sealed class CustomPostUnhandledExceptionToUser : IPostToBot
{
    private readonly IPostToBot inner;
    private readonly IBotToUser botToUser;
    private readonly ResourceManager resources;
    private readonly System.Diagnostics.TraceListener trace;

    public CustomPostUnhandledExceptionToUser(IPostToBot inner, IBotToUser botToUser, ResourceManager resources, System.Diagnostics.TraceListener trace)
    {
        SetField.NotNull(out this.inner, nameof(inner), inner);
        SetField.NotNull(out this.botToUser, nameof(botToUser), botToUser);
        SetField.NotNull(out this.resources, nameof(resources), resources);
        SetField.NotNull(out this.trace, nameof(trace), trace);
    }

    async Task IPostToBot.PostAsync(IActivity activity, CancellationToken token)
    {
        try
        {
            await this.inner.PostAsync(activity, token);
        }
        catch (Exception ex)
        {
            try
            {
                // Log exception and send custom error message here.
                await this.botToUser.PostAsync("custom error message");
            }
            catch (Exception inner)
            {
                this.trace.WriteLine(inner);
            }
        }

        throw;
    }
}

// Register this using AutoFac, replacing the default PostUnhandledExceptionToUser.
builder
    ..RegisterType<CustomPostUnhandledExceptionToUser>()
    .Keyed<IPostToBot>(typeof(PostUnhandledExceptionToUser));

```

#### **версия 4**

```

// Provide an error handler in your implementation of the BotFrameworkHttpAdapter.
public class AdapterWithErrorHandler : BotFrameworkHttpAdapter
{
    public AdapterWithErrorHandler(
        ICredentialProvider credentialProvider,
        ILogger<BotFrameworkHttpAdapter> logger,
        ConversationState conversationState = null)
        : base(credentialProvider)

    {
        OnTurnError = async (turnContext, exception) =>
        {
            // Log any leaked exception from the application.
            logger.LogError($"Exception caught : {exception.Message}");

            // Send a catch-all apology to the user.
            await turnContext.SendActivityAsync("Sorry, it looks like something went wrong.");

            if (conversationState != null)
            {
                try
                {
                    // Delete the conversation state for the current conversation, to prevent the
                    // bot from getting stuck in a error-loop caused by being in a bad state.
                    // Conversation state is similar to "cookie-state" in a web page.
                    await conversationState.DeleteAsync(turnContext);
                }
                catch (Exception e)
                {
                    logger.LogError(
                        $"Exception caught on attempting to Delete ConversationState : {e.Message}");
                }
            }
        };
    }
}

```

## Обработка других типов действий

### Версия 3

```

// Within your MessageController, check the message type.
string messageType = activity.GetActivityType();
if (messageType == ActivityTypes.Message)
{
    await Conversation.SendAsync(activity, () => new Dialogs.RootDialog());
}
else if (messageType == ActivityTypes.DeleteUserData)
{
}
else if (messageType == ActivityTypes.ConversationUpdate)
{
}
else if (messageType == ActivityTypes.ContactRelationUpdate)
{
}
else if (messageType == ActivityTypes.Typing)
{
}

```

### версия 4

```
// In the bot's ActivityHandler implementation, override relevant methods.

protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
{
    // Handle message activities here.
}

protected override Task OnConversationUpdateActivityAsync(ITurnContext<IConversationUpdateActivity>
turnContext, CancellationToken cancellationToken)
{
    // Handle conversation update activities in general here.
}

protected override Task OnEventActivityAsync(ITurnContext<IEventActivity> turnContext, CancellationToken
cancellationToken)
{
    // Handle event activities in general here.
}
```

## Запись всех действий в журнал

### Версия 3

Используется [IActivityLogger](#).

```
builder.RegisterType<ActivityLoggerImplementation>().AsImplementedInterfaces().InstancePerDependency();

public class ActivityLoggerImplementation : IActivityLogger
{
    async Task IActivityLogger.LogAsync(IActivity activity)
    {
        // Store the activity.
    }
}
```

### версия 4

Используйте [ITranscriptLogger](#).

```
var transcriptMiddleware = new TranscriptLoggerMiddleware(new
TranscriptLoggerImplementation(Configuration.GetSection("StorageConnectionString").Value));
adapter.Use(transcriptMiddleware);

public class TranscriptLoggerImplementation : ITranscriptLogger
{
    async Task ITranscriptLogger.LogActivityAsync(IActivity activity)
    {
        // Store the activity.
    }
}
```

## Добавление хранилища состояния ботов

Интерфейс для хранения *пользовательских данных, данных диалогов и данных частных диалогов* изменен.

### Версия 3

Состояние сохранено путем использования реализации [IBotDataStore](#), а также включения его в систему состояния диалога из пакета SDK с помощью Autofac. Майкрософт предоставляет классы [MemoryStorage](#),

`DocumentDbBotDataStore`, `TableBotDataStore` И `SqlBotDataStore` В `Microsoft.Bot.Builder.Azure`.

Для хранения данных используется `IBotDataStore`.

```
Task<bool> FlushAsync(IAddress key, CancellationToken cancellationToken);
Task<T> LoadAsync(IAddress key, BotStoreType botStoreType, CancellationToken cancellationToken);
Task SaveAsync(IAddress key, BotStoreType botStoreType, T data, CancellationToken cancellationToken);
```

```
var dbPath = ConfigurationManager.AppSettings["DocDbPath"];
var dbKey = ConfigurationManager.AppSettings["DocDbKey"];
var docDbUri = new Uri(dbPath);
var storage = new DocumentDbBotDataStore(docDbUri, dbKey);
builder.Register(c => storage)
    .Keyed<IBotDataStore<BotData>>(AzureModule.Key_DataStore)
    .AsSelf()
    .SingleInstance();
```

#### версия 4

На уровне хранилища используется интерфейс `IStorage`. Укажите объект уровня хранилища при создании каждого объекта управления состоянием для бота, например `UserState`, `ConversationState` или `PrivateConversationState`. Объект управления состоянием предоставляет ключи для базового уровня хранилища и также используется как диспетчер свойств. Например, используйте `IPropertyManager.CreateProperty<T>(string name)`, чтобы создать метод доступа к свойству состояния. Эти методы доступа к свойствам используются для получения и хранения значений в базовом хранилище бота и за его пределами.

Используйте `IStorage` для хранения данных.

```
Task DeleteAsync(string[] keys, CancellationToken cancellationToken = default(CancellationToken));
Task<IDictionary<string, object>> ReadAsync(string[] keys, CancellationToken cancellationToken = default(CancellationToken));
Task WriteAsync(IDictionary<string, object> changes, CancellationToken cancellationToken = default(CancellationToken));
```

```
var storageOptions = new CosmosDbPartitionedStorageOptions()
{
    AuthKey = configuration["cosmosKey"],
    ContainerId = configuration["cosmosContainer"],
    CosmosDbEndpoint = configuration["cosmosPath"],
    DatabaseId = configuration["cosmosDatabase"]
};

IStorage dataStore = new CosmosDbPartitionedStorage(storageOptions);
var conversationState = new ConversationState(dataStore);
services.AddSingleton(conversationState);
```

#### NOTE

При использовании `CosmosDbPartitionedStorage` вам нужно создать базу данных и предоставить сведения о конечной точке Cosmos DB, ключе авторизации и идентификаторе базы данных, как показано выше. Для контейнера вам достаточно лишь указать идентификатор — бот самостоятельно создаст и правильно настроит его для сохранения состояния бота. Если вы создаете контейнер самостоятельно, не забудьте указать ключ секции `/id` и задать свойство `CosmosDbPartitionedStorageOptions.ContainerId`.

## Использование FormFlow

### Версия 3

[Microsoft.Bot.Builder.FormFlow](#) включен в базовый пакет SDK Bot Builder.

### версия 4

[Bot.Builder.Community.Dialogs.FormFlow](#) теперь представляет библиотеку Bot Builder Community.

Исходный код доступен в [репозитории](#) сообщества.

## Использование LuisDialog

### Версия 3

[Microsoft.Bot.Builder.Dialogs.LuisDialog](#) включен в базовый пакет SDK Bot Builder.

### версия 4

[Bot.Builder.Community.Dialogs.Luis](#) теперь представляет библиотеку Bot Builder Community.

Исходный код доступен в [репозитории](#) сообщества.

## Использование QnA Maker

### Версия 3

```
[Serializable]
[QnAMaker("QnAEndpointKey", "QnAKnowledgebaseId", <ScoreThreshold>, <TotalResults>, "QnAEndpointHostName")]
public class SimpleQnADialog : QnAMakerDialog
{}
```

### версия 4

```
public class QnABot : ActivityHandler
{
    private readonly IConfiguration _configuration;
    private readonly ILogger<QnABot> _logger;
    private readonly IHttpClientFactory _httpClientFactory;

    public QnABot(IConfiguration configuration, ILogger<QnABot> logger, IHttpClientFactory httpClientFactory)
    {
        _configuration = configuration;
        _logger = logger;
        _httpClientFactory = httpClientFactory;
    }

    protected override async Task OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
    {
        var httpClient = _httpClientFactory.CreateClient();

        var qnaMaker = new QnAMaker(new QnAMakerEndpoint
        {
            KnowledgeBaseId = _configuration["QnAKnowledgebaseId"],
            EndpointKey = _configuration["QnAEndpointKey"],
            Host = _configuration["QnAEndpointHostName"]
        },
        null,
        httpClient);

        _logger.LogInformation("Calling QnA Maker");

        // The actual call to the QnA Maker service.
        var response = await qnaMaker.GetAnswersAsync(turnContext);
        if (response != null && response.Length > 0)
        {
            await turnContext.SendActivityAsync(MessageFactory.Text(response[0].Answer), cancellationToken);
        }
        else
        {
            await turnContext.SendActivityAsync(MessageFactory.Text("No QnA Maker answers were found."),
cancellationToken);
        }
    }
}
```

# Перенос бота .NET версии 3 в бота .NET Framework версии 4

27.03.2021 • 37 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье описывается, как преобразовать бота [ContosoHelpdeskChatBot версии 3](#) в бота версии 4, *не меняя тип проекта*. Это по-прежнему будет проект .NET Framework. Преобразование включает в себя следующие действия:

1. Обновление и установка пакетов NuGet
2. Обновление файла global.asax.cs
3. Обновление класса MessagesController.
4. Преобразование диалогов.

Результатом этого преобразования является [бот ContosoHelpdeskChatBot в .NET Framework версии 4](#).

Сведения о переносе в бот версии 4 .NET Core в новом проекте см. в статье [Перенос бота .NET версии 3 в бот .NET Core версии 4](#).

Пакет SDK версии 4 для Bot Framework использует тот же базовый REST API, что и пакет SDK версии 3. Но в пакете SDK версии 4 выполнен рефакторинг кода предыдущей версии пакета SDK для повышения гибкости и улучшения контроля над ботами. Основные изменения в пакете SDK:

- Управление состоянием осуществляется через соответствующие объекты и методы доступа к свойствам.
- Также изменились процессы настройки обработчика шагов и передачи ему действий.
- Элементов с возможностью оценки больше нет. Вы можете проверить наличие глобальных команд в обработчике шагов, прежде чем передавать средства управления в диалоги.
- Новая библиотека Dialogs, которая сильно отличается от предыдущей версии. Вам потребуется преобразовать все старые диалоги в новую систему, используя компонентные, каскадные диалоги и созданную сообществом реализацию диалогов Formflow для версии 4.

См. дополнительные сведения об [отличиях между версиями 3 и 4 пакета SDK для .NET](#).

## Обновление и установка пакетов NuGet

1. Обновите Microsoft.Bot.Builder.Azure и Microsoft.Bot.Builder.Integration.AspNet.WebApi до последней стабильной версии.

Так вы обновите пакеты Microsoft.Bot.Builder и Microsoft.Bot.Connector, которые указаны как зависимости.

2. Удалите пакет Microsoft.Bot.Builder.History. Он не входит в состав пакета SDK версии 4.
3. Добавьте Autofac.WebApi2.

Мы будем его использовать для внедрения зависимостей в ASP.NET.

4. Добавьте Bot.Builder.Community.Dialogs.Formflow.

Это библиотека сообщества для сборки диалогов версии 4 из файлов определения Formflow версии 3. Одно из зависимостей для нее является Microsoft.Bot.Builder.Dialogs, поэтому она тоже

автоматически устанавливается.

#### TIP

Если ваш проект предназначен для .NET Framework 4.6, понадобится обновление до версии 4.6.1 или последующей, так как `Bot.Builder.Community.Dialogs.Formflow` является библиотекой .NET Standard 2.0. Дополнительные сведения см. в разделе [Поддержка реализации .NET](#).

Если на этом этапе выполнить сборку, вы получите ошибки компилятора. На них можно не обращать внимания. Когда преобразование полностью завершится, код будет полностью рабочим.

## Обновление файла global.asax.cs

Есть некоторые изменения в формировании шаблонов, и теперь для версии 4 нужно самостоятельно настроить элементы инфраструктуры [управления состоянием](#). К примеру, в версии 4 применяется адаптер ботов для проверки подлинности и передачи действий в код бота, а это означает, что все свойства состояния нужно объявить заранее.

Мы создадим свойство состояния для `DialogState`, который потребуется в версии 4 для поддержки диалогов. Мы будем использовать внедрение зависимостей, чтобы получить информацию, которая нужна контроллеру и боту.

В файле `Global.asax.cs` выполните следующее:

1. Обновите инструкции `using`.

```
using Autofac;
using Autofac.Integration.WebApi;
using ContosoHelpdeskChatBot.Bots;
using ContosoHelpdeskChatBot.Dialogs;
using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.BotFramework;
using Microsoft.Bot.Builder.Integration.AspNet.WebApi;
using Microsoft.Bot.Connector.Authentication;
using System.Reflection;
using System.Web.Http;
```

2. Удалите из метода `Application_Start` следующие строки:

```
BotConfig.UpdateConversationContainer();
this.RegisterBotModules();
```

Теперь вставьте эту строку:

```
GlobalConfiguration.Configure(BotConfig.Register);
```

3. Удалите метод `RegisterBotModules`, на который больше нет ссылки.

4. Замените метод `BotConfig.UpdateConversationContainer` методом `BotConfig.Register`, где мы зарегистрируем объекты, необходимые для поддержки внедрения зависимостей. Этот бот-помощник не использует состояние *пользователя* или *приватной беседы*, поэтому мы создадим только объект управления состоянием беседы.

```

public static void Register(HttpConfiguration config)
{
    var builder = new ContainerBuilder();
    builder.RegisterApiControllers(Assembly.GetExecutingAssembly());

    // The ConfigurationCredentialProvider will retrieve the MicrosoftAppId and
    // MicrosoftAppPassword from Web.config
    builder.RegisterType<ConfigurationCredentialProvider>().As<ICredentialProvider>
    ().SingleInstance();

    // Create the Bot Framework Adapter with error handling enabled.
    builder.RegisterType<AdapterWithErrorHandler>().As<IBotFrameworkHttpAdapter>().SingleInstance();

    // The Memory Storage used here is for local bot debugging only. When the bot
    // is restarted, everything stored in memory will be gone.
    IStorage dataStore = new MemoryStorage();

    // Create Conversation State object.
    // The Conversation State object is where we persist anything at the conversation-scope.
    var conversationState = new ConversationState(dataStore);
    builder.RegisterInstance(conversationState).As<ConversationState>().SingleInstance();

    // Register the main dialog, which is injected into the DialogBot class
    builder.RegisterType<RootDialog>().SingleInstance();

    // Register the DialogBot with RootDialog as the IBot interface
    builder.RegisterType<DialogBot<RootDialog>>().As<IBot>();

    var container = builder.Build();
    var resolver = new AutofacWebApiDependencyResolver(container);
    config.DependencyResolver = resolver;
}

```

## Обновление класса MessagesController.

Так как именно здесь бот запускает шаг в версии 4, нам нужно внести значительные изменения. Все остальное, кроме обработчика шагов бота, можно считать стандартным кодом. В файле `Controllers\MessagesController.cs` сделайте следующее:

1. Обновите инструкции `using`.

```

using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.Integration.AspNet.WebApi;
using System.Net.Http;
using System.Threading.Tasks;
using System.Web.Http;

```

2. Удалите из класса атрибут `[BotAuthentication]`. В версии 4 проверку подлинности выполняет адаптер бота.
3. Добавьте эти поля и конструктор для инициализации. ASP.NET и Autofac используют внедрение зависимостей, чтобы получить значения параметров. (Чтобы поддерживать эту возможность, мы зарегистрировали объекты бота и адаптера в `Global.asax.cs`.)

```
private readonly IBotFrameworkHttpAdapter _adapter;
private readonly IBot _bot;

public MessagesController(IBotFrameworkHttpAdapter adapter, IBot bot)
{
    _adapter = adapter;
    _bot = bot;
}
```

4. Замените текст метода `Post`. С помощью адаптера мы вызовем цикл обработки сообщений бота (обработчик шагов).

```
public async Task<HttpResponseMessage> Post()
{
    var response = new HttpResponseMessage();

    // Delegate the processing of the HTTP POST to the adapter.
    // The adapter will invoke the bot.
    await _adapter.ProcessAsync(Request, response, _bot);
    return response;
}
```

#### Удаление классов `Cancelable` и `GlobalMessageHandlersBotModule`

Так как диалоги с возможностью оценки в версии 4 не существуют, а обработчик шагов теперь реагирует на сообщение `cancel`, можно спокойно удалить классы `Cancelable` (в `Dialogs\Cancelable.cs`) и `GlobalMessageHandlersBotModule`.

## Создание класса бота

В версии 4 логика цикла обработки сообщений или обработчика шагов находится преимущественно в файле бота. Мы получаем данные от `ActivityHandler`, который определяет обработчики для распространенных типов действий.

1. Создайте файл `Bots\DialogBots.cs`.

2. Обновите инструкции `using`.

```
using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Schema;
using System.Threading;
using System.Threading.Tasks;
```

3. Получите `DialogBot` из `ActivityHandler` и добавьте универсальный параметр для диалога.

```
public class DialogBot<T> : ActivityHandler where T : Dialog
```

4. Добавьте эти поля и конструктор для инициализации. Снова-таки, ASP.NET и Autofac используют внедрение зависимостей, чтобы получить значения параметров.

```
protected readonly Dialog _dialog;
protected readonly BotState _conversationState;

public DialogBot(ConversationState conversationState, T dialog)
{
    _conversationState = conversationState;
    _dialog = dialog;
}
```

5. Переопределите `OnMessageActivityAsync` для вызова основного диалога. (Мы определим метод расширения `Run` чуть позже.)

```
protected override async Task OnMessageActivityAsync(
    ITurnContext<IMessageActivity> turnContext,
    CancellationToken cancellationToken)
{
    // Run the Dialog with the new message Activity.
    await _dialog.Run(
        turnContext,
        _conversationState.CreateProperty<DialogState>("DialogState"),
        cancellationToken);
}
```

6. Переопределите `OnTurnAsync` для сохранения состояния беседы в конце шага. В версии 4 нам нужно сделать это явным образом, чтобы записать состояние на уровне сохраняемости. Метод `ActivityHandler.OnTurnAsync` вызывает методы конкретного обработчика действий с учетом типа полученного действия. Поэтому мы сохраним состояние после вызова базового метода.

```
public override async Task OnTurnAsync(ITurnContext turnContext, CancellationToken cancellationToken)
{
    await base.OnTurnAsync(turnContext, cancellationToken);

    // Save any state changes that might have occurred during the turn.
    await _conversationState.SaveChangesAsync(turnContext, false, cancellationToken);
}
```

### Создание метода расширения запуска

Мы создаем метод расширения, чтобы объединить код для запуска простого компонентного диалога из бота.

Создайте файл `DialogExtensions.cs` и реализуйте метод расширения `Run`.

```

using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.Dialogs;
using System;
using System.Threading;
using System.Threading.Tasks;

namespace ContosoHelpdeskChatBot
{
    public static class DialogExtensions
    {
        public static async Task Run(
            this Dialog dialog,
            ITurnContext turnContext,
            IStatePropertyAccessor<DialogState> accessor,
            CancellationToken cancellationToken)
        {
            var dialogSet = new DialogSet(accessor);
            dialogSet.Add(dialog);

            var dialogContext = await dialogSet.CreateContextAsync(turnContext, cancellationToken);

            // Handle 'cancel' interruption
            if (turnContext.Activity.Text.Equals("cancel", StringComparison.InvariantCultureIgnoreCase))
            {
                var reply = turnContext.Activity.CreateReply($"Ok restarting conversation.");
                await turnContext.SendActivityAsync(reply);
                await dialogContext.CancelAllDialogsAsync();
            }

            var results = await dialogContext.ContinueDialogAsync(cancellationToken);
            if (results.Status == DialogTurnStatus.Empty)
            {
                await dialogContext.BeginDialogAsync(dialog.Id, null, cancellationToken);
            }
        }
    }
}

```

## Преобразование диалогов.

Мы будем вносить в исходные диалоги изменения, позволяющие использовать их с пакетом SDK версии 4. Пока не беспокойтесь об ошибках компилятора. Они исчезнут, когда преобразование завершится. Мы не будем изменять исходный код больше, чем необходимо, поэтому некоторые предупреждения компилятора сохранятся после завершения переноса.

Все диалоги теперь будут не реализацией интерфейса `IDialog<object>` из версии 3, а станут производными от `ComponentDialog`.

Этот бот использует четыре диалога, и все их нужно преобразовать:

ДИАЛОГ	ОПИСАНИЕ
<code>RootDialog</code>	Представляет варианты на выбор и запускает другие диалоги.
<code>InstallAppDialog</code>	Обрабатывает запросы на установку приложения на компьютер.
<code>LocalAdminDialog</code>	Обрабатывает запросы на получение прав локального администратора на компьютере.

ДИАЛОГ	ОПИСАНИЕ
ResetPasswordDialog	Обрабатывает запросы на сброс пароля.

Все они собирают данные, но не выполняют никаких операций на компьютере.

#### Внесение изменений в диалоги на уровне решения

1. Для всего решения измените все вхождения `IDialog<object>` на `ComponentDialog`.
2. Для всего решения измените все вхождения `IDialogContext` на `DialogContext`.
3. Для каждого класса диалогов удалите атрибут `[Serializable]`.

Поток управления и обмен сообщениями в диалогах теперь обрабатываются другими способами, и это нужно учитывать при преобразовании каждого диалога.

ОПЕРАЦИЯ	КОД ДЛЯ ВЕРСИИ 3	КОД ДЛЯ ВЕРСИИ 4
Обработка запуска диалога	Реализуйте <code>IDialog.StartAsync</code>	Сделайте этот шаг первым в каскадном диалоге или реализуйте <code>Dialog.BeginDialogAsync</code>
Обработка продолжения диалога	Вызовите <code>IDialogContext.Wait</code>	Добавьте в каскадный диалог дополнительные шаги или реализуйте <code>Dialog.ContinueDialogAsync</code>
Отправка сообщения пользователю	Вызовите <code>IDialogContext.PostAsync</code>	Вызовите <code>ITurnContext.SendActivityAsync</code>
Запуск дочернего диалога	Вызовите <code>IDialogContext.Call</code>	Вызовите <code>DialogContext.BeginDialogAsync</code>
Информирование о завершении текущего диалога	Вызовите <code>IDialogContext.Done</code>	Вызовите <code>DialogContext.EndDialogAsync</code>
Получение введенных пользователем данных	Используйте параметр <code>IAwaitable&lt;IMessageActivity&gt;</code>	Используйте приглашение в каскадном диалоге или <code>ITurnContext.Activity</code>

Примечания о коде для версии 4:

- В коде диалога используйте свойство `DialogContext.Context`, чтобы получить контекст текущего шага.
- Каскадные шаги имеют параметр `WaterfallStepContext`, который является производным от `DialogContext`.
- Все конкретные классы диалогов и приглашений наследуют от абстрактного класса `Dialog`.
- Идентификатор присваивается при создании компонентного диалога. Каждый диалог в наборе диалогов должен иметь уникальный в пределах этого набора идентификатор.

#### Обновление корневого диалога

В этом боте корневой диалог предлагает пользователю выбрать один вариант из набора, а затем на основе этого выбора запускает дочерний диалог. Этот цикл повторяется неограниченно долго, пока существует беседа.

- Основной поток можно создать в формате каскадного диалога — это новая концепция в пакете SDK версии 4. Он выполняет фиксированный набор шагов в заданном порядке. Подробнее см. статью

## Реализация последовательной беседы.

- Запросы теперь обрабатываются через классы prompt. Эти короткие дочерние диалоги предлагают ввести данные, выполняют минимальную обработку и проверку этих данных и возвращают полученное значение. Дополнительные сведения о сборе данных от пользователя с помощью запросов диалога см. в [этой статье](#).

В файле Dialogs/RootDialog.cs сделайте следующее:

1. Обновите инструкции `using`.

```
using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Builder.Dialogs.Choices;
using System;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;
```

2. Нам нужно преобразовать варианты `HelpdeskOptions` из списка строк в список вариантов. Они будут применяться в строке запроса, которая в качестве допустимых входных данных принимает номер выбранного варианта в списке, значение любого из вариантов или его синоним.

```
private static List<Choice> HelpdeskOptions = new List<Choice>()
{
    new Choice(InstallAppOption) { Synonyms = new List<string> { "install" } },
    new Choice(ResetPasswordOption) { Synonyms = new List<string> { "password" } },
    new Choice(LocalAdminOption) { Synonyms = new List<string> { "admin" } }
};
```

3. Добавьте конструктор. Этот код делает следующее:

- Каждому экземпляру диалога при создании присваивается идентификатор. Идентификатор диалога входит в набор диалогов, к которому добавляется этот диалог. Не забывайте, что бот уже инициализирован с использованием объекта диалога в классе `MessageController`. Для всех `ComponentDialog` есть свой внутренний набор диалогов с отдельным набором идентификаторов диалогов.
- Включает остальные диалоги, в том числе запросы выбора, в качестве дочерних диалогов. Здесь мы используем для каждого диалога в качестве идентификатора соответствующее имя класса.
- Определяет каскадный диалог с тремя шагами. Их мы реализуем чуть позже.
  - Сначала диалог предлагает пользователю выбрать задание для выполнения.
  - Затем он запускает дочерний диалог, связанный с выбранным вариантом.
  - И уже потом он перезапускает сам себя.
- Каждый шаг каскадного диалога является делегатом. Мы реализуем их далее, по возможности сохраняя существующий код каждого исходного диалога.
- При запуске компонентного диалога будет запущен *начальный диалог*. По умолчанию это первый дочерний диалог, добавленный в компонентный диалог. Мы явным образом определим свойство `InitialDialogId`. Это значит, что нам не нужно добавлять основной каскадный диалог первым в набор диалогов. Например, вы можете сначала добавить запросы, и это не вызовет никаких проблем во время выполнения.

```

public RootDialog()
    : base(nameof(RootDialog))
{
    InitialDialogId = nameof(WaterfallDialog);
    AddDialog(new WaterfallDialog(nameof(WaterfallDialog), new WaterfallStep[]
    {
        PromptForOptionsAsync,
        ShowChildDialogAsync,
        ResumeAfterAsync,
    }));
    AddDialog(new InstallAppDialog());
    AddDialog(new LocalAdminDialog());
    AddDialog(new ResetPasswordDialog());
    AddDialog(new ChoicePrompt(nameof(ChoicePrompt)));
}

```

4. Метод `StartAsync` можно удалить. При запуске компонентный диалог автоматически выполняет свой *начальный* диалог. В нашем примере это тот каскадный диалог, который мы определяем в конструкторе. Он автоматически начинает выполнение с первого действия.

5. Мы удалим методы `MessageReceivedAsync` и `ShowOptions`, а затем заменим их первым шагом нашего каскадного диалога. Эти два метода ранее приветствовали пользователя и предлагали выбор из доступных вариантов.

- Здесь вы видите список значений, приветствия и сообщения об ошибках, которые передаются в вызов запроса на выбор в качестве параметров.
- Нам не нужно указывать следующий метод, который будет вызван в диалоге, так как каскадный диалог автоматически перейдет к следующему шагу после завершения запроса на выбор.
- Запрос выбора повторяется в цикле, пока не получит допустимые входные данные или весь стек диалогов не будет отменен.

```

private async Task<DialogTurnResult> PromptForOptionsAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    // Prompt the user for a response using our choice prompt.
    return await stepContext.PromptAsync(
        nameof(ChoicePrompt),
        new PromptOptions()
    {
        Choices = HelpdeskOptions,
        Prompt = MessageFactory.Text(GreetMessage),
        RetryPrompt = MessageFactory.Text(ErrorMessage)
    },
    cancellationToken);
}

```

6. `onOptionSelected` можно заменить вторым шагом каскадного диалога. Здесь мы также запускаем дочерний диалог в зависимости от ввода пользователя.

- Запрос выбора возвращает значение `FoundChoice`. Оно передается в свойстве `Result` этого шага. Стек диалогов обрабатывает все возвращаемые значения как объекты. Если это значение возвращается из вашего диалога, вы получите тип значения для этого объекта. Список значений, возвращаемых разными типами, вы можете изучить [здесь](#).
- Так как запрос выбора не создает исключение, мы можем удалить блок try-catch.
- Но нам нужно добавить механизм передачи управления, чтобы этот метод всегда возвращал допустимое значение. Этот код никогда не должен выполняться, но если это произойдет из-за сбоя, он корректно завершит работу диалога.

```

private async Task<DialogTurnResult> ShowChildDialogAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    // string optionSelected = await userReply;
    var optionSelected = (stepContext.Result as FoundChoice).Value;

    switch (optionSelected)
    {
        case InstallAppOption:
            //context.Call(new InstallAppDialog(), this.ResumeAfterOptionDialog);
            //break;
            return await stepContext.BeginDialogAsync(
                nameof(InstallAppDialog),
                cancellationToken);

        case ResetPasswordOption:
            //context.Call(new ResetPasswordDialog(), this.ResumeAfterOptionDialog);
            //break;
            return await stepContext.BeginDialogAsync(
                nameof(ResetPasswordDialog),
                cancellationToken);

        case LocalAdminOption:
            //context.Call(new LocalAdminDialog(), this.ResumeAfterOptionDialog);
            //break;
            return await stepContext.BeginDialogAsync(
                nameof(LocalAdminDialog),
                cancellationToken);
    }

    // We shouldn't get here, but fail gracefully if we do.
    await stepContext.Context.SendActivityAsync(
        "I don't recognize that option.",
        cancellationToken: cancellationToken);
    // Continue through to the next step without starting a child dialog.
    return await stepContext.NextAsync(cancellationToken: cancellationToken);
}

```

7. Наконец, замените старый метод `ResumeAfterOptionDialog` последним шагом каскадного диалога.

- В исходном диалоге мы завершали работу и возвращали номер билета. Теперь же мы перезапускаем каскадный диалог, заменяя его старый экземпляр в стеке диалогов новым экземпляром того же диалога. Это стало возможным, так как исходное приложение всегда игнорирует возвращаемое значение (номер билета) и перезапускает корневой диалог.

```

private async Task<DialogTurnResult> ResumeAfterAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    try
    {
        //var message = await userReply;
        var message = stepContext.Context.Activity;

        var ticketNumber = new Random().Next(0, 20000);
        //await context.PostAsync($"Thank you for using the Helpdesk Bot. Your ticket number is {ticketNumber}.");
        await stepContext.Context.SendActivityAsync(
            $"Thank you for using the Helpdesk Bot. Your ticket number is {ticketNumber}.",
            cancellationToken: cancellationToken);

        //context.Done(ticketNumber);
    }
    catch (Exception ex)
    {
        // await context.PostAsync($"Failed with message: {ex.Message}");
        await stepContext.Context.SendActivityAsync(
            $"Failed with message: {ex.Message}",
            cancellationToken: cancellationToken);

        // In general resume from task after calling a child dialog is a good place to handle
        exceptions
        // try catch will capture exceptions from the bot framework awaitable object which is
        essentially "userReply"
        logger.Error(ex);
    }

    // Replace on the stack the current instance of the waterfall with a new instance,
    // and start from the top.
    return await stepContext.ReplaceDialogAsync(
        nameof(WaterfallDialog),
        cancellationToken: cancellationToken);
}

```

## Обновление диалога установки приложения

Диалог установки приложения выполняет несколько логических задач, которые мы сейчас настроим в четырех шагах каскадного диалога. Перенос существующего кода в каскадные шаги будет отдельной логической задачей для каждого из диалогов. Для каждого шага указан исходный метод, из которого взят его код.

1. Запрос на ввод строки поиска пользователем.
2. Запрос возможных совпадений в базе данных.
  - Если есть одно совпадение, выбор этого значения и продолжение работы.
  - Если совпадений несколько, запрос на выбор одного варианта пользователем.
  - Если совпадений нет, выход из диалога.
3. Запрос на выбор компьютера, где пользователем будет установлено приложение.
4. Сохранение сведений в базе данных и отправка сообщения с подтверждением.

В файле `Dialogs/InstallAppDialog.cs` сделайте следующее:

1. Обновите инструкции `using`.

```
using ContosoHelpdeskChatBot.Models;
using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Builder.Dialogs.Choices;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
```

2. Определите константу для ключа, чтобы отслеживать собранные сведения.

```
// Set up keys for managing collected information.
private const string InstallInfo = "installInfo";
```

3. Добавьте конструктор и инициализируйте набор диалогов для компонентного диалога.

```
public InstallAppDialog()
    : base(nameof(InstallAppDialog))
{
    // Initialize our dialogs and prompts.
    InitialDialogId = nameof(WaterfallDialog);
    AddDialog(new WaterfallDialog(nameof(WaterfallDialog), new WaterfallStep[] {
        GetSearchTermAsync,
        ResolveAppNameAsync,
        GetMachineNameAsync,
        SubmitRequestAsync,
    }));
    AddDialog(new TextPrompt(nameof(TextPrompt)));
    AddDialog(new ChoicePrompt(nameof(ChoicePrompt)));
}
```

4. `StartAsync` можно заменить первым шагом каскадного диалога.

- Но нам придется самостоятельно управлять состоянием, для которого мы применим объект установки приложения в состоянии диалога.
- Сообщение с запросом пользовательских данных становится необязательным параметром в вызове запроса.

```
private async Task<DialogTurnResult> GetSearchTermAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    // Create an object in dialog state in which to track our collected information.
    stepContext.Values[InstallInfo] = new InstallApp();

    // Ask for the search term.
    return await stepContext.PromptAsync(
        nameof(TextPrompt),
        new PromptOptions
        {
            Prompt = MessageFactory.Text("Ok let's get started. What is the name of the application?"),
        },
        cancellationToken);
}
```

5. `appNameAsync` И `multipleAppsAsync` можно заменить вторым шагом каскадного диалога.

- Теперь мы получаем результат запроса, а не просто проверяем последнее сообщение пользователя.

- Запрос к базе данных и инструкции if организованы здесь так же, как в `appNameAsync`. Код в каждом блоке инструкции if обновлен для поддержки диалогов версии 4.
  - Если обнаружится только одно совпадение, мы обновляем состояние диалога и переходим к следующему шагу.
  - Если совпадений несколько, мы применяем запрос выбора и предлагаем пользователю выбрать вариант из списка. Это означает, что можно просто удалить `multipleAppsAsync`.
  - Если совпадений нет, мы завершаем диалог и возвращаем значение NULL в корневой диалог.

```
private async Task<DialogTurnResult> ResolveAppNameAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    // Get the result from the text prompt.
    var appname = stepContext.Result as string;

    // Query the database for matches.
    var names = await this.GetAppsAsync(appname);

    if (names.Count == 1)
    {
        // Get our tracking information from dialog state and add the app name.
        var install = stepContext.Values[InstallInfo] as InstallApp;
        install.AppName = names.First();

        return await stepContext.NextAsync();
    }
    else if (names.Count > 1)
    {
        // Ask the user to choose from the list of matches.
        return await stepContext.PromptAsync(
            nameof(ChoicePrompt),
            new PromptOptions
            {
                Prompt = MessageFactory.Text("I found the following applications. Please choose one:"),
                Choices = ChoiceFactory.ToChoices(names),
            },
            cancellationToken);
    }
    else
    {
        // If no matches, exit this dialog.
        await stepContext.Context.SendActivityAsync(
            $"Sorry, I did not find any application with the name '{appname}'.",
            cancellationToken: cancellationToken);

        return await stepContext.EndDialogAsync(null, cancellationToken);
    }
}
```

6. Метод `appNameAsync` также запрашивает у пользователя имя компьютера после обработки запроса. Мы реализуем эту часть логики в следующем шаге каскадного диалога.

- Не забывайте, что в версии 4 нам нужно самостоятельно управлять состоянием. Единственной сложностью будет то, что на этот шаг с предыдущего мы можем попасть через две разные ветви логики.
- Для запроса имени компьютера у пользователя мы применим тот же запрос текста, что и ранее, но с другими вариантами ответов.

```

private async Task<DialogTurnResult> GetMachineNameAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    // Get the tracking info. If we don't already have an app name,
    // Then we used the choice prompt to get it in the previous step.
    var install = stepContext.Values[InstallInfo] as InstallApp;
    if (install.AppName is null)
    {
        install.AppName = (stepContext.Result as FoundChoice).Value;
    }

    // We now need the machine name, so prompt for it.
    return await stepContext.PromptAsync(
        nameof(TextPrompt),
        new PromptOptions
        {
            Prompt = MessageFactory.Text(
                $"Found {install.AppName}. What is the name of the machine to install application?"),
        },
        cancellationToken);
}

```

7. Логика из метода `machineNameAsync` попадает в заключительный шаг каскадного диалога.

- Мы извлекаем имя компьютера из результата, возвращенного запросом текста, и обновляем состояние диалога.
- Вызов для обновления базы данных мы удалим, так как вспомогательный код размещается в другом проекте.
- Затем мы отправляем пользователю сообщение об успешном выполнении и заканчиваем диалог.

```

private async Task<DialogTurnResult> SubmitRequestAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    var install = default(InstallApp);
    if (stepContext.Reason != DialogReason.CancelCalled)
    {
        // Get the tracking info and add the machine name.
        install = stepContext.Values[InstallInfo] as InstallApp;
        install.MachineName = stepContext.Context.Activity.Text;

        //TODO: Save to this information to the database.
    }

    await stepContext.Context.SendActivityAsync(
        $"Great, your request to install {install.AppName} on {install.MachineName} has been
scheduled.",
        cancellationToken: cancellationToken);

    return await stepContext.EndDialogAsync(null, cancellationToken);
}

```

8. Чтобы сымитировать вызов базы данных, мы создадим вызов с использованием `getAppsAsync`, что позволит обратиться к статическому списку, а не реальной базе данных.

```
private async Task<List<string>> GetAppsAsync(string Name)
{
    var names = new List<string>();

    // Simulate querying the database for applications that match.
    return (from app in AppMsis
            where app.ToLower().Contains(Name.ToLower())
            select app).ToList();
}

// Example list of app names in the database.
private static readonly List<string> AppMsis = new List<string>
{
    "μTorrent 3.5.0.44178",
    "7-Zip 17.1",
    "Ad-Aware 9.0",
    "Adobe AIR 2.5.1.17730",
    "Adobe Flash Player (IE) 28.0.0.105",
    "Adobe Flash Player (Non-IE) 27.0.0.130",
    "Adobe Reader 11.0.14",
    "Adobe Shockwave Player 12.3.1.201",
    "Advanced SystemCare Personal 11.0.3",
    "Auslogics Disk Defrag 3.6",
    "avast! 4 Home Edition 4.8.1351",
    "AVG Anti-Virus Free Edition 9.0.0.698",
    "Bonjour 3.1.0.1",
    "CCleaner 5.24.5839",
    "Chmod Calculator 20132.4",
    "CyberLink PowerDVD 17.0.2101.62",
    "DAEMON Tools Lite 4.46.1.328",
    "FileZilla Client 3.5",
    "Firefox 57.0",
    "Foxit Reader 4.1.1.805",
    "Google Chrome 66.143.49260",
    "Google Earth 7.3.0.3832",
    "Google Toolbar (IE) 7.5.8231.2252",
    "GSpot 2701.0",
    "Internet Explorer 903235.0",
    "iTunes 12.7.0.166",
    "Java Runtime Environment 6 Update 17",
    "K-Lite Codec Pack 12.1",
    "Malwarebytes Anti-Malware 2.2.1.1043",
    "Media Player Classic 6.4.9.0",
    "Microsoft Silverlight 5.1.50907",
    "Mozilla Thunderbird 57.0",
    "Nero Burning ROM 19.1.1005",
    "OpenOffice.org 3.1.1 Build 9420",
    "Opera 12.18.1873",
    "Paint.NET 4.0.19",
    "Picasa 3.9.141.259",
    "QuickTime 7.79.80.95",
    "RealPlayer SP 12.0.0.319",
    "Revo Uninstaller 1.95",
    "Skype 7.40.151",
    "Spybot - Search & Destroy 1.6.2.46",
    "SpywareBlaster 4.6",
    "TuneUp Utilities 2009 14.0.1000.353",
    "Unlocker 1.9.2",
    "VLC media player 1.1.6",
    "Winamp 5.56 Build 2512",
    "Windows Live Messenger 2009 16.4.3528.331",
    "WinPatrol 2010 31.0.2014",
    "WinRAR 5.0",
};

};
```

В версии 3 этот диалог приветствовал пользователя, запускал диалог Formflow и сохранял результат в базе данных. Эти действия легко преобразовать в каскадный диалог с двумя шагами.

1. Обновите инструкции `using`. Обратите внимание на то, что этот диалог включает диалог Formflow из версии 3. В версии 4 мы применим библиотеку Formflow, созданную сообществом.

```
using Bot.Builder.Community.Dialogs.FormFlow;
using ContosoHelpdeskChatBot.Models;
using Microsoft.Bot.Builder.Dialogs;
using System.Threading;
using System.Threading.Tasks;
```

2. Мы можем удалить свойство экземпляра для `LocalAdmin`, так как результат будет доступен в состоянии диалога.
3. Добавьте конструктор и инициализируйте набор диалогов для компонентного диалога. Диалог Formflow создается аналогичным образом. Мы просто добавляем его в набор диалогов для компонентного диалога в конструкторе.

```
public LocalAdminDialog() : base(nameof(LocalAdminDialog))
{
    InitialDialogId = nameof(WaterfallDialog);
    AddDialog(new WaterfallDialog(nameof(WaterfallDialog), new WaterfallStep[]
    {
        BeginFormflowAsync,
        SaveResultAsync,
    }));
    AddDialog(FormDialog.FromForm(BuildLocalAdminForm, FormOptions.PromptInStart));
}
```

4. `StartAsync` можно заменить первым шагом каскадного диалога. Мы уже создали Formflow в конструкторе, а остальные две инструкции будут преобразованы соответствующим образом. Обратите внимание, что `FormBuilder` присваивает имя типа модели в качестве идентификатора созданного диалога (`LocalAdminPrompt` для этой модели).

```
private async Task<DialogTurnResult> BeginFormflowAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    await stepContext.Context.SendActivityAsync("Great I will help you request local machine
admin.");

    // Begin the Formflow dialog.
    return await stepContext.BeginDialogAsync(
        nameof(LocalAdminPrompt),
        cancellationToken: cancellationToken);
}
```

5. `ResumeAfterLocalAdminFormDialog` можно заменить вторым шагом каскадного диалога. Нам нужно получать возвращаемое значение из контекста шага, а не свойства экземпляра.

```

private async Task<DialogTurnResult> SaveResultAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    // Get the result from the Formflow dialog when it ends.
    if (stepContext.Reason != DialogReason.CancelCalled)
    {
        var admin = stepContext.Result as LocalAdminPrompt;

        //TODO: Save to this information to the database.
    }

    return await stepContext.EndDialogAsync(null, cancellationToken);
}

```

6. `BuildLocalAdminForm` почти не изменяется, за исключением того, что теперь FormFlow не будет обновлять свойство экземпляра.

```

// Nearly the same as before.
private IForm<LocalAdminPrompt> BuildLocalAdminForm()
{
    // Here's an example of how validation can be used with FormBuilder.
    return new FormBuilder<LocalAdminPrompt>()
        .Field(nameof(LocalAdminPrompt.MachineName),
        validate: async (state, value) =>
    {
        var result = new ValidateResult { IsValid = true, Value = value };
        //add validation here

        //this.admin.MachineName = (string)value;
        return result;
    })
        .Field(nameof(LocalAdminPrompt.AdminDuration),
        validate: async (state, value) =>
    {
        var result = new ValidateResult { IsValid = true, Value = value };
        //add validation here

        //this.admin.AdminDuration = Convert.ToInt32((long)value) as int?;
        return result;
    })
        .Build();
}

```

### Обновление диалога смены пароля

В версии 3 этот диалог приветствует пользователя, авторизует пользователя по коду доступа, возвращает ошибку авторизации или запускает диалог Formflow и сбрасывает значение пароля. Все это неплохо преобразуется в формат каскада.

1. Обновите инструкции `using`. Обратите внимание на то, что этот диалог включает диалог Formflow из версии 3. В версии 4 мы применим библиотеку Formflow, созданную сообществом.

```

using Bot.Builder.Community.Dialogs.FormFlow;
using ContosoHelpdeskChatBot.Models;
using Microsoft.Bot.Builder.Dialogs;
using System;
using System.Threading;
using System.Threading.Tasks;

```

2. Добавьте конструктор и инициализируйте набор диалогов для компонентного диалога. Диалог

Formflow создается аналогичным образом. Мы просто добавляем его в набор диалогов для компонентного диалога в конструкторе.

```
public ResetPasswordDialog()
    : base(nameof(ResetPasswordDialog))
{
    InitialDialogId = nameof(WaterfallDialog);
    AddDialog(new WaterfallDialog(nameof(WaterfallDialog), new WaterfallStep[]
    {
        BeginFormflowAsync,
        ProcessRequestAsync,
    }));
    AddDialog(FormDialog.FromForm(BuildResetPasswordForm, FormOptions.PromptInStart));
}
```

3. `startAsync` можно заменить первым шагом каскадного диалога. Мы уже создали Formflow в конструкторе. Для всего остального мы сохраняем прежнюю логику, а вызовы версии 3 просто преобразуем в эквиваленты для версии 4.

```
private async Task<DialogTurnResult> BeginFormflowAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    await stepContext.Context.SendActivityAsync("Alright I will help you create a temp password.");

    // Check the passcode and fail out or begin the Formflow dialog.
    if (SendPassCode(stepContext))
    {
        return await stepContext.BeginDialogAsync(
            nameof(ResetPasswordPrompt),
            cancellationToken: cancellationToken);
    }
    else
    {
        //here we can simply fail the current dialog because we have root dialog handling all
        exceptions
        throw new Exception("Failed to send SMS. Make sure email & phone number has been added to
database.");
    }
}
```

4. `sendPassCode` используется в основном для примера. Мы закомментировали весь исходный код, и теперь этот метод просто возвращает значение true. Также мы можем удалить адрес электронной почты, так как он не использовался в старой версии бота.

```

private bool SendPassCode(DialogContext context)
{
    //bool result = false;

    //Recipient Id varies depending on channel
    //refer ChannelAccount class https://docs.botframework.com/en-
    us/csharp/sdkreference/dd/def/class_microsoft_1_1_bot_1_1_connector_1_1_channel_account.html#
    a0b89cf01fdd73cbc00a524dce9e2ad1a
    //as well as Activity class https://docs.botframework.com/en-
    us/csharp/sdkreference/dc/d2f/class_microsoft_1_1_bot_1_1_connector_1_1_activity.html
    //int passcode = new Random().Next(1000, 9999);
    //Int64? smsNumber = 0;
    //string smsMessage = "Your Contoso Pass Code is ";
    //string countryDialPrefix = "+1";

    // TODO: save PassCode to database
    //using (var db = new ContosoHelpdeskContext())
    //{
    //    var reset = db.ResetPasswords.Where(r => r.EmailAddress == email).ToList();
    //    if (reset.Count >= 1)
    //    {
    //        reset.First().PassCode = passcode;
    //        smsNumber = reset.First().MobileNumber;
    //        result = true;
    //    }
    //

    //    db.SaveChanges();
    //}

    // TODO: send passcode to user via SMS.
    //if (result)
    //{
    //    result = Helper.SendSms($"{countryDialPrefix}{smsNumber.ToString()}", $"{smsMessage}{passcode}");
    //}

    //return result;
    return true;
}

```

5. `BuildResetPasswordForm` сохраняется в неизменном виде.

6. Мы можем заменить `ResumeAfterResetPasswordFormDialog` вторым шагом каскадного диалога, а возвращаемое значение теперь будем получать из контекста шага. Мы удалили адрес электронной почты, с которым исходный диалог не выполнял никаких действий, и вместо запроса к базе данных возвращаем фиктивный результат. Мы полностью сохраняем прежнюю логику, а вызовы версии 3 преобразуем в эквиваленты для версии 4.

```

private async Task<DialogTurnResult> ProcessRequestAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    // Get the result from the Formflow dialog when it ends.
    if (stepContext.Reason != DialogReason.CancelCalled)
    {
        var prompt = stepContext.Result as ResetPasswordPrompt;
        int? passcode;

        // TODO: Retrieve the passcode from the database.
        passcode = 1111;

        if (prompt.PassCode == passcode)
        {
            string temppwd = "TempPwd" + new Random().Next(0, 5000);
            await stepContext.Context.SendActivityAsync(
                $"Your temp password is {temppwd}",
                cancellationToken: cancellationToken);
        }
    }

    return await stepContext.EndDialogAsync(null, cancellationToken);
}

```

## Обновление моделей при необходимости

Нам нужно обновить инструкции `using` в некоторых моделях, которые ссылаются на библиотеку Formflow.

- В `LocalAdminPrompt` поместите вместо них этот код:

```
using Bot.Builder.Community.Dialogs.FormFlow;
```

- В `ResetPasswordPrompt` поместите вместо них этот код:

```
using Bot.Builder.Community.Dialogs.FormFlow;
using System;
```

## Обновление файла Web.config

Закомментируйте ключи конфигурации для `MicrosoftAppId` и `MicrosoftAppPassword`. Это позволит вам выполнять локальную отладку бота, не указывая эти значения в эмуляторе.

## Запуск и тестирование бота в эмуляторе

На этом этапе мы уже можем запустить бота локально в IIS и подключиться к эмулятору.

- Запустите бот в IIS.
- Запустите эмулятор и подключитесь к конечной точке Bot (например, `http://localhost:3978/api/messages`).
  - Если бот запускается впервые, щелкните **Файл > Новая программа-робот** и следуйте инструкциям на экране. В противном случае, чтобы открыть существующий бот, щелкните **Файл > Открыть программу-робота**.
  - Внимательно проверьте параметры порта в конфигурации. Например, если в браузере бот открывается по адресу `http://localhost:3979/`, укажите в эмуляторе для бота конечную точку `http://localhost:3979/api/messages`.

3. Все четыре диалога должны работать. Вы можете установить точки останова в нужных шагах каскада, чтобы проверить контекст и состояние диалога в этих точках.

## Дополнительные ресурсы

Тематические статьи по версии 4:

- [Принципы работы бота](#)
- [Управление состоянием](#)
- [Библиотека диалогов](#)

Пошаговые инструкции для версии 4:

- [Отправка и получение текстовых сообщений](#)
- [Сохранение данных пользователя и диалога](#)
- [Реализация процесса общения](#)
- [Отладка с помощью эмулятора](#)
- [Добавление данных телеметрии в бот](#)

# Перенос бота .NET версии 3 в бот .NET Core версии 4

27.03.2021 • 39 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье описывается, как преобразовать бот [ContosoHelpdeskChatBot версии 3](#) в бот версии 4 в новом проекте .NET Core. Преобразование включает в себя следующие действия:

1. создание проекта на основе шаблона;
2. установка дополнительных пакетов NuGet при необходимости;
3. настройка бота согласно требованиям, обновление файла Startup.cs, а затем класса контроллера;
4. обновление класса бота;
5. копирование и обновление диалогов и моделей;
6. и наконец, перенос данных.

В результате этого преобразования вы получите [бот ContosoHelpdeskChatBot в .NET Core версии 4](#).

Дополнительные сведения о том, как перенести бот в .NET Framework версии 4 *без преобразования типа проекта* см. [здесь](#).

Пакет SDK версии 4 для Bot Framework использует тот же базовый REST API, что и пакет SDK версии 3. Но в пакете SDK версии 4 выполнен рефакторинг кода предыдущей версии пакета SDK для повышения гибкости и улучшения контроля над ботами. Основные изменения в пакете SDK:

- Управление состоянием осуществляется через соответствующие объекты и методы доступа к свойствам.
- Также изменились процессы настройки обработчика шагов и передачи ему действий.
- Элементов с возможностью оценки больше нет. Вы можете проверить наличие глобальных команд в обработчике шагов, прежде чем передавать средства управления в диалоги.
- Новая библиотека Dialogs, которая сильно отличается от предыдущей версии. Вам потребуется преобразовать все старые диалоги в новую систему, используя компонентные, каскадные диалоги и созданную сообществом реализацию диалогов Formflow для версии 4.

См. дополнительные сведения об [отличиях между версиями 3 и 4 пакета SDK для .NET](#).

## Создание проекта на основе шаблона

### NOTE

Пакет [VSIX](#) включает версии .net Core 2.1 и .net Core 3.1 шаблонов C#. При создании ботов в Visual Studio 2019 следует использовать шаблоны .NET Core 3.1. В текущих примерах ботов используются шаблоны .NET Core 3.1. Примеры, использующие шаблоны .NET Core 2.1, можно найти в ветви [4.7-archive](#) репозитория BotBuilder-Samples. Сведения о развертывании .NET Core 3.1 программы-роботы в Azure см. в статье [развертывание программы Bot в Azure](#).

1. Установите [шаблон для C#](#) пакета SDK Bot Framework версии 4, если вы этого еще не сделали.
2. Откройте Visual Studio и создайте на основе шаблона проект для бота Echo. Назовите проект [ContosoHelpdeskChatBot](#).

## Установка дополнительных пакетов NuGet

С шаблоном устанавливаются большинство необходимых пакетов, в том числе `Microsoft.Bot.Builder` и `Microsoft.Bot.Connector`.

1. Добавьте `Bot.Builder.Community.Dialogs.Formflow`.

Это библиотека сообщества для сборки диалогов версии 4 из файлов определения Formflow версии 3. Одно из зависимостей для нее является `Microsoft.Bot.Builder.Dialogs`, поэтому она тоже автоматически устанавливается.

2. Добавьте `log4net`, чтобы обеспечить поддержку ведения журналов.

## Настройка бота согласно требованиям

1. Переименуйте файл бота `Bots\EchoBot.cs` на `Bots\DialogBot.cs`, а класс `EchoBot` — на `DialogBot`.
2. Переименуйте контроллер `Controllers\BotController.cs` на `Controllers\MessagesController.cs`, а класс `BotController` — на `MessagesController`.

## Обновление файла Startup.cs

Для бота изменились способ управления состоянием и получения входящих действий. Теперь для версии 4 нужно самостоятельно настроить элементы инфраструктуры [управления состоянием](#). К примеру, в версии 4 применяется адаптер ботов для проверки подлинности и передачи действий в код бота, а это означает, что все свойства состояния нужно объявить заранее.

Мы создадим свойство состояния для `DialogState`, который потребуется в версии 4 для поддержки диалогов. Мы будем использовать внедрение зависимостей, чтобы получить информацию, которая нужна контроллеру и боту.

В `Startup.cs`:

1. Обновите инструкции `using`.

```
using ContosoHelpdeskChatBot.Bots;
using ContosoHelpdeskChatBot.Dialogs;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.BotFramework;
using Microsoft.Bot.Builder.Integration.AspNet.Core;
using Microsoft.Bot.Connector.Authentication;
using Microsoft.Extensions.DependencyInjection;
```

2. Удалите этот конструктор:

```
public Startup(IConfiguration configuration)
{
    Configuration = configuration;
}
```

3. Удалите свойство `Configuration`.

4. Вместо метода `ConfigureServices` сохраните следующий код:

```

// This method gets called by the runtime. Use this method to add services to the container.
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

    // Create the credential provider to be used with the Bot Framework Adapter.
    services.AddSingleton<ICredentialProvider, ConfigurationCredentialProvider>();

    // Create the Bot Framework Adapter with error handling enabled.
    services.AddSingleton<IBotFrameworkHttpAdapter, AdapterWithErrorHandler>();

    // Create the storage we'll be using for User and Conversation state. (Memory is great for
    // testing purposes.)
    services.AddSingleton<IStorage, MemoryStorage>();

    // Create the Conversation state. (Used by the Dialog system itself.)
    services.AddSingleton<ConversationState>();

    // Create the Root Dialog as a singleton
    services.AddSingleton<RootDialog>();

    // Create the bot as a transient. In this case the ASP Controller is expecting an IBot.
    services.AddTransient<IBot, DialogBot<RootDialog>>();
}

```

Возможно, на этом этапе возникнут ошибки времени компиляции. Мы исправим их на следующих шагах.

## Класс MessagesController

Этот класс обрабатывает запрос. Внедрение зависимостей обеспечивает получение данных адаптера и реализацию IBot в среде выполнения. Этот класс шаблона останется без изменений.

В этой части кода в версии 4 бот запускает шаг, что существенно отличается от того, как это делает контроллер сообщений версии 3. Все остальное, кроме обработчика шагов бота, можно считать стандартным кодом.

Обработчик шагов бота будет определен в файле `Bots\DialogBot.cs`.

### Игнорирование классов CancelScorable и GlobalMessageHandlersBotModul

Так как в версии 4 больше нет элементов с возможностью оценки, эти классы следует просто игнорировать. Мы обновим обработчик шагов так, чтобы бот реагировал на сообщение `cancel`.

## Обновление класса бота

В версии 4 логика цикла обработки сообщений или обработчика шагов находится преимущественно в файле бота. Мы получаем данные от `ActivityHandler`, который определяет обработчики для распространенных типов действий.

1. Обновите файл `Bots\DialogBots.cs`.

2. Обновите инструкции `using`.

```

using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Schema;
using System.Threading;
using System.Threading.Tasks;

```

3. Обновите `DialogBot`, добавив универсальный параметр для диалога.

```
public class DialogBot<T> : ActivityHandler where T : Dialog
```

4. Добавьте эти поля и конструктор для инициализации. В ASP.NET также используется внедрение зависимостей, чтобы получить значения параметров.

```
protected readonly Dialog _dialog;
protected readonly BotState _conversationState;

public DialogBot(ConversationState conversationState, T dialog)
{
    _conversationState = conversationState;
    _dialog = dialog;
}
```

5. Измените внедрение `OnMessageActivityAsync` так, чтобы оно вызывало главный диалог. (Мы определим метод расширения `Run` чуть позже.)

```
protected override async Task OnMessageActivityAsync(
    ITurnContext<IMessageActivity> turnContext,
    CancellationToken cancellationToken)
{
    // Run the Dialog with the new message Activity.
    await _dialog.Run(
        turnContext,
        _conversationState.CreateProperty<DialogState>("DialogState"),
        cancellationToken);
}
```

1. Обновите `OnTurnAsync`, чтобы сохранять состояние беседы в конце шага. В версии 4 нам нужно сделать это явным образом, чтобы записать состояние на уровне сохраняемости. Метод `ActivityHandler.OnTurnAsync` вызывает методы конкретного обработчика действий с учетом типа полученного действия. Поэтому мы сохраним состояние после вызова базового метода.

```
public override async Task OnTurnAsync(ITurnContext turnContext, CancellationToken cancellationToken)
{
    await base.OnTurnAsync(turnContext, cancellationToken);

    // Save any state changes that might have occurred during the turn.
    await _conversationState.SaveChangesAsync(turnContext, false, cancellationToken);
}
```

### Создание метода расширения запуска

Мы создаем метод расширения, чтобы объединить код для запуска простого компонентного диалога из бота.

Создайте файл `DialogExtensions.cs` и реализуйте метод расширения `Run`.

```

using System.Threading;
using System.Threading.Tasks;
using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.Dialogs;

namespace ContosoHelpdeskChatBot
{
    public static class DialogExtensions
    {
        public static async Task Run(this Dialog dialog, ITurnContext turnContext,
IStatePropertyAccessor<DialogState> accessor, CancellationToken cancellationToken =
default(CancellationToken))
        {
            var dialogSet = new DialogSet(accessor);
            dialogSet.Add(dialog);

            var dialogContext = await dialogSet.CreateContextAsync(turnContext, cancellationToken);
            var results = await dialogContext.ContinueDialogAsync(cancellationToken);
            if (results.Status == DialogTurnStatus.Empty)
            {
                await dialogContext.BeginDialogAsync(dialog.Id, null, cancellationToken);
            }
        }
    }
}

```

## Копирование и преобразование диалогов

Мы будем вносить изменения в исходные диалоги версии 3, что позволит использовать их с пакетом SDK версии 4. Пока не беспокойтесь об ошибках компилятора. Они исчезнут, когда преобразование завершится. Мы не будем изменять исходный код больше, чем необходимо, поэтому некоторые предупреждения компилятора сохранятся после завершения переноса.

Все диалоги теперь будут не реализацией интерфейса `IDialog<object>` из версии 3, а станут производными от `ComponentDialog`.

Этот бот использует четыре диалога, и все их нужно преобразовать:

ДИАЛОГ	ОПИСАНИЕ
<code>RootDialog</code>	Представляет варианты на выбор и запускает другие диалоги.
<code>InstallAppDialog</code>	Обрабатывает запросы на установку приложения на компьютер.
<code>LocalAdminDialog</code>	Обрабатывает запросы на получение прав локального администратора на компьютере.
<code>ResetPasswordDialog</code>	Обрабатывает запросы на сброс пароля.

Мы не будем копировать класс `CancelScorable`, так как больше нет элементов с возможностью оценки. Вы можете проверить наличие *глобальных* команд в обработчике шагов, прежде чем передавать средства управления в диалоги.

Все они собирают данные, но не выполняют никаких операций на компьютере.

1. Создайте папку `Dialogs` (Диалоги) в проекте.
2. Скопируйте эти файлы из каталога с диалогами проекта версии 3 в новый каталог:

- `InstallAppDialog.cs`;
- `LocalAdminDialog.cs`;
- `ResetPasswordDialog.cs`;
- `RootDialog.cs`.

### Внесение изменений в диалоги на уровне решения

1. Для всего решения измените все вхождения `IDialog<object>` на `ComponentDialog`.
2. Для всего решения измените все вхождения `IDialogContext` на `DialogContext`.
3. Для каждого класса диалогов удалите атрибут `[Serializable]`.

Поток управления и обмен сообщениями в диалогах теперь обрабатываются другими способами, и это нужно учитывать при преобразовании каждого диалога.

ОПЕРАЦИЯ	КОД ДЛЯ ВЕРСИИ 3	КОД ДЛЯ ВЕРСИИ 4
Обработка запуска диалога	Реализуйте <code>IDialog.StartAsync</code>	Сделайте этот шаг первым в каскадном диалоге или реализуйте <code>Dialog.BeginDialogAsync</code>
Обработка продолжения диалога	Вызовите <code>IDialogContext.Wait</code>	Добавьте в каскадный диалог дополнительные шаги или реализуйте <code>Dialog.ContinueDialogAsync</code>
Отправка сообщения пользователю	Вызовите <code>IDialogContext.PostAsync</code>	Вызовите <code>ITurnContext.SendActivityAsync</code>
Запуск дочернего диалога	Вызовите <code>IDialogContext.Call</code>	Вызовите <code>DialogContext.BeginDialogAsync</code>
Информирование о завершении текущего диалога	Вызовите <code>IDialogContext.Done</code>	Вызовите <code>DialogContext.EndDialogAsync</code>
Получение введенных пользователем данных	Используйте параметр <code>IAwaitable&lt;IMessageActivity&gt;</code>	Используйте приглашение в каскадном диалоге или <code>ITurnContext.Activity</code>

Примечания о коде для версии 4:

- В коде диалога используйте свойство `DialogContext.Context`, чтобы получить контекст текущего шага.
- Каскадные шаги имеют параметр `WaterfallStepContext`, который является производным от `DialogContext`.
- Все конкретные классы диалогов и приглашений наследуют от абстрактного класса `Dialog`.
- Идентификатор присваивается при создании компонентного диалога. Каждый диалог в наборе диалогов должен иметь уникальный в пределах этого набора идентификатор.

### Обновление корневого диалога

В этом боте корневой диалог предлагает пользователю выбрать один вариант из набора, а затем на основе этого выбора запускает дочерний диалог. Этот цикл повторяется неограниченно долго, пока существует беседа.

- Основной поток можно создать в формате каскадного диалога — это новая концепция в пакете SDK версии 4. Он выполняет фиксированный набор шагов в заданном порядке. Подробнее см. статью [Реализация последовательной беседы](#).

- Запросы теперь обрабатываются через классы prompt. Эти короткие дочерние диалоги предлагают ввести данные, выполняют минимальную обработку и проверку этих данных и возвращают полученное значение. Дополнительные сведения о сборе данных от пользователя с помощью запросов диалога см. в [этой статье](#).

В файле Dialogs/RootDialog.cs сделайте следующее:

1. Обновите инструкции `using`.

```
using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Builder.Dialogs.Choices;
using System;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;
```

2. Нам нужно преобразовать варианты `HelpdeskOptions` из списка строк в список вариантов. Они будут применяться в строке запроса, которая в качестве допустимых входных данных принимает номер выбранного варианта в списке, значение любого из вариантов или его синоним.

```
private static List<Choice> HelpdeskOptions = new List<Choice>()
{
    new Choice(InstallAppOption) { Synonyms = new List<string> { "install" } },
    new Choice(ResetPasswordOption) { Synonyms = new List<string> { "password" } },
    new Choice(LocalAdminOption) { Synonyms = new List<string> { "admin" } }
};
```

3. Добавьте конструктор. Этот код делает следующее:

- Каждому экземпляру диалога при создании присваивается идентификатор. Идентификатор диалога входит в набор диалогов, к которому добавляется этот диалог. Не забывайте, что бот уже инициализирован с использованием объекта диалога в классе `MessageController`. Для всех `ComponentDialog` есть свой внутренний набор диалогов с отдельным набором идентификаторов диалогов.
- Включает остальные диалоги, в том числе запросы выбора, в качестве дочерних диалогов. Здесь мы используем для каждого диалога в качестве идентификатора соответствующее имя класса.
- Определяет каскадный диалог с тремя шагами. Их мы реализуем чуть позже.
  - Сначала диалог предлагает пользователю выбрать задание для выполнения.
  - Затем он запускает дочерний диалог, связанный с выбранным вариантом.
  - И уже потом он перезапускает сам себя.
- Каждый шаг каскадного диалога является делегатом. Мы реализуем их далее, по возможности сохраняя существующий код каждого исходного диалога.
- При запуске компонентного диалога будет запущен *начальный диалог*. По умолчанию это первый дочерний диалог, добавленный в компонентный диалог. Мы явным образом определим свойство `InitialDialogId`. Это значит, что нам не нужно добавлять основной каскадный диалог первым в набор диалогов. Например, вы можете сначала добавить запросы, и это не вызовет никаких проблем во время выполнения.

```

public RootDialog()
    : base(nameof(RootDialog))
{
    InitialDialogId = nameof(WaterfallDialog);
    AddDialog(new WaterfallDialog(nameof(WaterfallDialog), new WaterfallStep[]
    {
        PromptForOptionsAsync,
        ShowChildDialogAsync,
        ResumeAfterAsync,
    }));
    AddDialog(new InstallAppDialog());
    AddDialog(new LocalAdminDialog());
    AddDialog(new ResetPasswordDialog());
    AddDialog(new ChoicePrompt(nameof(ChoicePrompt)));
}

```

4. Метод `StartAsync` можно удалить. При запуске компонентный диалог автоматически выполняет свой *начальный* диалог. В нашем примере это тот каскадный диалог, который мы определяем в конструкторе. Он автоматически начинает выполнение с первого действия.

5. Мы удалим методы `MessageReceivedAsync` и `ShowOptions`, а затем заменим их первым шагом нашего каскадного диалога. Эти два метода ранее приветствовали пользователя и предлагали выбор из доступных вариантов.

- Здесь вы видите список значений, приветствия и сообщения об ошибках, которые передаются в вызов запроса на выбор в качестве параметров.
- Нам не нужно указывать следующий метод, который будет вызван в диалоге, так как каскадный диалог автоматически перейдет к следующему шагу после завершения запроса на выбор.
- Запрос выбора повторяется в цикле, пока не получит допустимые входные данные или весь стек диалогов не будет отменен.

```

private async Task<DialogTurnResult> PromptForOptionsAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    // Prompt the user for a response using our choice prompt.
    return await stepContext.PromptAsync(
        nameof(ChoicePrompt),
        new PromptOptions()
    {
        Choices = HelpdeskOptions,
        Prompt = MessageFactory.Text(GreetMessage),
        RetryPrompt = MessageFactory.Text(ErrorMessage)
    },
    cancellationToken);
}

```

6. `OnOptionSelected` можно заменить вторым шагом каскадного диалога. Здесь мы также запускаем дочерний диалог в зависимости от ввода пользователя.

- Запрос выбора возвращает значение `FoundChoice`. Оно передается в свойстве `Result` этого шага. Стек диалогов обрабатывает все возвращаемые значения как объекты. Если это значение возвращается из вашего диалога, вы получите тип значения для этого объекта. Список значений, возвращаемых разными типами, вы можете изучить [здесь](#).
- Так как запрос выбора не создает исключение, мы можем удалить блок `try-catch`.
- Но нам нужно добавить механизм передачи управления, чтобы этот метод всегда возвращал допустимое значение. Этот код никогда не должен выполняться, но если это произойдет из-за сбоя, он корректно завершит работу диалога.

```

private async Task<DialogTurnResult> ShowChildDialogAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    // string optionSelected = await userReply;
    var optionSelected = (stepContext.Result as FoundChoice).Value;

    switch (optionSelected)
    {
        case InstallAppOption:
            //context.Call(new InstallAppDialog(), this.ResumeAfterOptionDialog);
            //break;
            return await stepContext.BeginDialogAsync(
                nameof(InstallAppDialog),
                cancellationToken);
        case ResetPasswordOption:
            //context.Call(new ResetPasswordDialog(), this.ResumeAfterOptionDialog);
            //break;
            return await stepContext.BeginDialogAsync(
                nameof(ResetPasswordDialog),
                cancellationToken);
        case LocalAdminOption:
            //context.Call(new LocalAdminDialog(), this.ResumeAfterOptionDialog);
            //break;
            return await stepContext.BeginDialogAsync(
                nameof(LocalAdminDialog),
                cancellationToken);
    }

    // We shouldn't get here, but fail gracefully if we do.
    await stepContext.Context.SendActivityAsync(
        "I don't recognize that option.",
        cancellationToken: cancellationToken);
    // Continue through to the next step without starting a child dialog.
    return await stepContext.NextAsync(cancellationToken: cancellationToken);
}

```

7. Наконец, замените старый метод `ResumeAfterOptionDialog` последним шагом каскадного диалога.

- В исходном диалоге мы завершали работу и возвращали номер билета. Теперь же мы перезапускаем каскадный диалог, заменяя его старый экземпляр в стеке диалогов новым экземпляром того же диалога. Это стало возможным, так как исходное приложение всегда игнорирует возвращаемое значение (номер билета) и перезапускает корневой диалог.

```

private async Task<DialogTurnResult> ResumeAfterAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    try
    {
        //var message = await userReply;
        var message = stepContext.Context.Activity;

        var ticketNumber = new Random().Next(0, 20000);
        //await context.PostAsync($"Thank you for using the Helpdesk Bot. Your ticket number is {ticketNumber}.");
        await stepContext.Context.SendActivityAsync(
            $"Thank you for using the Helpdesk Bot. Your ticket number is {ticketNumber}.",
            cancellationToken: cancellationToken);

        //context.Done(ticketNumber);
    }
    catch (Exception ex)
    {
        // await context.PostAsync($"Failed with message: {ex.Message}");
        await stepContext.Context.SendActivityAsync(
            $"Failed with message: {ex.Message}",
            cancellationToken: cancellationToken);

        // In general resume from task after calling a child dialog is a good place to handle
        exceptions
        // try catch will capture exceptions from the bot framework awaitable object which is
        essentially "userReply"
        logger.Error(ex);
    }

    // Replace on the stack the current instance of the waterfall with a new instance,
    // and start from the top.
    return await stepContext.ReplaceDialogAsync(
        nameof(WaterfallDialog),
        cancellationToken: cancellationToken);
}

```

## Обновление диалога установки приложения

Диалог установки приложения выполняет несколько логических задач, которые мы сейчас настроим в четырех шагах каскадного диалога. Перенос существующего кода в каскадные шаги будет отдельной логической задачей для каждого из диалогов. Для каждого шага указан исходный метод, из которого взят его код.

1. Запрос на ввод строки поиска пользователем.
2. Запрос возможных совпадений в базе данных.
  - Если есть одно совпадение, выбор этого значения и продолжение работы.
  - Если совпадений несколько, запрос на выбор одного варианта пользователем.
  - Если совпадений нет, выход из диалога.
3. Запрос на выбор компьютера, где пользователем будет установлено приложение.
4. Сохранение сведений в базе данных и отправка сообщения с подтверждением.

В файле `Dialogs/InstallAppDialog.cs` сделайте следующее:

1. Обновите инструкции `using`.

```
using ContosoHelpdeskChatBot.Models;
using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Builder.Dialogs.Choices;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
```

2. Определите константу для ключа, чтобы отслеживать собранные сведения.

```
// Set up keys for managing collected information.
private const string InstallInfo = "installInfo";
```

3. Добавьте конструктор и инициализируйте набор диалогов для компонентного диалога.

```
public InstallAppDialog()
    : base(nameof(InstallAppDialog))
{
    // Initialize our dialogs and prompts.
    InitialDialogId = nameof(WaterfallDialog);
    AddDialog(new WaterfallDialog(nameof(WaterfallDialog), new WaterfallStep[] {
        GetSearchTermAsync,
        ResolveAppNameAsync,
        GetMachineNameAsync,
        SubmitRequestAsync,
    }));
    AddDialog(new TextPrompt(nameof(TextPrompt)));
    AddDialog(new ChoicePrompt(nameof(ChoicePrompt)));
}
```

4. `StartAsync` можно заменить первым шагом каскадного диалога.

- Но нам придется самостоятельно управлять состоянием, для которого мы применим объект установки приложения в состоянии диалога.
- Сообщение с запросом пользовательских данных становится необязательным параметром в вызове запроса.

```
private async Task<DialogTurnResult> GetSearchTermAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    // Create an object in dialog state in which to track our collected information.
    stepContext.Values[InstallInfo] = new InstallApp();

    // Ask for the search term.
    return await stepContext.PromptAsync(
        nameof(TextPrompt),
        new PromptOptions
        {
            Prompt = MessageFactory.Text("Ok let's get started. What is the name of the application?"),
        },
        cancellationToken);
}
```

5. `appNameAsync` И `multipleAppsAsync` можно заменить вторым шагом каскадного диалога.

- Теперь мы получаем результат запроса, а не просто проверяем последнее сообщение пользователя.

- Запрос к базе данных и инструкции if организованы здесь так же, как в `appNameAsync`. Код в каждом блоке инструкции if обновлен для поддержки диалогов версии 4.
  - Если обнаружится только одно совпадение, мы обновляем состояние диалога и переходим к следующему шагу.
  - Если совпадений несколько, мы применяем запрос выбора и предлагаем пользователю выбрать вариант из списка. Это означает, что можно просто удалить `multipleAppsAsync`.
  - Если совпадений нет, мы завершаем диалог и возвращаем значение NULL в корневой диалог.

```

private async Task<DialogTurnResult> ResolveAppNameAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    // Get the result from the text prompt.
    var appname = stepContext.Result as string;

    // Query the database for matches.
    var names = await this.GetAppsAsync(appname);

    if (names.Count == 1)
    {
        // Get our tracking information from dialog state and add the app name.
        var install = stepContext.Values[InstallInfo] as InstallApp;
        install.AppName = names.First();

        return await stepContext.NextAsync();
    }
    else if (names.Count > 1)
    {
        // Ask the user to choose from the list of matches.
        return await stepContext.PromptAsync(
            nameof(ChoicePrompt),
            new PromptOptions
            {
                Prompt = MessageFactory.Text("I found the following applications. Please choose one:"),
                Choices = ChoiceFactory.ToChoices(names),
            },
            cancellationToken);
    }
    else
    {
        // If no matches, exit this dialog.
        await stepContext.Context.SendActivityAsync(
            $"Sorry, I did not find any application with the name '{appname}'.",
            cancellationToken: cancellationToken);

        return await stepContext.EndDialogAsync(null, cancellationToken);
    }
}

```

6. Метод `appNameAsync` также запрашивает у пользователя имя компьютера после обработки запроса. Мы реализуем эту часть логики в следующем шаге каскадного диалога.

- Не забывайте, что в версии 4 нам нужно самостоятельно управлять состоянием. Единственной сложностью будет то, что на этот шаг с предыдущего мы можем попасть через две разные ветви логики.
- Для запроса имени компьютера у пользователя мы применим тот же запрос текста, что и ранее, но с другими вариантами ответов.

```

private async Task<DialogTurnResult> GetMachineNameAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    // Get the tracking info. If we don't already have an app name,
    // Then we used the choice prompt to get it in the previous step.
    var install = stepContext.Values[InstallInfo] as InstallApp;
    if (install.AppName is null)
    {
        install.AppName = (stepContext.Result as FoundChoice).Value;
    }

    // We now need the machine name, so prompt for it.
    return await stepContext.PromptAsync(
        nameof(TextPrompt),
        new PromptOptions
        {
            Prompt = MessageFactory.Text(
                $"Found {install.AppName}. What is the name of the machine to install application?"),
            cancellationToken);
}

```

7. Логика из метода `machineNameAsync` попадает в заключительный шаг каскадного диалога.

- Мы извлекаем имя компьютера из результата, возвращенного запросом текста, и обновляем состояние диалога.
- Вызов для обновления базы данных мы удалим, так как вспомогательный код размещается в другом проекте.
- Затем мы отправляем пользователю сообщение об успешном выполнении и заканчиваем диалог.

```

private async Task<DialogTurnResult> SubmitRequestAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    var install = default(InstallApp);
    if (stepContext.Reason != DialogReason.CancelCalled)
    {
        // Get the tracking info and add the machine name.
        install = stepContext.Values[InstallInfo] as InstallApp;
        install.MachineName = stepContext.Context.Activity.Text;

        //TODO: Save to this information to the database.
    }

    await stepContext.Context.SendActivityAsync(
        $"Great, your request to install {install.AppName} on {install.MachineName} has been
scheduled.",
        cancellationToken: cancellationToken);

    return await stepContext.EndDialogAsync(null, cancellationToken);
}

```

8. Чтобы сымитировать вызов базы данных, мы создадим вызов с использованием `getAppsAsync`, что позволит обратиться к статическому списку, а не реальной базе данных.

```
private async Task<List<string>> GetAppsAsync(string Name)
{
    var names = new List<string>();

    // Simulate querying the database for applications that match.
    return (from app in AppMsis
            where app.ToLower().Contains(Name.ToLower())
            select app).ToList();
}

// Example list of app names in the database.
private static readonly List<string> AppMsis = new List<string>
{
    "μTorrent 3.5.0.44178",
    "7-Zip 17.1",
    "Ad-Aware 9.0",
    "Adobe AIR 2.5.1.17730",
    "Adobe Flash Player (IE) 28.0.0.105",
    "Adobe Flash Player (Non-IE) 27.0.0.130",
    "Adobe Reader 11.0.14",
    "Adobe Shockwave Player 12.3.1.201",
    "Advanced SystemCare Personal 11.0.3",
    "Auslogics Disk Defrag 3.6",
    "avast! 4 Home Edition 4.8.1351",
    "AVG Anti-Virus Free Edition 9.0.0.698",
    "Bonjour 3.1.0.1",
    "CCleaner 5.24.5839",
    "Chmod Calculator 20132.4",
    "CyberLink PowerDVD 17.0.2101.62",
    "DAEMON Tools Lite 4.46.1.328",
    "FileZilla Client 3.5",
    "Firefox 57.0",
    "Foxit Reader 4.1.1.805",
    "Google Chrome 66.143.49260",
    "Google Earth 7.3.0.3832",
    "Google Toolbar (IE) 7.5.8231.2252",
    "GSpot 2701.0",
    "Internet Explorer 903235.0",
    "iTunes 12.7.0.166",
    "Java Runtime Environment 6 Update 17",
    "K-Lite Codec Pack 12.1",
    "Malwarebytes Anti-Malware 2.2.1.1043",
    "Media Player Classic 6.4.9.0",
    "Microsoft Silverlight 5.1.50907",
    "Mozilla Thunderbird 57.0",
    "Nero Burning ROM 19.1.1005",
    "OpenOffice.org 3.1.1 Build 9420",
    "Opera 12.18.1873",
    "Paint.NET 4.0.19",
    "Picasa 3.9.141.259",
    "QuickTime 7.79.80.95",
    "RealPlayer SP 12.0.0.319",
    "Revo Uninstaller 1.95",
    "Skype 7.40.151",
    "Spybot - Search & Destroy 1.6.2.46",
    "SpywareBlaster 4.6",
    "TuneUp Utilities 2009 14.0.1000.353",
    "Unlocker 1.9.2",
    "VLC media player 1.1.6",
    "Winamp 5.56 Build 2512",
    "Windows Live Messenger 2009 16.4.3528.331",
    "WinPatrol 2010 31.0.2014",
    "WinRAR 5.0",
};
};
```

В версии 3 этот диалог приветствовал пользователя, запускал диалог Formflow и сохранял результат в базе данных. Эти действия легко преобразовать в каскадный диалог с двумя шагами.

1. Обновите инструкции `using`. Обратите внимание на то, что этот диалог включает диалог Formflow из версии 3. В версии 4 мы применим библиотеку Formflow, созданную сообществом.

```
using Bot.Builder.Community.Dialogs.FormFlow;
using ContosoHelpdeskChatBot.Models;
using Microsoft.Bot.Builder.Dialogs;
using System.Threading;
using System.Threading.Tasks;
```

2. Мы можем удалить свойство экземпляра для `LocalAdmin`, так как результат будет доступен в состоянии диалога.
3. Добавьте конструктор и инициализируйте набор диалогов для компонентного диалога. Диалог Formflow создается аналогичным образом. Мы просто добавляем его в набор диалогов для компонентного диалога в конструкторе.

```
public LocalAdminDialog() : base(nameof(LocalAdminDialog))
{
    InitialDialogId = nameof(WaterfallDialog);
    AddDialog(new WaterfallDialog(nameof(WaterfallDialog), new WaterfallStep[]
    {
        BeginFormflowAsync,
        SaveResultAsync,
    }));
    AddDialog(FormDialog.FromForm(BuildLocalAdminForm, FormOptions.PromptInStart));
}
```

4. `StartAsync` можно заменить первым шагом каскадного диалога. Мы уже создали Formflow в конструкторе, а остальные две инструкции будут преобразованы соответствующим образом. Обратите внимание, что `FormBuilder` присваивает имя типа модели в качестве идентификатора созданного диалога (`LocalAdminPrompt` для этой модели).

```
private async Task<DialogTurnResult> BeginFormflowAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    await stepContext.Context.SendActivityAsync("Great I will help you request local machine
admin.");

    // Begin the Formflow dialog.
    return await stepContext.BeginDialogAsync(
        nameof(LocalAdminPrompt),
        cancellationToken: cancellationToken);
}
```

5. `ResumeAfterLocalAdminFormDialog` можно заменить вторым шагом каскадного диалога. Нам нужно получать возвращаемое значение из контекста шага, а не свойства экземпляра.

```

private async Task<DialogTurnResult> SaveResultAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    // Get the result from the Formflow dialog when it ends.
    if (stepContext.Reason != DialogReason.CancelCalled)
    {
        var admin = stepContext.Result as LocalAdminPrompt;

        //TODO: Save to this information to the database.
    }

    return await stepContext.EndDialogAsync(null, cancellationToken);
}

```

6. `BuildLocalAdminForm` почти не изменяется, за исключением того, что теперь FormFlow не будет обновлять свойство экземпляра.

```

// Nearly the same as before.
private IForm<LocalAdminPrompt> BuildLocalAdminForm()
{
    // Here's an example of how validation can be used with FormBuilder.
    return new FormBuilder<LocalAdminPrompt>()
        .Field(nameof(LocalAdminPrompt.MachineName),
        validate: async (state, value) =>
    {
        var result = new ValidateResult { IsValid = true, Value = value };
        //add validation here

        //this.admin.MachineName = (string)value;
        return result;
    })
        .Field(nameof(LocalAdminPrompt.AdminDuration),
        validate: async (state, value) =>
    {
        var result = new ValidateResult { IsValid = true, Value = value };
        //add validation here

        //this.admin.AdminDuration = Convert.ToInt32((long)value) as int?;
        return result;
    })
        .Build();
}

```

### Обновление диалога смены пароля

В версии 3 этот диалог приветствует пользователя, авторизует пользователя по коду доступа, возвращает ошибку авторизации или запускает диалог Formflow и сбрасывает значение пароля. Все это неплохо преобразуется в формат каскада.

1. Обновите инструкции `using`. Обратите внимание на то, что этот диалог включает диалог Formflow из версии 3. В версии 4 мы применим библиотеку Formflow, созданную сообществом.

```

using Bot.Builder.Community.Dialogs.FormFlow;
using ContosoHelpdeskChatBot.Models;
using Microsoft.Bot.Builder.Dialogs;
using System;
using System.Threading;
using System.Threading.Tasks;

```

2. Добавьте конструктор и инициализируйте набор диалогов для компонентного диалога. Диалог

Formflow создается аналогичным образом. Мы просто добавляем его в набор диалогов для компонентного диалога в конструкторе.

```
public ResetPasswordDialog()
    : base(nameof(ResetPasswordDialog))
{
    InitialDialogId = nameof(WaterfallDialog);
    AddDialog(new WaterfallDialog(nameof(WaterfallDialog), new WaterfallStep[]
    {
        BeginFormflowAsync,
        ProcessRequestAsync,
    }));
    AddDialog(FormDialog.FromForm(BuildResetPasswordForm, FormOptions.PromptInStart));
}
```

3. `startAsync` можно заменить первым шагом каскадного диалога. Мы уже создали Formflow в конструкторе. Для всего остального мы сохраняем прежнюю логику, а вызовы версии 3 просто преобразуем в эквиваленты для версии 4.

```
private async Task<DialogTurnResult> BeginFormflowAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    await stepContext.Context.SendActivityAsync("Alright I will help you create a temp password.");

    // Check the passcode and fail out or begin the Formflow dialog.
    if (SendPassCode(stepContext))
    {
        return await stepContext.BeginDialogAsync(
            nameof(ResetPasswordPrompt),
            cancellationToken: cancellationToken);
    }
    else
    {
        //here we can simply fail the current dialog because we have root dialog handling all
        exceptions
        throw new Exception("Failed to send SMS. Make sure email & phone number has been added to
database.");
    }
}
```

4. `sendPassCode` используется в основном для примера. Мы закомментировали весь исходный код, и теперь этот метод просто возвращает значение true. Также мы можем удалить адрес электронной почты, так как он не использовался в старой версии бота.

```

private bool SendPassCode(DialogContext context)
{
    //bool result = false;

    //Recipient Id varies depending on channel
    //refer ChannelAccount class 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
    //as well as Activity class https://docs.botframework.com/en-
    us/csharp/builder/sdkreference/dc/d2f/class_microsoft_1_1_bot_1_1_connector_1_1_activity.html
    //int passcode = new Random().Next(1000, 9999);
    //Int64? smsNumber = 0;
    //string smsMessage = "Your Contoso Pass Code is ";
    //string countryDialPrefix = "+1";

    // TODO: save PassCode to database
    //using (var db = new ContosoHelpdeskContext())
    //{
    //    var reset = db.ResetPasswords.Where(r => r.EmailAddress == email).ToList();
    //    if (reset.Count >= 1)
    //    {
    //        reset.First().PassCode = passcode;
    //        smsNumber = reset.First().MobileNumber;
    //        result = true;
    //    }
    //

    //    db.SaveChanges();
    //}

    // TODO: send passcode to user via SMS.
    //if (result)
    //{
    //    result = Helper.SendSms($"{countryDialPrefix}{smsNumber.ToString()}", $"{smsMessage}{passcode}");
    //}

    //return result;
    return true;
}

```

5. `BuildResetPasswordForm` сохраняется в неизменном виде.

6. Мы можем заменить `ResumeAfterResetPasswordFormDialog` вторым шагом каскадного диалога, а возвращаемое значение теперь будем получать из контекста шага. Мы удалили адрес электронной почты, с которым исходный диалог не выполнял никаких действий, и вместо запроса к базе данных возвращаем фиктивный результат. Мы полностью сохраняем прежнюю логику, а вызовы версии 3 преобразуем в эквиваленты для версии 4.

```

private async Task<DialogTurnResult> ProcessRequestAsync(
    WaterfallStepContext stepContext,
    CancellationToken cancellationToken = default(CancellationToken))
{
    // Get the result from the Formflow dialog when it ends.
    if (stepContext.Result != null)
    {
        var prompt = stepContext.Result as ResetPasswordPrompt;
        int? passcode;

        // TODO: Retrieve the passcode from the database.
        passcode = 1111;

        if (prompt.PassCode == passcode)
        {
            string temppwd = "TempPwd" + new Random().Next(0, 5000);
            await stepContext.Context.SendActivityAsync(
                $"Your temp password is {temppwd}",
                cancellationToken: cancellationToken);
        }
    }

    return await stepContext.EndDialogAsync(null, cancellationToken);
}

```

## Копирование и обновление моделей при необходимости

Вы можете использовать модели версии 3 в библиотеке сообщества FormFlow версии 4.

1. Создайте папку **Models** (Модели) в проекте.
2. Скопируйте эти файлы из каталога с моделями проекта версии 3 в новый каталог:
  - **InstallApp.cs**;
  - **LocalAdmin.cs**;
  - **LocalAdminPrompt.cs**;
  - **ResetPassword.cs**;
  - **ResetPasswordPrompt.cs**.

### Обновление инструкций using

В классах моделей необходимо обновить инструкции `using` как показано ниже.

1. В **InstallApp.cs** измените их так:

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

```

2. В **LocalAdmin.cs** измените их так:

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

```

3. В **LocalAdminPrompt.cs** измените их так:

```

using Bot.Builder.Community.Dialogs.FormFlow;

```

4. В **ResetPassword.cs** измените их так:

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
```

Кроме того, удалите инструкции `using` в пространстве имен.

5. В `ResetPasswordPrompt.cs` измените их так:

```
using Bot.Builder.Community.Dialogs.FormFlow;
using System;
```

#### Дополнительные изменения

В `ResetPassword.cs` измените тип возвращаемого значения `MobileNumber` так:

```
public long? MobileNumber { get; set; }
```

## Конечный перенос данных

Чтобы завершить процесс переноса, сделайте следующее:

1. Создайте класс `AdapterWithErrorHandler`, чтобы определить адаптер, что предусматривает обработчик ошибок, перехватывающий исключения в программном обеспечении промежуточного слоя и приложениях. Адаптер обрабатывает и направляет входящие действия через конвейер по промежуточного слоя в логику Bot, а затем обратно. Используйте код ниже, чтобы создать класс:

```

using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.Integration.AspNet.Core;
using Microsoft.Bot.Connector.Authentication;
using System;

namespace ContosoHelpdeskChatBot
{
    public class AdapterWithErrorHandler : BotFrameworkHttpAdapter
    {
        private static log4net.ILog logger
            = log4net.LogManager.GetLogger(System.Reflection.MethodBase.GetCurrentMethod().DeclaringType);

        public AdapterWithErrorHandler(
            ICredentialProvider credentialProvider,
            ConversationState conversationState = null)
            : base(credentialProvider)
        {
            OnTurnError = async (turnContext, exception) =>
            {
                // Log any leaked exception from the application.
                logger.Error($"Exception caught : {exception.Message}");

                // Send a catch-all apology to the user.
                await turnContext.SendActivityAsync("Sorry, it looks like something went wrong.");

                if (conversationState != null)
                {
                    try
                    {
                        // Delete the conversationState for the current conversation to prevent the
                        // bot from getting stuck in a error-loop caused by being in a bad state.
                        // ConversationState should be thought of as similar to "cookie-state" in a Web
pages.
                        await conversationState.DeleteAsync(turnContext);
                    }
                    catch (Exception e)
                    {
                        logger.Error($"Exception caught on attempting to Delete ConversationState :
{e.Message}");
                    }
                }
            };
        }
    }
}

```

1. Измените страницу `wwwroot\default.htm` согласно требованиям.

## Запуск и тестирование бота в эмуляторе

На этом этапе мы уже можем запустить бота локально в IIS и подключить его к эмулятору.

1. Запустите бот в IIS.
2. Запустите эмулятор и подключитесь к конечной точке Bot (например, <http://localhost:3978/api/messages>).
  - Если бот запускается впервые, щелкните **Файл > Новая программа-робот** и следуйте инструкциям на экране. В противном случае, чтобы открыть существующий бот, щелкните **Файл > Открыть программу-робота**.
  - Внимательно проверьте параметры порта в конфигурации. Например, если в браузере бот открывается по адресу `http://localhost:3979/`, укажите в эмуляторе для бота конечную точку `http://localhost:3979/api/messages`.
3. Все четыре диалога должны работать. Вы можете установить точки останова в нужных шагах каскада,

чтобы проверить контекст и состояние диалога в этих точках.

## Дополнительные ресурсы

Тематические статьи по версии 4:

- [Принципы работы бота](#)
- [Управление состоянием](#)
- [Библиотека диалогов](#)

Пошаговые инструкции для версии 4:

- [Отправка и получение текстовых сообщений](#)
- [Сохранение данных пользователя и диалога](#)
- [Реализация процесса общения](#)
- [Отладка с помощью эмулятора](#)
- [Добавление данных телеметрии в бот](#)

# Использование данных .NET о состоянии пользователя версии 3 в боте версии 4

27.03.2021 • 5 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье описано, как бот версии 4 может выполнять с информацией о состоянии пользователя версии 3 операции чтения, записи и удаления. Бот поддерживает состояние беседы с помощью `MemoryStorage` для ее отслеживания, управления ее ходом и отправки вопросов пользователю. Он сохраняет **состояние пользователя** в формате версии 3 для отслеживания ответов пользователя с помощью настраиваемого класса `IStorage` с именем `V3V4Storage`. Одним из аргументов этого класса является `IBotDataStore`. База кода пакета SDK версии 3 была скопирована в `Bot.Builder.Azure.V3V4` и содержит всех три поставщика хранилищ из пакета SDK версии 3 (Azure SQL, таблицы Azure и Cosmos DB). Это нужно, чтобы разрешить внедрение существующего **состояния пользователя** версии 3 в перенесенного бота версии 4.

Пример кода можно найти в [поставщиках с кодом версии 4 \(Bot\)](#).

## Предварительные требования

- [Пакет SDK для .NET Core](#) версии 2.1.

```
# determine dotnet version
dotnet --version
```

## Настройка

### 1. Клонирование репозитория

```
git clone https://github.com/microsoft/botbuilder-samples.git
```

### 2. В окне терминала перейдите в папку `MigrationV3V4/CSharp/V4StateBotFromV3Providers`.

```
cd BotBuilder-Samples/MigrationV3V4/CSharp/V4StateBotFromV3Providers
```

### 3. Запустите бота из окна терминала или из Visual Studio и выберите вариант A или B.

- В окне терминала

```
# run the bot
dotnet run
```

- Или из Visual Studio

- Откройте Visual Studio и выберите "Файл" -> "Открыть" -> "Проект или решение".
- Перейдите в папку `BotBuilder-Samples/MigrationV3V4/CSharp/V4StateBotFromV3Providers`.
- Выберите файл `V4StateBot.sln`.
- Нажмите клавишу F5, чтобы запустить проект.

## Настройка поставщика хранилища

Предполагается, что вам доступно настроенное и используемое существующее хранилище состояний версии 3. В таком случае для подготовки этого примера с использованием существующего хранилища состояний просто нужно добавить сведения о подключениях поставщика хранилища в файл `web.config`, как показано далее.

- Cosmos DB

```
"v3CosmosEndpoint": "https://yourcosmosdb.documents.azure.com:443/",  
"v3CosmosKey": "YourCosmosDbKey",  
"v3CosmosDatabaseName": "v3botdb",  
"v3CosmosCollectionName": "v3botcollection",
```

- Azure SQL

```
"ConnectionStrings": {  
    "SqlBotData": "Server=YourServer;Initial Catalog=BotData;Persist Security Info=False;User  
ID=YourUserName;Password=YourUserPassword;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate  
=True;Connection Timeout=30;"  
},
```

- таблице Azure

```
"ConnectionStrings": {  
    "AzureTable":  
    "DefaultEndpointsProtocol=https;AccountName=YourAccountName;AccountKey=YourAccountKey;EndpointSuffix=core.wi  
ndows.net"  
},
```

- Укажите поставщика хранилища для бота.

Откройте файл `Startup.cs` в корне проекта `V4V3StateBot`. В середине файла (приблизительно строки 52–76) хранятся настройки для каждого поставщика хранилищ. Они считывают значения конфигурации из файла `web.config`.

```

Uri docDbEmulatorUri = new Uri(Configuration["v3CosmosEndpoint"]);
var documentDbBotDataStore = new DocumentDbBotDataStore(docDbEmulatorUri,
    Configuration["v3CosmosKey"],
    databaseId: Configuration["v3CosmosDatabaseName"],
    collectionId: Configuration["v3CosmosCollectionName"]);

// SqlBotDataStore for V3V4 User State
//var sqlConnectionString = Configuration.GetConnectionString("SqlBotData");
//var sqlBotDataStore = new SqlBotDataStore(sqlConnectionString);

// TableBotDataStore for V3V4 User State
//var tableConnectionString = Configuration.GetConnectionString("AzureTable");
//var tableBotDataStore = new TableBotDataStore(tableConnectionString);

// TableBotDataStore2 for V3V4 User State
//var tableBotDataStore2 = new TableBotDataStore2(tableConnectionString);

// Create the V3V4Storage layer bridge, providing a V3 storage.
// Then use that storage to create a V3V4State, and inject as a singleton
var v3v4Storage = new V3V4Storage(documentDbBotDataStore);
//var v3v4Storage = new V3V4Storage(sqlBotDataStore);
//var v3v4Storage = new V3V4Storage(tableBotDataStore);
//var v3v4Storage = new V3V4Storage(tableBotDataStore2);
services.AddSingleton<V3V4State>(new V3V4State(v3v4Storage));

```

Укажите поставщика хранилищ, который будет использоваться ботом, раскомментировав соответствующие строки для выбранного экземпляра. После настройки поставщика убедитесь, что его класс передается `V3V4Storage` (приблизительно строки 72–75).

```

var v3v4Storage = new V3V4Storage(documentDbBotDataStore);
//var v3v4Storage = new V3V4Storage(sqlBotDataStore);
//var v3v4Storage = new V3V4Storage(tableBotDataStore);
//var v3v4Storage = new V3V4Storage(tableBotDataStore2);

```

По умолчанию задается Cosmos DB (ранее — Document DB). Вы можете выбрать

```

documentDbBotDataStore
tableBotDataStore
tableBotDataStore2
sqlBotDataStore

```

- Запустите приложение.

## V3V4 Storage и классы состояния

### **V3V4Storage**

Класс `V3V4Storage` содержит основные функции сопоставления хранилищ. Он реализует интерфейс `IStorage` версии 4 и сопоставляет методы поставщика хранилищ ( чтение, запись и удаление ) с классами поставщика хранилищ версии 3, чтобы состояние пользователя в формате версии 3 можно было использовать в боте версии 4.

### **V3V4State**

Этот класс наследуется от класса `BotState` версии 4 и использует ключ в стиле версии 3 ( `IAddress` ). Это позволяет выполнять операции чтения, записи и удаления с хранилищем версии 3 аналогично тому, как они реализованы для хранилища состояний версии 3.

## Тестирование бота с помощью Bot Framework Emulator

Bot Framework Emulator — это классическое приложение, которое позволяет разработчикам бота локально или удаленно (через туннель) тестировать боты или выполнять их отладку.

- Установите эмулятор Bot Framework [версии 4.3.0 или более поздней](#).

### Подключение к боту с помощью Bot Framework Emulator

- Запустите Bot Framework Emulator.
- Выберите "Файл" -> "Открыть бота".
- Введите URL-адрес бота `http://localhost:3978/api/messages`.

## Дополнительные материалы

- [Общие сведения о службе Azure Bot](#)
- [Состояние бота](#)
- [Непосредственная запись в хранилище](#)
- [Управление состоянием беседы и пользователя](#)

# Различия между версиями 3 и 4 пакета SDK для JavaScript

27.03.2021 • 17 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Версия 4 пакета SDK Bot Framework поддерживает ту же базовую службу Bot Framework, что и версия 3. Но в версии 4 выполнен рефакторинг кода предыдущей версии пакета SDK для повышения гибкости и улучшения контроля над ботами. Основные изменения в пакете SDK:

- Общие сведения об адаптере бота.
  - Является компонентом стека обработки действий.
  - Выполняет аутентификацию Bot Framework.
  - Управляет входящим и исходящим трафиком между каналом и обработчиком шагов вашего бота, инкапсулируя вызовы в Bot Framework Connector.
  - Инициализирует контекст для каждого шага.
  - Дополнительные сведения см. в статье [Принципы работы бота](#).
- Выполнен рефакторинг управления состоянием.
  - Данные о состоянии больше не предоставляются в боте автоматически.
  - Управление состоянием теперь осуществляется через соответствующие объекты и методы доступа к свойствам.
  - Дополнительные сведения см. в статье [Управление состоянием](#).
- Новая библиотека диалогов.
  - Диалоги версии 3 необходимо переписать для новой библиотеки диалогов.
  - Диалоги с возможностью оценки Scoreable больше не существуют. Вы можете проверить наличие глобальных команд, прежде чем передавать средства управления в диалоги. В зависимости от архитектуры бота версии 4 это можно сделать в обработчике сообщений или в родительском диалоге. Пример см. в статье [Обработка прерываний со стороны пользователя](#).
  - Дополнительные сведения см. в статье [Библиотека диалогов](#).

## Обработка действий

При создании адаптера для бота вы также указываете делегата обработчика сообщений, который будет получать входящие действия от каналов и пользователей. Адаптер создает объект контекста шага для каждого полученного действия. Он передает объект контекста шага обработчику шагов в боте, а затем удаляет объект после завершения шага.

Обработчик шагов может получать действия разных типов. Как правило, диалогам в боте необходимо перенаправлять только действия *сообщений*. Если вы наследуете бота из `ActivityHandler`, обработчик шагов в нем будет пересыпать все сообщения о действиях в `OnMessage`. Переопределите этот метод, чтобы добавить логику обработки сообщений. Дополнительные сведения о типах действий см. в статье о [схеме действий](#).

### Шаги обработки

При обработке сообщений используйте контекст шага, чтобы получить данные о входящем действии и отправить действия пользователю:

ДЕЙСТВИЕ	ОПИСАНИЕ
Получение входящего действия	Получите свойство <code>Activity</code> контекста шага.
Создание действия и отправка его пользователю	Вызовите метод <code>SendActivity</code> контекста шага. Дополнительные сведения см. в статьях <a href="#">Отправка и получение текстовых сообщений</a> и <a href="#">Добавление мультимедиа в сообщения</a> .

Класс `MessageFactory` предоставляет некоторые вспомогательные методы для создания и форматирования действий.

### Обработка прерываний

Обрабатывайте такие элементы в цикле сообщений бота. Описание такого процесса для диалогов в версии 4 см. в статье [Обработка прерываний со стороны пользователя](#).

## Управление данными о состоянии

В версии 3 вы могли хранить данные о беседе в службе "Состояние бота", которая входит в обширный набор служб для Bot Framework. Но после 31 марта 2018 г. эта служба более недоступна. Начиная с версии 4 применяются те же рекомендации по управлению состоянием, что и для любого веб-приложения. Такое управление можно реализовать несколькими способами. Обычно проще всего кэшировать состояние в памяти в том же процессе, но в производственных приложениях состояние следует сохранять в более устойчивом хранилище, например в базе данных SQL или NoSQL или в больших двоичных объектах.

В версии 4 для управления состоянием не используются свойства `UserData`, `ConversationData` и `PrivateConversationData`, а также контейнеры данных. Управление состоянием теперь осуществляется через соответствующие объекты и методы доступа к свойствам, как описано в статье [Управление состоянием](#).

Версия 4 определяет классы `UserState`, `ConversationState` и `PrivateConversationState`, которые управляют данными состояния для бота. Вам нужно создать метод доступа к свойству состояния для каждого свойства, которое нужно сохранить (а не просто считывать и записывать его в предопределенный контейнер данных).

### Настройка состояния

Состояние должно быть настроено в файле точки входа приложения. В приложениях Node.js это обычно файл `index.js` или `app.js`.

- Инициализируйте один или несколько объектов, реализующих интерфейс `Storage`, предоставляемый `botbuilder-core`. Вы получите резервное хранилище для данных вашего бота. Пакет SDK версии 4 предоставляет несколько [уровней хранения](#). Вы также можете реализовать собственные уровни, чтобы подключаться к хранилищам различных типов.
- Затем при необходимости создайте и зарегистрируйте объекты [управления состоянием](#). Вам доступны те же области, что и в версии 3, и при необходимости вы можете создать другие.
- Наконец, создайте и зарегистрируйте [методы доступа к свойствам состояния](#) для нужных боту свойств. В объекте управления состоянием каждый метод доступа к свойству должен иметь уникальное имя.

### Использование состояния

Используйте методы доступа к свойствам состояния, чтобы получить и обновить свои свойства, а объекты управления состоянием, чтобы записать какие-либо изменения в хранилище. Так как вам необходимо учитывать особенности параллелизма, ниже описано, как выполнять некоторые

стандартные задачи.

ЗАДАЧА	ОПИСАНИЕ
Создание метода доступа к свойству состояния	Вызовите метод <code>createProperty</code> объекта <code>BotState</code> . <code>BotState</code> является абстрактным базовым классом для беседы, закрытой беседы и состояния пользователя.
Получение текущего значения свойства	Вызовите процедуру <code>StatePropertyAccessor.get(TurnContext)</code> . Если значение не задано ранее, для его создания будет использоваться фабричный параметр по умолчанию.
Сохранение изменений состояний в хранилище	Вызовите <code>BotState.saveChanges(TurnContext, boolean)</code> для любого объекта управления состоянием, в котором состояние было изменено до выхода из обработчика шагов.

### Управление параллелизмом

Возможно, вашему боту потребуется управлять параллелизмом состояния. Дополнительные сведения см. в разделе [Сохранение состояния](#) статьи [Управление состоянием](#) и в разделе [Управление параллелизмом с помощью тегов eTag](#) статьи [Запись данных напрямую в хранилище](#).

## Библиотека диалогов

Ниже приведены некоторые из основных изменений в диалогах:

- Библиотека диалогов теперь имеет вид отдельного пакета npm с именем `botbuilder-dialogs`.
- Управление состоянием диалога осуществляется через метод доступа к свойству и классу состояния `DialogState`.
  - Свойство состояния диалога теперь сохраняется между шагами в отличие от самого объекта диалога.
- В ходе выполнения шага вы создаете контекст диалога для *набора диалогов*.
  - Этот контекст диалога инкапсулирует стек диалога. Эта информация сохраняется в свойстве состояния диалога.
- Обе версии содержат абстрактный класс `Dialog`, но в версии 3 он расширяет класс `ActionSet`, тогда как в версии 4 он расширяет `Object`.

### Определение диалогов

Версия 3 предоставляла гибкую возможность реализовать диалоги с помощью класса `Dialog`. Но для этого требовалось создать собственный код для таких функций, как проверка. В версии 4 появились классы запроса, которые автоматически проверяют пользовательский ввод, применяют к нему ограничение по типу (например целое число) и повторно запрашивают данные у пользователя, пока они не будут соответствовать этим ограничениям. В большинстве случаев это позволяет разработчику писать меньше кода.

Теперь вам доступно несколько способов определения диалогов:

ТИП ДИАЛОГА	ОПИСАНИЕ
Компонентный диалог, наследуемый от класса <code>ComponentDialog</code> .	Позволяет инкапсулировать код диалога без конфликтов именования с другими контекстами. См. статью <a href="#">Повторное использование диалогов</a> .

ТИП ДИАЛОГА	ОПИСАНИЕ
Каскадный диалог, экземпляр класса <code>WaterfallDialog</code> .	Разработан для оптимальной работы с диалогами запроса, которые предлагают пользователю ввести данные и проверяют их. Каскадный диалог автоматизирует большинство процессов, но требует применения определенной формы для кода диалога (см. статью <a href="#">Реализация процесса общения</a> ).
Настраиваемый диалог, полученный из абстрактного класса <code>Dialog</code> .	Дает максимальную гибкость в аспекте поведения диалогов, но также требует более глубоких знаний о реализации стека диалогов.

Создавая каскадный диалог, вы определяете шаги диалога в конструкторе. Порядок выполняемых действий строго соответствует тому, в котором вы их объявили, и переход между ними происходит автоматически.

Можно также создать сложные потоки управления с помощью нескольких диалогов, как описано в статье [Создание сложного потока беседы с использованием ветвлений и циклов](#).

Чтобы получить доступ к диалогу, вам необходимо разместить его экземпляр в *наборе диалогов*, а затем сгенерировать *контекст диалога* для такого набора. При создании набора диалогов вам нужно указать метод доступа к свойству состояния диалога. Это позволит платформе сохранять состояние диалога при переходе между шагами. Управление состоянием в версии 4 описано в статье [Управление состоянием](#).

### Использование диалогов

Ниже приведен список стандартных операций в версии 3, а также описывается их выполнение в каскадном диалоге. Обратите внимание, что каждый шаг каскадного диалога должен возвращать значение `DialogTurnResult`. В противном случае каскадный диалог может преждевременно завершиться.

ОПЕРАЦИЯ	ВЕРСИЯ 3	ВЕРСИЯ 4
Обработка запуска диалога	Вызовите <code>session.beginDialog</code> , передав идентификатор диалога.	Позвони <code>DialogContext.beginDialog</code>
Отправка действия	Вызовите процедуру <code>session.send</code> .	Вызовите процедуру <code>TurnContext.sendActivity</code> . Используйте свойство <code>Context</code> контекста шага, чтобы получить контекст шага ( <code>step.context.sendActivity</code> ).
Ожидание ответа пользователя	Вызовите запрос внутри шага каскадного диалога, например <code>builder.Prompts.text(session, 'Please enter your destination')</code> . Получите ответ на следующем шаге.	Верните ожидание <code>TurnContext.prompt</code> , чтобы начать диалог с запросом. Затем получите результаты на следующем шаге каскадного диалога.
Обработка продолжения диалога	Автоматически	Добавьте в каскадный диалог дополнительные шаги или реализуйте <code>Dialog.continueDialog</code>
Обозначение завершения обработки до следующего сообщения пользователя	Вызовите процедуру <code>session.endDialog</code> .	Возвращается значение <code>Dialog.EndOfTurn</code> .

ОПЕРАЦИЯ	ВЕРСИЯ 3	ВЕРСИЯ 4
Запуск дочернего диалога	Вызовите процедуру <code>session.beginDialog</code> .	Верните ожидание для метода <code>beginDialog</code> контекста шага. Если дочерний диалог вернет значение, оно будет доступно на следующем шаге каскадного диалога с помощью свойства <code>Result</code> контекста шага.
Замена текущего диалога новым	Вызовите процедуру <code>session.replaceDialog</code> .	Верните ожидание для <code>ITurnContext.replaceDialog</code> .
Информирование о завершении текущего диалога	Вызовите процедуру <code>session.endDialog</code> .	Верните ожидание для метода <code>endDialog</code> контекста шага.
Выход из диалога	Вызовите процедуру <code>session.pruneDialogStack</code> .	Сгенерируйте исключение, которое будет зарегистрировано на другом уровне бота, завершите шаг с состоянием <code>Cancelled</code> либо вызовите шаг или <code>cancelAllDialogs</code> контекста диалога.

Другие примечания о коде версии 4:

- Различные производные классы `Prompt` в версии 4 реализуют запросы пользователей в виде отдельных диалогов из двух шагов. См. сведения о том, [как реализовать последовательный поток беседы] [последовательный поток].
- Используйте `DialogSet.createContext`, чтобы создать контекст диалога для текущего шага.
- Используйте свойство `DialogContext.context` из диалога, чтобы получить контекст текущего шага.
- Каскадные шаги имеют параметр `WaterfallStepContext`, который является производным от `DialogContext`.
- Все конкретные классы диалогов и приглашений наследуют от абстрактного класса `Dialog`.
- Идентификатор присваивается при создании компонентного диалога. Каждый диалог в наборе диалогов должен иметь уникальный в пределах этого набора идентификатор.

#### Передача состояния в диалогах и между ними

В разделе [состояние диалоговой](#) статьи **Библиотека диалоговых окон** и в разделах [свойства контекста каскада](#) и [параметры диалоговых окон](#) статьи **о компонентах и каскадных диалоговых окнах** описывается управление состоянием диалога в v4.

#### Получение ответа пользователя

Чтобы получить действие пользователя на шаге, получите его из контекста шага.

Чтобы отправить запрос пользователю и получить результат:

- Добавьте соответствующий экземпляр запроса к вашему набору диалогов.
- Вызовите запрос из шага в каскадном диалоге.
- Извлеките результат из свойства `Result` контекста шага на следующем шаге.

## Дополнительные ресурсы

Указанные ниже ресурсы содержат дополнительные и справочные сведения.

<b>РАЗДЕЛ</b>	<b>ОПИСАНИЕ</b>
Перенос бота на основе пакета SDK для JavaScript версии 3 в версию 4	Перенос бота для JavaScript из версии 3 в версию 4
Новые возможности Bot Framework	Основные возможности и усовершенствования Bot Framework и Azure Bot
Принципы работы бота	Внутренний механизм бота
Управление состоянием	Абстракции, упрощающие управление состоянием
Библиотека диалогов	Основные концепции управления диалогом
Отправка и получение текстовых сообщений	Основной способ взаимодействия бота с пользователями
Отправка мультимедиа	Отправка во вложении мультимедийных материалов, таких как изображения, файлы, видео и аудиозаписи
Реализация процесса общения	Постановка вопросов как основной способ взаимодействия бота с пользователями
Сохранение данных пользователя и диалога	Отслеживание диалога без отслеживания состояния
Сложный поток беседы	Управление сложным процессом общения
Повторное использование диалогов	Создание независимых диалогов для особых сценариев
Обработка прерываний	Обработка прерываний для повышения эффективности бота
Схема действий	Схема действий, предпринимаемых в ходе беседы людьми и автоматическим программным обеспечением

# Краткий справочник по миграции для JavaScript

27.03.2021 • 7 minutes to read • [Edit Online](#)

## применимо к: Пакет SDK v4

В пакете SDK Bot Builder для Javascript версии 4 реализовано несколько важных изменений, связанных с процессом создания ботов. В этом руководстве описаны различия между способами выполнения задач в пакетах SDK версий 3 и 4.

- Способ передачи данных между каналами и ботом изменился. В версии 3 вы использовали объекты *соединителя* и *сессии*. В версии 4 они заменены объектами *адаптера* и *контекста шага*.
- Кроме того, разграничены экземпляры диалогов и бота. В версии 3 диалоги регистрировались непосредственно в конструкторе бота. В версии 4 диалоги передаются в экземпляры бота как аргументы, делая процесс создания более гибким.
- Кроме того, в версии 4 реализован класс `ActivityHandler` для автоматизации обработки разных типов действий, таких как *сообщение*, *обновление беседы* и *событие*.

В результате изменен синтаксис для разработки ботов на Javascript. В частности это касается создания объектов бота, определения диалогов и создания логики обработки событий.

Ниже приведено сравнение конструкций в пакетах SDK Bot Framework для JavaScript версий 3 и 4.

## Прослушивание входящих запросов

### Версия 3

```
var connector = new builder.ChatConnector({
  appId: process.env.MicrosoftAppId,
  appPassword: process.env.MicrosoftAppPassword
});

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

### версия 4

```
const adapter = new BotFrameworkAdapter({
  appId: process.env.MicrosoftAppId,
  appPassword: process.env.MicrosoftAppPassword
});

server.post('/api/messages', (req, res) => {
  adapter.processActivity(req, res, async (context) => {
    await bot.run(context);
  });
});
```

## Создание экземпляра бота

### Версия 3

```
var bot = new builder.UniversalBot(connector, [ ...DIALOG_STEPS ]);
```

## **версия 4**

```
// Define the bot class as extending ActivityHandler.  
const { ActivityHandler } = require('botbuilder');  
  
class MyBot extends ActivityHandler {  
    // ...  
}  
  
// Instantiate a new bot instance.  
const bot = new MyBot(conversationState, dialog);
```

## Отправка сообщения пользователю

### **Версия 3**

```
session.send('Hello and welcome to the help desk bot.');
```

### **версия 4**

```
await context.sendActivity('Hello and welcome to the help desk bot.');
```

## Определение каталога по умолчанию

### **Версия 3**

```
var bot = new builder.UniversalBot(connector, [  
    // Add default dialog waterfall steps...  
]);
```

### **версия 4**

```
// In the main dialog class, define a run method.
async run(turnContext, accessor) {
    const dialogSet = new DialogSet(accessor);
    dialogSet.add(this);

    const dialogContext = await dialogSet.createContext(turnContext);
    const results = await dialogContext.continueDialog();
    if (results.status === DialogTurnStatus.empty) {
        await dialogContext.beginDialog(this.id);
    }
}

// Pass conversation state management and a main dialog objects to the bot (in index.js).
const bot = new MyBot(conversationState, dialog);

// Inside the bot's constructor, add the dialog as a member property and define a DialogState property
accessor.
this.dialog = dialog;
this.dialogState = this.conversationState.createProperty('DialogState');

// Inside the bot's message handler, invoke the dialog's run method, passing in the turn context and
DialogState accessor.
this.onMessage(async (context, next) => {
    // Run the Dialog with the new message Activity.
    await this.dialog.run(context, this.dialogState);
    await next();
});
```

## Запуск дочернего диалога

Родительский диалог возобновляется после завершения дочернего диалога.

### Версия 3

```
session.beginDialog(DIALOG_ID);
```

### версия 4

```
await context.beginDialog(DIALOG_ID);
```

## Замена диалога

Текущий диалог в стеке заменяется новым диалогом.

### Версия 3

```
session.replaceDialog(DIALOG_ID);
```

### версия 4

```
await context.replaceDialog(DIALOG_ID);
```

## Завершение диалога

### Версия 3

```
session.endDialog();
```

#### версия 4

Вы можете включить необязательное возвращаемое значение.

```
await context.endDialog(returnValue);
```

## Регистрация диалога с использованием экземпляра бота

#### Версия 3

```
// Create the bot.  
var bot = new builder.UniversalBot(connector, [  
    // ...  
]);  
  
// Add the dialog to the bot.  
bot.dialog('myDialog', require('./mydialog'));
```

#### версия 4

```
// In the main dialog class constructor.  
this.addDialog(new MyDialog(DIALOG_ID));  
  
// In the app entry point file (index.js)  
const { MainDialog } = require('./dialogs/main');  
  
// Create the base dialog and bot, passing the main dialog as an argument.  
const dialog = new MainDialog();  
const reservationBot = new ReservationBot(conversationState, dialog);
```

## Запрос пользователю на ввод данных

#### Версия 3

```
var builder = require('botbuilder');  
builder.Prompts.text(session, 'Please enter your destination');
```

#### версия 4

```
const { TextPrompt } = require('botbuilder-dialogs');  
  
// In the dialog's constructor, register the prompt.  
this.addDialog(new TextPrompt('initialPrompt'));  
  
// In the dialog step, invoke the prompt.  
return await stepContext.prompt(  
    'initialPrompt', {  
        prompt: 'Please enter your destination'  
    }  
);
```

## Получение ответа пользователя на запрос

### **Версия 3**

```
function (session, result) {
    var destination = result.response;
    // ...
}
```

### **версия 4**

```
// In the dialog step after the prompt step, retrieve the result from the waterfall step context.
async nextStep(stepContext) {
    const destination = stepContext.result;
    // ...
}
```

## Сохранение информации в состоянии диалога

### **Версия 3**

```
session.dialogData.destination = results.response;
```

### **версия 4**

```
// In a dialog, set the value in the waterfall step context.
stepContext.values.destination = destination;
```

## Запись изменений состояния в уровень сохраняемости

### **Версия 3**

```
// State data is saved by default at the end of the turn.
// Do this to save it manually.
session.save();
```

### **версия 4**

```
// State changes are not saved automatically at the end of the turn.
await this.conversationState.saveChanges(context, false);
await this.userState.saveChanges(context, false);
```

## Создание и регистрация хранилища состояний

### **Версия 3**

```
var builder = require('botbuilder');

// Create conversation state with in-memory storage provider.
var inMemoryStorage = new builder.MemoryBotStorage();

// Create the base dialog and bot
var bot = new builder.UniversalBot(connector, [ /*...*/ ]).set('storage', inMemoryStorage);
```

### **версия 4**

```
const { MemoryStorage } = require('botbuilder');

// Create the memory layer object.
const memoryStorage = new MemoryStorage();

// Create the conversation state management object, using the storage provider.
const conversationState = new ConversationState(memoryStorage);

// Create the base dialog and bot.
const dialog = new MainDialog();
const reservationBot = new ReservationBot(conversationState, dialog);
```

## Перехват ошибки, связанной с диалогом

### Версия 3

```
// Set up the error handler.
session.on('error', function (err) {
    session.send('Failed with message:', err.message);
    session.endDialog();
});

// Throw an error.
session.error('An error has occurred.');
```

### версия 4

```
// Set up the error handler, using the adapter.
adapter.onTurnError = async (context, error) => {
    const errorMsg = error.message

    // Clear out conversation state to avoid keeping the conversation in a corrupted bot state.
    await conversationState.delete(context);

    // Send a message to the user.
    await context.sendActivity(errorMsg);
};

// Throw an error.
async () => {
    throw new Error('An error has occurred.');
}
```

## Регистрация обработчиков действий

### Версия 3

```
bot.on('conversationUpdate', function (message) {
    // ...
}

bot.on('contactRelationUpdate', function (message) {
    // ...
})
```

### версия 4

```
// In the bot's constructor, add the handlers.  
this.onMessage(async (context, next) => {  
    // ...  
}  
  
this.onDialog(async (context, next) => {  
    // ...  
}  
  
this.onMembersAdded(async (context, next) => {  
    // ...  
}
```

## Отправка вложений

### Версия 3

```
var message = new builder.Message()  
.attachments(hotelHeroCards  
.attachmentLayout(builder.AttachmentLayout.carousel));
```

### версия 4

```
await context.sendActivity({  
    attachments: hotelHeroCards,  
    attachmentLayout: AttachmentLayoutTypes.Carousel  
});
```

## Использование распознавания естественного языка (LUIS)

### Версия 3

```
// The LUIS recognizer was part of the 'botbuilder' library  
var builder = require('botbuilder');  
  
var recognizer = new builder.LuisRecognizer(LUIS_MODEL_URL);  
bot.recognizer(recognizer);
```

### версия 4

```
// The LUIS recognizer is now part of the 'botbuilder-ai' library  
const { LuisRecognizer } = require('botbuilder-ai');  
  
const luisApp = { applicationId: LuisAppId, endpointKey: LuisAPIKey, endpoint: `https://${ LuisAPIHostName }` };  
const recognizer = new LuisRecognizer(luisApp);  
  
const recognizerResult = await recognizer.recognize(context);  
const intent = LuisRecognizer.topIntent(recognizerResult);
```

## Диалоговое окно намерений в версии 3 и эквивалент в версии 4

### Версия 3

```

// Create a 'greetings' RegExpRecognizer that can be turned off
var greetings = new builder.RegExpRecognizer('Greetings', /hello|hi|hey|greetings/i)
    .onEnabled(function (session, callback) {
        // Check to see if this recognizer should be enabled
        if (session.conversationData.useGreetings) {
            callback(null, true);
        } else {
            callback(null, false);
        }
    });
}

// Create our IntentDialog and add recognizers
var intents = new builder.IntentDialog({ recognizers: [greetings] });

bot.dialog('/', intents);

// If no intent is recognized, direct user to Recognizer Menu
intents.onDefault('RecognizerMenu');

// Match our "Greetings" and "Farewell" intents with their dialogs
intents.matches('Greetings', 'Greetings');

// Add a greetings dialog
bot.dialog('Greetings', [
    function (session) {
        session.endDialog('Greetings!');
    }
]);

```

#### версия 4

```

this.onMessage(async (context, next) => {

    const recognizerResult = {
        text: context.activity.text,
        intents: []
    };

    const greetingRegex = RegExp(/hello|hi|hey|greetings/i);

    if (greetingRegex.test(context.activity.text)) {
        // greeting intent identified
        recognizerResult.intents.push('Greeting');
    }

    if (recognizerResult.intents.includes('Greeting')) {
        // Run the 'Greeting' dialog
        await context.beginDialog(GREETING_DIALOG_ID);
    }

    await next();
});

```

# Перенос бота JavaScript версии 3 в бот версии 4

27.03.2021 • 21 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье вы узнаете, как перенести пакет SDK версии 3 для языка JavaScript с несколькими [диалогами – 3](#)-я Bot на новый робот v4 JavaScript. Это преобразование включает следующие этапы:

1. создание нового проекта и добавление зависимостей;
2. обновление точки входа и определение констант;
3. Создайте диалоговые окна и повторно реализуйте их с помощью пакета SDK v4.
4. обновление кода бота для вызова диалогов;
5. перенос вспомогательного файла `store.js`.

В конце этого процесса у вас будет рабочий робот v4. Копию преобразованного бота также можно получить из репозитория с примерами [core-MultiDialogs-v4](#).

Пакет SDK Bot Framework версии 4 использует тот же базовый REST API, что и пакет SDK версии 3. Но в пакете SDK версии 4 выполнен рефакторинг кода предыдущей версии пакета SDK для повышения гибкости и улучшения контроля над ботами. Основные изменения в пакете SDK:

- Управление состоянием осуществляется через соответствующие объекты и методы доступа к свойствам.
- Способ обработки изменений изменился, то есть как Bot получает и реагирует на входящее действие из канала пользователя.
- В версии 4 вместо объекта `session` используется объект *контекста шага*, который содержит сведения о входящих действиях и позволяет отправлять пользователю ответные действия.
- Новая библиотека диалогов существенно отличается от той, которая была в версии 3. Вам потребуется преобразовать все старые диалоги в новую систему, используя компонентные и каскадные диалоги.

## NOTE

В рамках миграции необходимо также очистить часть кода. В этой статье описываются изменения, которые необходимо внести в логику v3 в рамках процесса миграции.

## Предварительные требования

- Node.js
- Visual Studio Code
- [Bot Framework Emulator](#).

## Сведения о боте

На работе, который вы переносите, демонстрируется использование нескольких диалоговых окон для управления потоком диалога. Этот бот умеет искать сведения о рейсах и отелях.

- Диалог `main` спрашивает у пользователя, какие сведения ему нужны.
- Диалог отелей предлагает пользователю параметры для поиска и имитирует процесс поиска.
- Диалог рейсов создает сообщение об ошибке, которое бот перехватывает и правильно обрабатывает.

## Создание и открытие нового проекта для бота версии 4

- Для переноса кода бота вам потребуется проект версии 4. Чтобы создать такой проект на локальном компьютере, см. руководство по [созданию бота с помощью пакета SDK Bot Framework для JavaScript](#).

### TIP

Вы также можете создать проект в Azure с помощью руководства по [созданию бота с помощью службы Azure Bot](#). Созданные с помощью этих методов вспомогательные файлы немного отличаются. Для этой статьи мы создали локальный проект версии 4.

- Откройте проект в Visual Studio Code.

## Обновление файла package.json

- Добавьте зависимость от пакета `botbuilder-dialogs`, введя команду `npm i botbuilder-dialogs` в окне терминала Visual Studio Code.
- Измените файл `./package.json` и обновите `name`, `version`, `description` и другие свойства.

## Обновление точки входа приложения версии 4

Шаблон версии 4 создает файл `index.js` для точки входа приложения и файл `bot.js` для логики бота. В последующих шагах файл `bot.js` будет переименован в `программы-роботы/reservationBot.js` на более позднем шаге и добавлен класс для каждого диалогового окна.

Измените файл `./index.js`, который является точкой входа для приложения бота. В нем сохранится часть кода из файла `app.js` версии 3, где настраивается сервер HTTP.

- Кроме `BotFrameworkAdapter`, импортируйте `MemoryStorage` и `ConversationState` из пакета `botbuilder`. Также импортируйте модули диалогов `bot` и `main`. (Вы создадите их в ближайшее время, но вам потребуется ссылка на них.)

```
// Import required bot services.  
// See https://aka.ms/bot-services to learn more about the different parts of a bot.  
const { BotFrameworkAdapter, MemoryStorage, ConversationState } = require('botbuilder');  
  
// This bot's main dialog.  
const { MainDialog } = require('./dialogs/main')  
const { ReservationBot } = require('./bots/reservationBot');
```

- Определите обработчик `onTurnError` для адаптера.

```
// Catch-all for errors.  
adapter.onTurnError = async (context, error) => {  
    const errorMsg = error.message ? error.message : `Oops. Something went wrong!`;  
    // This check writes out errors to console log .vs. app insights.  
    console.error(`\n [onTurnError]: ${error}`);  
    // Clear out state  
    await conversationState.delete(context);  
    // Send a message to the user  
    await context.sendActivity(errorMsg);  
};
```

В версии 4 для направления входящих действий в Bot используется *адаптер Bot*. Этот адаптер позволяет перехватывать ошибки и реагировать на них до завершения шага. В этом случае вы очищаете состояние беседы, если возникает ошибка приложения, которая сбрасывает все диалоговые окна и не сохраняет сообщение Bot в поврежденном состоянии диалога.

- Замените код шаблона для создаваемого бота следующим кодом.

```
// Define state store for your bot.  
const memoryStorage = new MemoryStorage();  
  
// Create conversation state with in-memory storage provider.  
const conversationState = new ConversationState(memoryStorage);  
  
// Create the base dialog and bot  
const dialog = new MainDialog();  
const reservationBot = new ReservationBot(conversationState, dialog);
```

Уровень хранения в памяти теперь предоставляемый `MemoryStorage` классом, и необходимо явно создать объект управления состоянием диалога.

Код определения диалогового окна был перемещен в `MainDialog` класс, который предстоит определить в ближайшее время. Вы также перенесете код определения Bot в `ReservationBot` класс.

- Наконец, вы обновляете обработчик запросов сервера, чтобы использовать адаптер для маршрутизации действий в Bot.

```
// Listen for incoming requests.  
server.post('/api/messages', (req, res) => {  
    adapter.processActivity(req, res, async (context) => {  
        // Route incoming activities to the bot.  
        await reservationBot.run(context);  
    });  
});
```

В версии 4 Bot является производным от `ActivityHandler`, который определяет `run` метод получения действия для включения.

## Добавление файла констант

Создайте файл `./const.js` для хранения идентификаторов для программы Bot.

```
module.exports = {  
    MAIN_DIALOG: 'mainDialog',  
    INITIAL_PROMPT: 'initialPrompt',  
    HOTELS_DIALOG: 'hotelsDialog',  
    INITIAL_HOTEL_PROMPT: 'initialHotelPrompt',  
    CHECKIN_DATETIME_PROMPT: 'checkinTimePrompt',  
    HOW_MANY_NIGHTS_PROMPT: 'howManyNightsPrompt',  
    FLIGHTS_DIALOG: 'flightsDialog',  
};
```

В версии 4 объектам диалогов и запросов присваиваются идентификаторы, по которым они затем вызываются.

## Создание файлов диалогов

Создайте описанные ниже файлы.

ИМЯ ФАЙЛА	ОПИСАНИЕ
./dialogs/flights.js	Здесь будет содержаться логика диалога <code>hotels</code> , перенесенная из прежней версии.
./dialogs/hotels.js	Здесь будет содержаться логика диалога <code>flights</code> , перенесенная из прежней версии.
./dialogs/main.js	Здесь будет содержаться логика бота, перенесенная из прежней версии. Этот файл заменяет старый диалог <code>root</code> .

Мы не переносим диалог поддержки. Пример реализации диалога справки в версии 4 см. в руководстве по [обработке прерываний диалога пользователем](#).

### Реализация диалога main

В версии 3 все программы-роботы построены поверх системы диалоговых окон. В версии 4 логика бота и логика диалогов наконец разделены. Вы предоставили *корневое диалоговое окно* в Bot-роботе v3 и сделали `MainDialog` класс, чтобы сделать его место.

Измените файл `./dialogs/main.js`.

1. Импортируйте классы и константы, необходимые для диалогового окна.

```
const { DialogSet, DialogTurnStatus, ComponentDialog, WaterfallDialog,
    ChoicePrompt } = require('botbuilder-dialogs');
const { FlightDialog } = require('./flights');
const { HotelsDialog } = require('./hotels');
const { MAIN_DIALOG,
    INITIAL_PROMPT,
    HOTELS_DIALOG,
    FLIGHTS_DIALOG
} = require('../const');
```

2. Определите и экспортируйте класс `MainDialog`.

```

const initialId = 'mainWaterfallDialog';

class MainDialog extends ComponentDialog {
    constructor() {
        super(MAIN_DIALOG);

        // Create a dialog set for the bot. It requires a DialogState accessor, with which
        // to retrieve the dialog state from the turn context.
        this.addDialog(new ChoicePrompt(INITIAL_PROMPT, this.validateNumberOfAttempts.bind(this)));
        this.addDialog(new FlightDialog(FLIGHTS_DIALOG));

        // Define the steps of the base waterfall dialog and add it to the set.
        this.addDialog(new WaterfallDialog(initialId, [
            this.promptForBaseChoice.bind(this),
            this.respondToBaseChoice.bind(this)
        ]));

        // Define the steps of the hotels waterfall dialog and add it to the set.
        this.addDialog(new HotelsDialog(HOTELS_DIALOG));

        this.initialDialogId = initialId;
    }
}

module.exports.MainDialog = MainDialog;

```

Здесь объявляются другие диалоги и запросы, на которые напрямую ссылается диалог main.

- Каскадный диалог main, который содержит шаги для этого диалога. При запуске компонентный диалог выполняет свой *начальный диалог*.
- Запрос выбора, который будет использоваться для запроса пользователя о задаче, которую необходимо выполнить. Вы создали запрос Choice с помощью проверяющего элемента управления.
- Два дочерних диалога для рейсов и отелей.

3. Добавьте в этот класс вспомогательный метод `run`.

```

/**
 * The run method handles the incoming activity (in the form of a TurnContext) and passes it through
 * the dialog system.
 * If no dialog is active, it will start the default dialog.
 * @param {*} turnContext
 * @param {*} accessor
 */
async run(turnContext, accessor) {
    const dialogSet = new DialogSet(accessor);
    dialogSet.add(this);

    const dialogContext = await dialogSet.createContext(turnContext);
    const results = await dialogContext.continueDialog();
    if (results.status === DialogTurnStatus.empty) {
        await dialogContext.beginDialog(this.id);
    }
}

```

В версии 4 бот при взаимодействии с системой диалогов сначала создает контекстный диалог, а затем вызывает `continueDialog`. Есть уже есть активный диалог, управление передается ему. В противном случае этот вызов просто завершается. Результат `empty` указывает, что диалоговое окно не было активно, и поэтому вы снова запускаете главное диалоговое окно.

Параметр `accessor` передает свойство состояния диалога в метод доступа. В этом свойстве хранится состояние *стека диалогов*. Дополнительные сведения о том, как в версии 4 работают

состояния и диалоги, см. в руководствах по [управлению состоянием](#) и [использованию библиотеки диалогов](#) соответственно.

4. Добавьте к классу каскадные шаги диалога main и проверяющий элемент для запроса выбора.

```
async promptForBaseChoice(stepContext) {
    return await stepContext.prompt(
        INITIAL_PROMPT, {
            prompt: 'Are you looking for a flight or a hotel?',
            choices: ['Hotel', 'Flight'],
            retryPrompt: 'Not a valid option'
        }
    );
}

async respondToBaseChoice(stepContext) {
    // Retrieve the user input.
    const answer = stepContext.result.value;
    if (!answer) {
        // exhausted attempts and no selection, start over
        await stepContext.context.sendActivity('Not a valid option. We\'ll restart the dialog ' +
            'so you can try again!');
        return await stepContext.endDialog();
    }
    if (answer === 'Hotel') {
        return await stepContext.beginDialog(HOTELS_DIALOG);
    }
    if (answer === 'Flight') {
        return await stepContext.beginDialog(FLIGHTS_DIALOG);
    }
    return await stepContext.endDialog();
}

async validateNumberOfAttempts(promptContext) {
    if (promptContext.attemptCount > 3) {
        // cancel everything
        await promptContext.context.sendActivity('Oops! Too many attempts :( But don\'t worry, I\'m ' +
            'handling that exception and you can try again!');
        return await promptContext.context.endDialog();
    }

    if (!promptContext.recognized.succeeded) {
        await promptContext.context.sendActivity(promptContext.options.retryPrompt);
        return false;
    }
    return true;
}
```

Первый шаг каскадного диалога предлагает пользователю сделать выбор. Для этого запускается запрос выбора, который также является диалогом. Второй шаг каскадного диалога обрабатывает результат, полученный от запроса выбора. Он может запустить дочерний диалог (если был сделан выбор) или завершить диалог main (если пользователь не смог сделать выбор).

Запрос выбора возвращает выбранный пользователем вариант, если он считается допустимым, или повторно предлагает пользователю сделать выбор. Проверяющий элемент отслеживает, сколько раз подряд пользователю предлагается выбор, и после 3 неудачных попыток завершает запрос ошибкой и передает управление каскадному диалогу main.

## Реализация диалога для рейсов

В боте версии 3 диалог рейсов был реализован как заглушка, которая демонстрирует обработку ботом сообщений об ошибке беседы. Здесь вы выполняете те же действия.

Отредактируйте файл ./dialogs/flights.js

```
const { ComponentDialog, WaterfallDialog } = require('botbuilder-dialogs');

const initialId = 'flightsWaterfallDialog';

class FlightDialog extends ComponentDialog {
    constructor(id) {
        super(id);

        // ID of the child dialog that should be started anytime the component is started.
        this.initialDialogId = initialId;

        // Define the conversation flow using a waterfall model.
        this.addDialog(new WaterfallDialog(initialId, [
            async () => {
                throw new Error('Flights Dialog is not implemented and is instead ' +
                    'being used to show Bot error handling');
            }
        ]));
    }

    exports.FlightDialog = FlightDialog;
```

### Реализация диалога отелей

Вы сохраняете один общий поток в диалоговом окне отеля: запросите место назначения, запросите дату, запросите число ночей, чтобы остановиться, а затем покажите пользователю список параметров, соответствующих условиям поиска.

Отредактируйте файл ./dialogs/hotels.js.

1. Импортируйте классы и константы, которые понадобятся для диалогового окна.

```
const { ComponentDialog, WaterfallDialog, TextPrompt, DateTimePrompt } = require('botbuilder-dialogs');
const { AttachmentLayoutTypes, CardFactory } = require('botbuilder');
const store = require('../store');
const {
    INITIAL_HOTEL_PROMPT,
    CHECKIN_DATETIME_PROMPT,
    HOW_MANY_NIGHTS_PROMPT
} = require('../const');
```

2. Определите и экспортируйте класс `HotelsDialog`.

```

const initialId = 'hotelsWaterfallDialog';

class HotelsDialog extends ComponentDialog {
    constructor(id) {
        super(id);

        // ID of the child dialog that should be started anytime the component is started.
        this.initialDialogId = initialId;

        // Register dialogs
        this.addDialog(new TextPrompt(INITIAL_HOTEL_PROMPT));
        this.addDialog(new DateTimePrompt(CHECKIN_DATETIME_PROMPT));
        this.addDialog(new TextPrompt(HOW_MANY_NIGHTS_PROMPT));

        // Define the conversation flow using a waterfall model.
        this.addDialog(new WaterfallDialog(initialId, [
            this.destinationPromptStep.bind(this),
            this.destinationSearchStep.bind(this),
            this.checkinPromptStep.bind(this),
            this.checkinTimeSetStep.bind(this),
            this.stayDurationPromptStep.bind(this),
            this.stayDurationSetStep.bind(this),
            this.hotelSearchStep.bind(this)
        ]));
    }
}

exports.HotelsDialog = HotelsDialog;

```

- Добавьте в класс пару вспомогательных функций, которые будут использоваться в шагах диалогового окна.

```

addDays(startDate, days) {
    const date = new Date(startDate);
    date.setDate(date.getDate() + days);
    return date;
};

createHotelHeroCard(hotel) {
    return CardFactory.heroCard(
        hotel.name,
        `${hotel.rating} stars. ${hotel.numberOfReviews} reviews. From ${hotel.priceStarting} per
night.`,
        CardFactory.images([hotel.image]),
        CardFactory.actions([
            {
                type: 'openUrl',
                title: 'More details',
                value: `https://www.bing.com/search?
q=hotels+in+${encodeURIComponent(hotel.location)}`
            }
        ])
    );
}

```

`createHotelHeroCard` создает карту для имиджевого баннера с информацией об отеле.

- Добавьте к этому классу каскадные шаги, которые используются в диалоге.

```

async destinationPromptStep(stepContext) {
    await stepContext.context.sendActivity('Welcome to the Hotels finder!');
    return await stepContext.prompt(
        INITIAL_HOTEL_PROMPT, {
            prompt: 'Please enter your destination'
        }
    );
}

async destinationSearchStep(stepContext) {
    const destination = stepContext.result;
    stepContext.values.destination = destination;
    await stepContext.context.sendActivity(`Looking for hotels in ${destination}`);
    return stepContext.next();
}

async checkinPromptStep(stepContext) {
    return await stepContext.prompt(
        CHECKIN_DATETIME_PROMPT, {
            prompt: 'When do you want to check in?'
        }
    );
}

async checkinTimeSetStep(stepContext) {
    const checkinTime = stepContext.result[0].value;
    stepContext.values.checkinTime = checkinTime;
    return stepContext.next();
}

async stayDurationPromptStep(stepContext) {
    return await stepContext.prompt(
        HOW_MANY_NIGHTS_PROMPT, {
            prompt: 'How many nights do you want to stay?'
        }
    );
}

async stayDurationSetStep(stepContext) {
    const numberOfNights = stepContext.result;
    stepContext.values.numberOfNights = parseInt(numberOfNights);
    return stepContext.next();
}

async hotelSearchStep(stepContext) {
    const destination = stepContext.values.destination;
    const checkIn = new Date(stepContext.values.checkinTime);
    const checkOut = this.addDays(checkIn, stepContext.values.numberOfNights);

    await stepContext.context.sendActivity(`Ok. Searching for Hotels in ${destination} from
${checkIn.toDateString()} to ${checkOut.toDateString()}...`);
    const hotels = await store.searchHotels(destination, checkIn, checkOut);
    await stepContext.context.sendActivity(`I found in total ${hotels.length} hotels for your
dates:`);

    const hotelHeroCards = hotels.map(this.createHotelHeroCard);

    await stepContext.context.sendActivity({
        attachments: hotelHeroCards,
        attachmentLayout: AttachmentLayoutTypes.Carousel
    });

    return await stepContext.endDialog();
}

```

Вы выполнили миграцию шагов из диалогового окна v3 Отели в каскадные шаги диалогового окна

## Обновление бота

В версии 4 бот может реагировать на действия за пределами системы диалогов. Класс `ActivityHandler` определяет обработчики для распространенных типов действий, чтобы упростить управление кодом.

Переименуйте файл `./bot.js` в `./bots/reservationBot.js` и измените его.

1. Этот файл импортирует `ActivityHandler`, который предоставляет базовую реализацию бота.

```
const { ActivityHandler } = require('botbuilder');
```

2. Переименуйте класс в `ReservationBot`.

```
class ReservationBot extends ActivityHandler {
    // ...
}

module.exports.ReservationBot = ReservationBot;
```

3. Обновите сигнатуру конструктора, чтобы принять получаемые объекты.

```
/**
 *
 * @param {ConversationState} conversationState
 * @param {Dialog} dialog
 * @param {any} logger object for logging events, defaults to console if none is provided
 */
constructor(conversationState, dialog, logger) {
    super();
    // ...
}
```

4. В конструкторе добавьте проверку параметров со значением `NULL` и определите свойства для конструктора класса.

```
if (!conversationState) throw new Error('[DialogBot]: Missing parameter. conversationState is required');
if (!dialog) throw new Error('[DialogBot]: Missing parameter. dialog is required');
if (!logger) {
    logger = console;
    logger.log('[DialogBot]: logger not passed in, defaulting to console');
}

this.conversationState = conversationState;
this.dialog = dialog;
this.logger = logger;
this.dialogState = this.conversationState.createProperty('DialogState');
```

Здесь создается метод доступа к свойству состояния диалогового окна, который будет хранить состояние стека диалоговых окон.

5. Обновите в конструкторе обработчик `onMessage` и добавьте обработчик `onDialog`.

```

this.onMessage(async (context, next) => {
    this.logger.log('Running dialog with Message Activity.');

    // Run the Dialog with the new message Activity.
    await this.dialog.run(context, this.dialogState);

    // By calling next() you ensure that the next BotHandler is run.
    await next();
});

this.onDialog(async (context, next) => {
    // Save any state changes. The load happened during the execution of the Dialog.
    await this.conversationState.saveChanges(context, false);

    // By calling next() you ensure that the next BotHandler is run.
    await next();
});

```

`ActivityHandler` перенаправляет действия сообщений в `onMessage`. Этот бот обрабатывает все введенные пользователем в диалоги данные.

`ActivityHandler` вызывает `onDialog` в конце шага, прежде чем возвращать управление адаптеру. Необходимо явно сохранить состояние перед выходом из очереди. В противном случае изменения состояния не будут сохраняться, что нарушит работу диалога.

## 6. И наконец, обновите обработчик `onMembersAdded` в конструкторе.

```

this.onMembersAdded(async (context, next) => {
    const membersAdded = context.activity.membersAdded;
    for (let cnt = 0; cnt < membersAdded.length; ++cnt) {
        if (membersAdded[cnt].id !== context.activity.recipient.id) {
            await context.sendActivity('Hello and welcome to Contoso help desk bot.');
        }
    }
    // By calling next() you ensure that the next BotHandler is run.
    await next();
});

```

`ActivityHandler` вызывает `onMembersAdded` при получении действия обновления беседы, информируя о добавлении к беседе любого участника, кроме нашего бота. Вы обновляете этот метод для отправки сообщения приветствия, когда пользователь присоединяется к беседе.

## Создание файла для хранилища

Создайте файл `./store.js`, требуемый для диалога отелей. `searchHotels` имитирует работу функции поиска отелей, как и в старом боте версии 3.

```

module.exports = {
  searchHotels: destination => {
    return new Promise(resolve => {

      // Filling the hotels results manually just for demo purposes
      const hotels = [];
      for (let i = 1; i <= 5; i++) {
        hotels.push({
          name: `${destination} Hotel ${i}`,
          location: destination,
          rating: Math.ceil(Math.random() * 5),
          numberOfReviews: Math.floor(Math.random() * 5000) + 1,
          priceStarting: Math.floor(Math.random() * 450) + 80,
          image: `https://placeholdit.imgix.net/~text?txtsize=35&txt=Hotel${i}&w=500&h=260`
        });
      }

      hotels.sort((a, b) => a.priceStarting - b.priceStarting);

      // complete promise with a timer to simulate async response
      setTimeout(() => { resolve(hotels); }, 1000);
    });
  }
};

```

## Тестирование бота в эмуляторе

На этом этапе вы сможете запустить программу Bot локально и присоединить к ней с помощью эмулятора.

1. Выполните этот пример на локальном компьютере. Если сеанс отладки запущен в Visual Studio Code, отладочные сведения отправляются в консоль отладки при любых действиях с ботом.
2. Запустите эмулятор и подключитесь к роботу.
3. Отправьте сообщения, чтобы протестировать все диалоги: диалог main, а также диалоги рейсов и отелей.

## Дополнительные ресурсы

Тематические статьи по версии 4:

- [Принципы работы бота](#)
- [Управление состоянием](#)
- [Библиотека диалогов](#)

Пошаговые инструкции для версии 4:

- [Отправка и получение текстовых сообщений](#)
- [Сохранение данных пользователя и диалога](#)
- [Реализация процесса общения](#)
- [Отладка с помощью эмулятора](#)
- [Добавление данных телеметрии в бот](#)

# Использование данных JavaScript о состоянии пользователя версии 3 в боте версии 4

27.03.2021 • 5 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье описано, как бот версии 4 может выполнять с информацией о состоянии пользователя версии 3 операции чтения, записи и удаления.

Пример кода можно найти в [примере адаптера для пользовательского состояния v4 v3](#).

## NOTE

Бот поддерживает **состояние беседы** для ее отслеживания, управления ее ходом и отправки вопросов пользователю. Он поддерживает **состояние пользователя** для отслеживания ответов пользователя.

## Предварительные требования

- NPM версии 6.9.0 или более поздней (требуется для поддержки псевдонимов пакетов).
- Node.js версии 10.14.1 или более поздней.

Чтобы проверить, какая версия установлена на компьютере, выполните следующую команду в окне терминала:

```
# determine node version
node --version
```

## Настройка

### 1. Клонирование репозитория

```
git clone https://github.com/microsoft/botbuilder-samples.git
```

### 2. В окне терминала перейдите в папку

```
BotBuilder-Samples/MigrationV3V4/Node/V4V3-user-state-adapter-sample-bot .
```

```
cd BotBuilder-Samples/MigrationV3V4/Node/V4V3-user-state-adapter-sample-bot
```

### 3. Запустите команду `npm install` в следующих расположениях:

```
root
/V4V3StorageMapper
/V4V3UserState
```

### 4. Выполните команду `npm run build` или `tsc`, чтобы скомпилировать модули `StorageMapper` и `UserState` в следующих расположениях:

```
/V4V3StorageMapper  
/V4V3UserState
```

5. Настройте базу данных.

- Скопируйте содержимое файла `.env.example`.
- Создайте новый файл с именем `.env` и вставьте в него предыдущее содержимое.
- Укажите значения для своих поставщиков хранилищ. Обратите внимание, что *имя пользователя, пароль и сведения об узле* можно найти на портале Azure в разделе для конкретного поставщика хранилища, например *Cosmos DB*, *Хранилище таблиц* или *База данных SQL*. Имена таблиц и коллекций определяются пользователем.

6. Укажите поставщика хранилища для бота.

- Откройте файл `index.js` в корне проекта. В начале файла (приблизительно строки 38–98) хранятся настройки для каждого поставщика хранилища (указаны в комментариях). Они считывают значения конфигурации из файла `.env` через `process.env` узла. В приведенном ниже фрагменте кода показано, как настроить Базу данных SQL.

```
/*-----  
   SQL Database Configuration  
-----*/  
  
const sqlConfig = {  
    userName: process.env.SQL_USER_NAME, // obtain from Azure Portal  
    password: process.env.SQL_PASSWORD, // obtain from Azure Portal  
    server: process.env.SQL_SERVER_HOST, // obtain from Azure Portal  
    enforceTable: true, // If this property is not set to true it defaults to false. When  
    // false if the specified table is not found, the bot will throw an error.  
    options: {  
        database: process.env.SQL_DATABASE_NAME, // user defined  
        table: process.env.SQL_TABLE_NAME, // user defined  
        encrypt: true,  
        rowCollectionOnRequestCompletion: true  
    }  
}
```

- Укажите поставщика хранилища, который будет использоваться ботом, передав выбранный вами экземпляр клиента хранилища адаптеру `StorageMapper` (приблизительно строка 107).

```
/** Pass current storage provider to StorageMapper **/  
/** possible values: cosmosStorageClient, tableStorage, sqlStorage **/  
const storageMapper = new StorageMapper(cosmosStorageClient);
```

Значение по умолчанию: *Cosmos DB*. Вы можете выбрать

```
cosmosStorageClient  
tableStorage  
sqlStorage
```

7. Запустите приложение. В корне проекта выполните следующую команду:

```
npm run start
```

## Классы адаптера

## V4V3StorageMapper

Класс `StorageMapper` содержит основную функциональность адаптера. Он реализует интерфейс хранилища версии 4 и сопоставляет методы поставщика хранилища (чтение, запись и удаление) с классами поставщика хранилища версии 3, чтобы состояние пользователя в формате версии 3 можно было использовать в боте версии 4.

## V4V3UserState

Этот класс расширяет класс `BotState` версии 4 (`botbuilder-core`), чтобы он использовал ключ в стиле версии 3. Это позволяет выполнять операции чтения, записи и удаления с хранилищем версии 3.

## Тестирование бота с помощью Bot Framework Emulator

[Bot Framework Emulator](#) — это классическое приложение, которое позволяет тестировать ботов локально или удаленно (через туннель) и выполнять их отладку.

- Установите эмулятор Bot Framework [версии 4.3.0 или более поздней](#).

### Подключение к боту с помощью Bot Framework Emulator

1. Запустите Bot Framework Emulator.
2. Введите следующий URL-адрес (конечная точка): `http://localhost:3978/api/messages`.

### Этапы проверки

1. Откройте программу Bot в эмуляторе и отправьте сообщение. При появлении запроса укажите свое имя.
2. После окончания шага отправьте боту еще одно сообщение.
3. Убедитесь, что запрос на ввод имени не отображается повторно. Бот должен считать его из хранилища и распознать, что он уже отправлял вам такой запрос.
4. Бот должен вернуть полученное от вас сообщение.
5. Перейдите к поставщику хранилища в Azure и убедитесь, что ваше имя хранится в виде пользовательских данных в базе данных.

## Развертывание бота в Azure

Полный список инструкций для развертывания бота в Azure см. в статье [Развертывание бота](#).

## Дополнительные материалы

- [Общие сведения о службе Azure Bot](#)
- [Состояние бота](#)
- [Непосредственная запись в хранилище](#)
- [Управление состоянием беседы и пользователя](#)
- [Restify](#)
- [dotenv](#)

# Преобразование бота версии 3 в навык

27.10.2020 • 2 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В некоторых сценариях нет смысла для миграции бота из версии 3 в версию 4, но у вас может возникнуть желание воспользоваться дополнительными функциями пакета SDK версии 4. В таких случаях вы можете преобразовать бот версии 3 в навык и на основе пакета SDK версии 4 создать бот, который будет выполнять роль потребителя навыков и передавать сообщения в бот версии 3. Дополнительные сведения о навыках и потребителях навыков см. в статье [с обзорными сведениями о навыках](#).

Для более сложных ботов такой подход позволяет выполнять миграцию постепенно. Вы сможете выбирать, какие сообщения потребитель навыка будет обрабатывать сам, а какие — передавать в навык. Это позволяет поэтапно переносить функциональность в новый бот, чтобы в конечном итоге полностью отключить навык после перемещения всех функций.

## Необходимые условия

### Обновление пакета SDK бота версии 3 до последней версии 3.x

Необходимые обработчики для преобразования бота в навык были добавлены в пакет SDK для .NET версии 3.30.2 и пакет SDK для JavaScript версии 3.30.0.

### Преобразование бота версии 3 в навык

После обновления версии пакета SDK вам нужно добавить в бот некоторую логику, чтобы управлять обменом сообщений с потребителем навыка.

### Создание бота версии 4 в качестве потребителя навыка

Вам нужно создать новый бот, который будет выполнять роль потребителя навыка, и добавить в него логику для отбора сообщений, направляемых в этот навык. Этот новый бот будет взаимодействовать с клиентами. Именно в него вы добавите новые функции, которые используют расширенные возможности пакета SDK версии 4.

### Подключение пользователей к новому боту, выполняющему роль потребителя навыка

И теперь вам осталось лишь заменить бот версии 3 новым ботом версии 4.

## Начало работы

Готовы начать работу?

- [Преобразование бота .NET версии 3 в навык](#)
- [Преобразование бота JavaScript версии 3 в навык](#)

# Преобразование бота .NET версии 3 в навык

27.10.2020 • 23 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье описывается, как преобразовать три примера ботов .NET версии 3 в навыки и создать потребитель навыков версии 4, который может обращаться к этим навыкам. Чтобы преобразовать бот JavaScript версии 3 в навык, воспользуйтесь [этой инструкцией](#). Сведения о переносе бота .NET из версии 3 в версию 4 см. в [этой статье](#).

## Предварительные требования

- Visual Studio 2019.
- .NET Core 3.1.
- .NET Framework 4.6.1 или более поздней версии.
- Подписка Azure. Если у вас еще нет подписки Azure, [создайте бесплатную учетную запись](#), прежде чем начинать работу.
- Копии примеров ботов для .NET версии 3, которые вы будете преобразовывать: эхо-бот, [PizzaBot](#) и [SimpleSandwichBot](#).
- Копия примера потребителя навыка для .NET версии 4: [SimpleRootBot](#).

## Сведения о ботах

В этой статье каждый бот версии 3 обновляется для использования в качестве навыка. Предоставляется также потребитель навыка версии 4, который позволяет протестировать боты, преобразованные в навыки.

- Программа EchoBot возвращает все полученные сообщения. Работая как навык, она *завершается* при получении сообщения end или stop. Преобразуемый бот основан на шаблоне проекта Bot Builder Echo Bot для версии 3.
- PizzaBot проводит пользователя через процесс заказа пиццы. Работая как навык, этот бот после завершения отправляет заказ пользователю обратно в родительский объект.
- SimpleSandwichBot проводит пользователя через процесс заказа сандвичей. Работая как навык, этот бот после завершения отправляет заказ пользователю обратно в родительский объект.

Кроме того, в примере потребителя навыков версии 4 [SimpleRootBot](#) показано, как можно использовать и тестировать навыки.

Чтобы использовать потребитель навыков для тестирования навыков, все 4 бота должны выполняться одновременно. Боты можно тестировать локально с помощью Bot Framework Emulator, где каждый бот использует свой локальный порт.

## Создание ресурсов Azure для ботов

Для проверки подлинности в сценарии взаимодействия ботов требуется предоставить каждому из этих ботов действительный идентификатор приложения и пароль.

1. Создайте нужное количество регистраций каналов для ботов.
2. Запишите идентификатор приложения и пароль для каждой из них.

## Процесс преобразования

Чтобы преобразовать существующий бот в бот-навык, нужно выполнить лишь несколько шагов, которые описаны в следующих разделах. Более подробные сведения о навыках см. [здесь](#).

- Обновите файл `web.config` бота, включив в него идентификатор приложения и пароль и добавив свойство `allowed callers`.
- Добавьте проверку утверждений. Таким образом вы сможете ограничить доступ к навыку, предоставляя его только пользователям и корневому боту. Дополнительные сведения о стандартных и пользовательских проверках утверждений см. [здесь](#).
- Измените контроллер сообщений бота, чтобы обрабатывать действия `endOfConversation` от корневого бота.
- Измените код бота, чтобы он возвращал действие `endOfConversation` после завершения работы навыка.
- При каждом завершении работы навыка, если он еще поддерживает беседу или некоторые ресурсы, ему необходимо очистить состояние беседы и освободить ресурсы.
- При необходимости добавьте файл манифеста. Так как потребитель навыка не всегда имеет доступ к коду этого навыка, опишите в манифесте все действия, которые ваш навык умеет получать и создавать, все его входные и выходные параметры, а также конечные точки. Текущая схема манифеста хранится в файле `skill-manifest-2.0.0.json`.

## Преобразование эхо-бота

1. Создайте новый проект из шаблона проекта Bot Builder Echo Bot для версии 3 и настройте его для использования порта 3979.
  - a. Создайте проект.
  - b. Откройте вкладку свойств этого проекта.
  - c. Выберите категорию **Веб** и задайте для параметра **URL-адрес проекта** значение `http://localhost:3979/`.
  - d. Сохраните изменения и закройте вкладку свойств.
2. В файл конфигурации добавьте идентификатор приложения и пароль эхо-бота. Там же, в параметрах приложения, добавьте свойство `EchoBotAllowedCallers` и присвойте ему значение идентификатора приложения простого корневого бота.

### V3EchoBot\Web.config

```
<appSettings>
    <!-- update these with your Microsoft App Id and your Microsoft App Password-->
    <add key="MicrosoftAppId" value="YOUR Echo bot's MicrosoftAppId" />
    <add key="MicrosoftAppPassword" value="YOUR Echo bot's MicrosoftAppPassword" />
    <add key="EchoBotAllowedCallers" value="YOUR root bot's MicrosoftAppId" />
</appSettings>
```

3. Добавьте средство проверки утверждений и вспомогательный класс конфигурации проверки подлинности.

### V3EchoBot\Authentication\CustomAllowedCallersClaimsValidator.cs

Этот пример реализует пользовательское средство проверки утверждений, а при неудачной проверке создает ошибку `UnauthorizedAccessException`.

```
using Microsoft.Bot.Connector.SkillAuthentication;
using System;
using System.Collections.Generic;
```

```

using System.Collections.Generic;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;

namespace Microsoft.Bot.Sample.EchoBot.Authentication
{
    /// <summary>
    /// Sample claims validator that loads an allowed list from configuration if present
    /// and checks that requests are coming from allowed parent bots.
    /// </summary>
    public class CustomAllowedCallersClaimsValidator : ClaimsValidator
    {
        private readonly IList<string> _allowedCallers;

        public CustomAllowedCallersClaimsValidator(IList<string> allowedCallers)
        {
            // AllowedCallers is the setting in web.config file
            // that consists of the list of parent bot IDs that are allowed to access the skill.
            // To add a new parent bot simply go to the AllowedCallers and add
            // the parent bot's Microsoft app ID to the list.

            _allowedCallers = allowedCallers ?? throw new
ArgumentOutOfRangeException(nameof(allowedCallers));
            if (!_allowedCallers.Any())
            {
                throw new ArgumentException(nameof(allowedCallers), "AllowedCallers must contain
at least one element of '*' or valid MicrosoftAppId(s).");
            }
        }

        /// <summary>
        /// This method is called from JwtTokenValidation.ValidateClaimsAsync
        /// </summary>
        /// <param name="claims"></param>
        public override Task ValidateClaimsAsync(IList<Claim> claims)
        {
            if (claims == null)
            {
                throw new ArgumentNullException(nameof(claims));
            }

            if (!claims.Any())
            {
                throw new UnauthorizedAccessException("ValidateClaimsAsync.claims parameter must
contain at least one element.");
            }

            if (SkillValidation.IsSkillClaim(claims))
            {
                // if _allowedCallers has one item of '*', allow all parent bot calls and do not
                validate the appid from claims
                if (_allowedCallers.Count == 1 && _allowedCallers[0] == "*")
                {
                    return Task.CompletedTask;
                }

                // Check that the appId claim in the skill request is in the list of skills
                configured for this bot.
                var appId = JwtTokenValidation.GetAppIdFromClaims(claims).ToUpperInvariant();
                if (_allowedCallers.Contains(appId))
                {
                    return Task.CompletedTask;
                }

                throw new UnauthorizedAccessException($"Received a request from a bot with an app ID
of \'{appId}\'. To enable requests from this caller, add the app ID to your configuration file.");
            }
        }
    }
}

```

```
        throw new UnauthorizedAccessException($"validateClaimsAsync called without a skill claim  
in claims.");  
    }  
}
```

### V3EchoBot\Authentication\CustomSkillAuthenticationConfiguration.cs

Этот пример загружает сведения о допустимыхзывающих из файла конфигурации и применяет `CustomAllowedCallersClaimsValidator` для проверки утверждения.

```
using Microsoft.Bot.Connector.SkillAuthentication;  
using System.Configuration;  
using System.Linq;  
  
namespace Microsoft.Bot.Sample.EchoBot.Authentication  
{  
    public class CustomSkillAuthenticationConfiguration : AuthenticationConfiguration  
    {  
        private const string AllowedCallersConfigKey = "EchoBotAllowedCallers";  
        public CustomSkillAuthenticationConfiguration()  
        {  
            // Could pull this list from a DB or anywhere.  
            var allowedCallers =  
                ConfigurationManager.AppSettings[AllowedCallersConfigKey].Split(',').Select(s =>  
                    s.Trim().ToUpperInvariant()).ToList();  
            ClaimsValidator = new CustomAllowedCallersClaimsValidator(allowedCallers);  
        }  
    }  
}
```

4. Обновите класс `MessagesController`.

### V3EchoBot\Controllers\MessagesController.cs

Обновите инструкции `using`.

```
using System.Diagnostics;  
using System.Net;  
using System.Net.Http;  
using System.Threading;  
using System.Threading.Tasks;  
using System.Web.Http;  
using Autofac;  
using Microsoft.Bot.Builder.Dialogs;  
using Microsoft.Bot.Builder.Dialogs.Internals;  
using Microsoft.Bot.Connector;  
using Microsoft.Bot.Connector.SkillAuthentication;  
using Microsoft.Bot.Sample.EchoBot.Authentication;
```

Замените атрибут класса `BotAuthentication` на `SkillBotAuthentication`. Используйте необязательный параметр `AuthenticationConfigurationProviderType`, чтобы применить пользовательский поставщик проверки утверждений.

```
// Specify which type provides the authentication configuration to allow for validation for skills.  
[SkillBotAuthentication(AuthenticationConfigurationProviderType =  
typeof(CustomSkillAuthenticationConfiguration))]  
public class MessagesController : ApiController
```

В методе `HandleSystemMessage` добавьте условие для обработки сообщения `endOfConversation`. Это позволит навыкам очищать состояние и освобождать ресурсы при завершении беседы по сигналу

от потребителя навыков.

```
if (messageType == ActivityTypes.EndOfConversation)
{
    Trace.TraceInformation($"EndOfConversation: {message}");

    // This Recipient null check is required for PVA manifest validation.
    // PVA will send an EOC activity with null Recipient.
    if (message.Recipient != null)
    {
        // Clear the dialog stack if the root bot has ended the conversation.
        using (var scope = DialogModule.BeginLifetimeScope(Conversation.Container, message))
        {
            var botData = scope.Resolve<IBotData>();
            await botData.LoadAsync(default(CancellationToken));

            var stack = scope.Resolve<IDialogStack>();
            stack.Reset();

            await botData.FlushAsync(default(CancellationToken));
        }
    }
}
```

5. Измените код бота так, чтобы навык мог отмечать завершение беседы при получении от пользователя сообщения end или stop. Также навык должен очищать состояние и освобождать ресурсы, когда он завершает беседу.

#### V3EchoBot\Dialogs\RootDialog.cs

```
private async Task MessageReceivedAsync(IDialogContext context, IAwaitable<object> result)
{
    var activity = await result as Activity;

    // Send an `endOfConversation` activity if the user cancels the skill.
    if (activity.Text.ToLower().Contains("end") || activity.Text.ToLower().Contains("stop"))
    {
        await context.PostAsync($"Ending conversation from the skill...");
        var endOfConversation = activity.CreateReply();
        endOfConversation.Type = ActivityTypes.EndOfConversation;
        endOfConversation.Code = EndOfConversationCodes.UserCancelled;
        await context.PostAsync(endOfConversation);
    }
    else
    {
        await context.PostAsync($"Echo (dotnet V3): {activity.Text}");
        await context.PostAsync($"Say 'end' or 'stop' and I'll end the conversation and back to the parent.");
    }

    context.Wait(MessageReceivedAsync);
}
```

6. Примените этот манифест для эхо-бота. В качестве идентификатора приложения конечной точки укажите идентификатор приложения бота.

#### V3EchoBot\wwwroot\echo-bot-manifest.json

```
{
  "$schema": "https://raw.githubusercontent.com/microsoft/botframework-
sdk/master/schemas/skills/skill-manifest-2.0.0.json",
  "$id": "YourEchoBotHandle",
  "name": "V3 Echo Skill Bot",
  "version": "1.0",
  "description": "This is a sample skill for echoing what the user sent the bot.",
  "publisherName": "Microsoft",
  "privacyUrl": "https://microsoft.com/privacy",
  "copyright": "Copyright (c) Microsoft Corporation. All rights reserved.",
  "license": "",
  "iconUrl": "https://myskill.contoso.com/icon.png",
  "tags": [
    "sample",
    "echo"
  ],
  "endpoints": [
    {
      "name": "YourEchoBotName",
      "protocol": "BotFrameworkV3",
      "description": "Default endpoint for the skill",
      "endpointUrl": "http://localhost:3979/api/messages",
      "msAppId": "Your Echo Bot's MicrosoftAppId"
    }
  ],
  "activities": {
    "EchoDotNetV3": {
      "description": "Echo user responses",
      "type": "message",
      "name": "V3Echo"
    }
  }
}
```

См. сведения о [схеме манифеста навыка \(skill-manifest-2.0.0.json\)](#).

## Преобразование бота для заказа пиццы

1. Откройте копию проекта PizzaBot и настройте в нем использование порта 3980.
  - a. Откройте вкладку свойств этого проекта.
  - b. Выберите категорию **Веб** и задайте для параметра **URL-адрес проекта** значение `http://localhost:3980/`.
  - c. Сохраните изменения и закройте вкладку свойств.
2. В файл конфигурации добавьте идентификатор приложения и пароль бота для заказа пиццы. Там же, в параметрах приложения, добавьте свойство `AllowedCallers` и присвойте ему значение идентификатора приложения простого корневого бота.

### V3PizzaBot\Web.config

```
<appSettings>
  <!-- update these with your Microsoft App Id and your Microsoft App Password-->
  <add key="MicrosoftAppId" value="YOUR Pizza bot's MicrosoftAppId" />
  <add key="MicrosoftAppPassword" value="YOUR Pizza bot's MicrosoftAppPassword" />
  <add key="AllowedCallers" value="YOUR root bot's MicrosoftAppId" />
</appSettings>
```

3. Добавьте класс `ConversationHelper` со вспомогательными методами, реализовав следующее:
  - Отправка действия `endOfConversation` по завершении навыка. Позволяет вернуть сведения о

заказе в свойстве `Value` действия и задать свойство `Code`, чтобы обозначить причину завершения беседы.

- Очистка состояния беседы и освобождение всех связанных ресурсов.

### V3PizzaBot\ConversationHelper.cs

```
using System;
using System.Collections.Concurrent;
using System.Configuration;
using System.Threading;
using System.Threading.Tasks;
using Autofac;
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Builder.Dialogs.Internals;
using Microsoft.Bot.Connector;
using Newtonsoft.Json;

namespace Microsoft.Bot.Sample.PizzaBot
{
    internal static class ConversationHelper
    {
        private static readonly ConcurrentDictionary<string, ConnectorClient> _connectorClientCache =
            new ConcurrentDictionary<string, ConnectorClient>();

        /// <summary>
        /// Helper method that sends an `endOfConversation` activity.
        /// </summary>
        /// <param name="incomingActivity">The incoming user activity for this turn.</param>
        /// <param name="order">Optional. The completed order.</param>
        /// <param name="endOfConversationCode">Optional. The EndOfConversationCode to send to the parent bot.
        /// Defaults to EndOfConversationCodes.CompletedSuccessfully.</param>
        /// <remarks>Sending the `endOfConversation` activity when the conversation completes allows the bot to be consumed as a skill.</remarks>
        internal static async Task EndConversation(Activity incomingActivity, PizzaOrder order =
null, string endOfConversationCode = EndOfConversationCodes.CompletedSuccessfully)
        {
            var connectorClient = _connectorClientCache.GetOrAdd(incomingActivity.ServiceUrl, key =>
            {
                var appId =
                    ConfigurationManager.AppSettings[MicrosoftAppCredentials.MicrosoftAppIdKey];
                var appPassword =
                    ConfigurationManager.AppSettings[MicrosoftAppCredentials.MicrosoftAppPasswordKey];
                return new ConnectorClient(new Uri(incomingActivity.ServiceUrl), appId, appPassword);
            });

            // Send End of conversation as reply.
            await
                connectorClient.Conversations.SendToConversationAsync(incomingActivity.CreateReply("Ending
conversation from the skill..."));
            var endOfConversation = incomingActivity.CreateReply();
            if (order != null)
            {
                endOfConversation.Value = JsonConvert.SerializeObject(order);
            }
            endOfConversation.Type = ActivityTypes.EndOfConversation;
            endOfConversation.Code = endOfConversationCode;
            await connectorClient.Conversations.SendToConversationAsync(endOfConversation);
        }

        /// <summary>
        /// Clear the dialog stack and data bags.
        /// </summary>
        /// <param name="activity">The incoming activity to use for scoping the Conversation.Container.</param>
        internal static async Task ClearState(Activity activity)
        {
            // This is required for DVA manifest validation
        }
    }
}
```

```

// THIS IS REQUIRED FOR PVA MANIFEST VALIDATION.
// PVA will send an EOC activity with null Recipient.
if (activity.Recipient == null)
    return;

using (var scope = DialogModule.BeginLifetimeScope(Conversation.Container, activity))
{
    var botData = scope.Resolve<IBotData>();
    await botData.LoadAsync(default(CancellationToken));

    // Some skills might persist data between invocations.
    botData.UserData.Clear();
    botData.ConversationData.Clear();
    botData.PrivateConversationData.Clear();

    var stack = scope.Resolve<IDialogStack>();
    stack.Reset();

    await botData.FlushAsync(default(CancellationToken));
}
}
}
}

```

4. Обновите класс `MessagesController`.

#### V3PizzaBot\Controllers\MessagesController.cs

Обновите инструкции using.

```

using System.Web.Http;
using System.Threading.Tasks;

using Microsoft.Bot.Connector;
using Microsoft.Bot.Builder.FormFlow;
using Microsoft.Bot.Builder.Dialogs;
using System.Web.Http.Description;
using System.Net.Http;
using System.Diagnostics;
using Microsoft.Bot.Connector.SkillAuthentication;
using Microsoft.Bot.Builder.Dialogs.Internals;
using Autofac;
using System.Threading;

```

Замените атрибут класса `BotAuthentication` на `SkillBotAuthentication`. Этот бот использует стандартное средство проверки утверждений.

```

[SkillBotAuthentication]
public class MessagesController : ApiController

```

В методе `Post` измените условие действия `message`, чтобы разрешить пользователю отменить процесс заказа из навыка. Также добавьте условие действия `endOfConversation`, которое позволит навыкам очищать состояние и освобождать ресурсы при завершении беседы по сигналу от потребителя навыков.

```

case ActivityTypes.Message:
    // Send an `endOfConversation` activity if the user cancels the skill.
    if (activity.Text.ToLower().Contains("end") || activity.Text.ToLower().Contains("stop"))
    {
        await ConversationHelper.ClearState(activity);
        await ConversationHelper.EndConversation(activity, endOfConversationCode:
EndOfConversationCodes.UserCancelled);
    }
    else
    {
        await Conversation.SendAsync(activity, MakeRoot);
    }
    break;
case ActivityTypes.EndOfConversation:
    Trace.TraceInformation($"EndOfConversation: {activity}");

    // Clear the dialog stack if the root bot has ended the conversation.
    await ConversationHelper.ClearState(activity);

    break;

```

## 5. Изменение кода бота.

### V3PizzaBot\PizzaOrderDialog.cs

Обновите инструкции using.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Builder.FormFlow;
using Microsoft.Bot.Builder.Luis;
using Newtonsoft.Json;
using Microsoft.Bot.Builder.Luis.Models;
using Microsoft.Bot.Connector;

```

Добавьте приветственное сообщение. Это поможет пользователю понять, что происходит, когда корневой бот вызывает бот для заказа пиццы в качестве навыка.

```

public override async Task StartAsync(IDialogContext context)
{
    await context.PostAsync("Welcome to the Pizza Order Bot. Let me know if you would like to order a
pizza, or know our store hours.");
    await context.PostAsync("Say 'end' or 'stop' and I'll end the conversation and back to the
parent.");

    await base.StartAsync(context);
}

```

Измените код бота так, чтобы навык мог отмечать завершение диалога, когда пользователь отменяет или завершает заказ. Также навык должен очищать состояние и освобождать ресурсы, когда он завершает беседу.

```
private async Task PizzaFormComplete(IDialogContext context, IAwaitable<PizzaOrder> result)
{
    PizzaOrder order = null;
    try
    {
        order = await result;
    }
    catch (OperationCanceledException)
    {
        await context.PostAsync("You canceled the form!");

        // If the user cancels the skill, send an `endOfConversation` activity to the skill consumer.
        await ConversationHelper.EndConversation(context.Activity as Activity, endOfConversationCode:
EndOfConversationCodes.UserCancelled);
        return;
    }

    if (order != null)
    {
        await context.PostAsync("Your Pizza Order: " + order.ToString());
    }
    else
    {
        await context.PostAsync("Form returned empty response!");
    }

    // When the skill completes, send an `endOfConversation` activity and include the finished order.
    await ConversationHelper.EndConversation(context.Activity as Activity, order);
    context.Wait(MessageReceived);
}
```

6. Примените этот манифест для бота заказа пиццы. В качестве идентификатора приложения конечной точки укажите идентификатор приложения бота.

V3PizzaBot\wwwroot\pizza-bot-manifest.json

```
{
  "$schema": "https://raw.githubusercontent.com/microsoft/botframework-sdk/master/schemas/skills/skill-manifest-2.0.0.json",
  "$id": "YourPizzaBotHandle",
  "name": "Pizza Skill Bot",
  "version": "1.0",
  "description": "This is a sample skill for ordering a Pizza.",
  "publisherName": "Microsoft",
  "privacyUrl": "https://microsoft.com/privacy",
  "copyright": "Copyright (c) Microsoft Corporation. All rights reserved.",
  "license": "",
  "iconUrl": "https://myskill.contoso.com/icon.png",
  "tags": [
    "sample",
    "pizza"
  ],
  "endpoints": [
    {
      "name": "YourPizzaBotName",
      "protocol": "BotFrameworkV3",
      "description": "Default endpoint for the skill",
      "endpointUrl": "http://localhost:3980/api/messages",
      "msAppId": "YOUR Pizza Bot's MicrosoftAppId"
    }
  ],
  "activities": {
    "OrderPizza": {
      "description": "Order a Pizza",
      "type": "message",
      "name": "OrderPizza"
    }
  }
}
```

См. сведения о [схеме манифеста навыка \(skill-manifest-2.0.0.json\)](#).

## Преобразование бота для заказа сандвичей

1. Откройте копию проекта SimpleSandwichBot и настройте в нем использование порта 3981.
  - a. Откройте вкладку свойств этого проекта.
  - b. Выберите категорию **Веб** и задайте для параметра **URL-адрес проекта** значение `http://localhost:3981/`.
  - c. Сохраните изменения и закройте вкладку свойств.
2. В файл конфигурации добавьте идентификатор приложения и пароль бота для заказа пиццы. Там же, в параметрах приложения, добавьте свойство `AllowedCallers` и присвойте ему значение идентификатора приложения простого корневого бота.

### V3SimpleSandwichBot\Web.config

```
<appSettings>
  <!-- update these with your Microsoft App Id and your Microsoft App Password-->
  <add key="MicrosoftAppId" value="YOUR Sandwich bot's MicrosoftAppId" />
  <add key="MicrosoftAppPassword" value="YOUR Sandwich bot's MicrosoftAppPassword" />
  <add key="AllowedCallers" value="YOUR root bot's MicrosoftAppId" />
</appSettings>
```

3. Добавьте класс `ConversationHelper` со вспомогательными методами, реализовав следующее:
  - Отправка действия `endOfConversation` по завершении навыка. Позволяет вернуть сведения о

заказе в свойстве `Value` действия и задать свойство `Code`, чтобы обозначить причину завершения беседы.

- Очистка состояния беседы и освобождение всех связанных ресурсов.

### V3SimpleSandwichBot\ConversationHelper.cs

```
using System;
using System.Collections.Concurrent;
using System.Configuration;
using System.Threading;
using System.Threading.Tasks;
using Autofac;
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Builder.Dialogs.Internals;
using Microsoft.Bot.Connector;
using Newtonsoft.Json;

namespace Microsoft.Bot.Sample.SimpleSandwichBot
{
    internal static class ConversationHelper
    {
        private static readonly ConcurrentDictionary<string, ConnectorClient> _connectorClientCache =
            new ConcurrentDictionary<string, ConnectorClient>();

        /// <summary>
        /// Helper method that sends an `endOfConversation` activity.
        /// </summary>
        /// <param name="incomingActivity">The incoming user activity for this turn.</param>
        /// <param name="order">Optional. The completed order.</param>
        /// <param name="endOfConversationCode">Optional. The EndOfConversationCode to send to the parent bot.
        /// Defaults to EndOfConversationCodes.CompletedSuccessfully.</param>
        /// <remarks>Sending the `endOfConversation` activity when the conversation completes allows the bot to be consumed as a skill.</remarks>
        internal static async Task EndConversation(Activity incomingActivity, SandwichOrder order =
null, string endOfConversationCode = EndOfConversationCodes.CompletedSuccessfully)
        {
            var connectorClient = _connectorClientCache.GetOrAdd(incomingActivity.ServiceUrl, key =>
            {
                var appId =
                    ConfigurationManager.AppSettings[MicrosoftAppCredentials.MicrosoftAppIdKey];
                var appPassword =
                    ConfigurationManager.AppSettings[MicrosoftAppCredentials.MicrosoftAppPasswordKey];
                return new ConnectorClient(new Uri(incomingActivity.ServiceUrl), appId, appPassword);
            });

            // Send End of conversation as reply.
            await
                connectorClient.Conversations.SendToConversationAsync(incomingActivity.CreateReply("Ending
conversation from the skill..."));
            var endOfConversation = incomingActivity.CreateReply();
            if (order != null)
            {
                endOfConversation.Value = JsonConvert.SerializeObject(order);
            }
            endOfConversation.Type = ActivityTypes.EndOfConversation;
            endOfConversation.Code = endOfConversationCode;
            await connectorClient.Conversations.SendToConversationAsync(endOfConversation);
        }

        /// <summary>
        /// Clear the dialog stack and data bags.
        /// </summary>
        /// <param name="activity">The incoming activity to use for scoping the Conversation.Container.</param>
        internal static async Task ClearState(Activity activity)
        {
            // This Recipient null check is required for DVA manifest validation
        }
    }
}
```

```

// This recipient null check is required for PVA manifest validation.
// PVA will send an EOC activity with null Recipient.
if (activity.Recipient == null)
    return;

using (var scope = DialogModule.BeginLifetimeScope(Conversation.Container, activity))
{
    var botData = scope.Resolve<IBotData>();
    await botData.LoadAsync(default(CancellationToken));

    // Some skills might persist data between invocations.
    botData.UserData.Clear();
    botData.ConversationData.Clear();
    botData.PrivateConversationData.Clear();

    var stack = scope.Resolve<IDialogStack>();
    stack.Reset();

    await botData.FlushAsync(default(CancellationToken));
}
}
}
}

```

4. Обновите класс `MessagesController`.

V3SimpleSandwichBot\Controllers\MessagesController.cs

Обновите инструкции using.

```

using System.Threading.Tasks;
using System.Web.Http;

using Microsoft.Bot.Connector;
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Builder.FormFlow;
using System.Net.Http;
using System.Web.Http.Description;
using System.Diagnostics;
using Microsoft.Bot.Connector.SkillAuthentication;
using Newtonsoft.Json;
using Microsoft.Bot.Builder.Dialogs.Internals;
using Autofac;
using System.Threading;

```

Замените атрибут класса `BotAuthentication` на `SkillBotAuthentication`. Этот бот использует стандартное средство проверки утверждений.

```

[SkillBotAuthentication]
public class MessagesController : ApiController

```

В методе `Post` измените условие действия `message`, чтобы разрешить пользователю отменить процесс заказа из навыка. Также добавьте условие действия `endOfConversation`, которое позволит навыкам очищать состояние и освобождать ресурсы при завершении беседы по сигналу от потребителя навыков.

```

        case ActivityTypes.Message:
            if (activity.Text.ToLower().Contains("end") || activity.Text.ToLower().Contains("stop"))
            {
                await ConversationHelper.ClearState(activity);
                await ConversationHelper.EndConversation(activity, endOfConversationCode:
EndOfConversationCodes.UserCancelled);
            }
            else
            {
                await Conversation.SendAsync(activity, MakeRootDialog);
            }

            break;
        case ActivityTypes.EndOfConversation:
            Trace.TraceInformation($"EndOfConversation: {activity}");

            // Clear the dialog stack if the root bot has ended the conversation.
            await ConversationHelper.ClearState(activity);

            break;
    }
}

```

5. Измените форму заказа сандвичей.

V3SimpleSandwichBot\Sandwich.cs

Обновите инструкции using.

```

using Microsoft.Bot.Builder.FormFlow;
using Microsoft.Bot.Connector;
using System;
using System.Collections.Generic;
using System.Runtime.Remoting.Messaging;

```

Измените в форме метод `BuildForm`, чтобы этот навык мог отмечать завершение беседы.

```

public static IForm<SandwichOrder> BuildForm()
{
    // When the skill completes (OnCompletion), send an `endOfConversation` activity and include the
    finished order.
    return new FormBuilder<SandwichOrder>()
        .Message("Welcome to the simple sandwich order bot! Say 'end' or 'stop' and I'll end the
        conversation and back to the parent.")
        .OnCompletion((context, order) => ConversationHelper.EndConversation(context.Activity as
Activity, order))
        .Build();
}

```

6. Примените этот манифест для бота заказа сандвичей. В качестве идентификатора приложения конечной точки укажите идентификатор приложения бота.

V3SimpleSandwichBot\wwwroot\sandwich-bot-manifest.json

```
{
  "$schema": "https://raw.githubusercontent.com/microsoft/botframework-
sdk/master/schemas/skills/skill-manifest-2.0.0.json",
  "$id": "YourSandwichBotHandle",
  "name": "Sandwich Skill Bot",
  "version": "1.0",
  "description": "This is a sample skill for ordering a sandwich.",
  "publisherName": "Microsoft",
  "privacyUrl": "https://microsoft.com/privacy",
  "copyright": "Copyright (c) Microsoft Corporation. All rights reserved.",
  "license": "",
  "iconUrl": "https://myskill.contoso.com/icon.png",
  "tags": [
    "sample",
    "pizza"
  ],
  "endpoints": [
    {
      "name": "YourSandwichBotName",
      "protocol": "BotFrameworkV3",
      "description": "Default endpoint for the skill",
      "endpointUrl": "http://localhost:3981/api/messages",
      "msAppId": "YOUR Sandwich Bots MicrosoftAppId"
    }
  ],
  "activities": {
    "OrderSandwich": {
      "description": "Order a sandwich",
      "type": "message",
      "name": "OrderSandwich"
    }
  }
}
```

См. сведения о [схеме манифеста навыка \(skill-manifest-2.0.0.json\)](#).

## Создание корневого бота версии 4

Этот постой корневой бот выполняет роль потребителя для трех навыков и позволяет убедиться, что шаги беседы проходят в соответствии с планом. Настройте локальное выполнение бота на порту 3978.

1. В файл конфигурации добавьте идентификатор приложения и пароль корневого бота. Добавьте идентификатор приложения для каждого навыка версии 3.

V4SimpleRootBot\appsettings.json

```
{
  "MicrosoftAppId": "YOUR Root Skill Host bot's MicrosoftAppId",
  "MicrosoftAppPassword": "YOUR Root Skill Host bot's MicrosoftAppPassword",
  "SkillHostEndpoint": "http://localhost:3978/api/skills",
  "BotFrameworkSkills": [
    {
      "Id": "Echo",
      "AppId": "YOUR Echo bot's MicrosoftAppId",
      "SkillEndpoint": "http://localhost:3979/api/messages"
    },
    {
      "Id": "Pizza",
      "AppId": "YOUR Pizza bot's MicrosoftAppId",
      "SkillEndpoint": "http://localhost:3980/api/messages"
    },
    {
      "Id": "Sandwich",
      "AppId": "YOUR Sandwich bot's MicrosoftAppId",
      "SkillEndpoint": "http://localhost:3981/api/messages"
    }
  ]
}
```

## Тестирование корневого бота

Скачайте и установите последнюю версию [Bot Framework Emulator](#).

1. Скомпилируйте и запустите все четыре бота на локальном компьютере.
2. Запустите эмулятор и подключите его к корневому боту.
3. Протестируйте работу потребителя и навыков.

## Дополнительные сведения

### Аутентификация взаимодействия между ботами

Корневой бот и навык обмениваются данными по протоколу HTTP. Эта платформа использует токены носителя и идентификаторы приложения бота для идентификации каждого бота. Она использует объект конфигурации проверки подлинности для проверки заголовков проверки подлинности во входящих запросах. В конфигурацию проверки подлинности можно добавить средство проверки утверждений. Утверждения оцениваются после заголовка проверки подлинности. Код средства проверки должен создавать исключение или ошибку, чтобы отклонить запрос.

Стандартное средство проверки утверждений считывает параметр приложения `AllowedCallers` из файла конфигурации бота. Этот параметр должен содержать разделенный запятыми список идентификаторов приложений ботов, которым разрешено вызывать этот навык, или значение "\*", чтобы разрешить всем ботам вызывать навык.

Чтобы реализовать пользовательское средство проверки утверждений, реализуйте классы, производные от `AuthenticationConfiguration` и `ClaimsValidator`, а затем укажите в атрибуте `SkillBotAuthentication` ссылку на полученную конфигурацию проверки подлинности. Пример классов для средства проверки утверждений вы можете найти на шагах 3 и 4 раздела, посвященного [преобразованию эхо-бота](#).

# Преобразование бота JavaScript версии 3 в навык

27.03.2021 • 15 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье описывается, как преобразовать два примера ботов JavaScript версии 3 в навыки и создать потребитель навыков версии 4, который может обращаться к этим навыкам. Чтобы преобразовать бот .NET версии 3 в навык, воспользуйтесь [этой инструкцией](#). Сведения о переносе бота JavaScript из версии 3 в версию 4 см. в [этой статье](#).

## Предварительные требования

- Visual Studio Code.
- Node.js.
- Подписка Azure. Если у вас еще нет подписки Azure, [создайте бесплатную учетную запись](#), прежде чем начинать работу.
- Чтобы протестировать навыки, вам потребуется эмулятор Bot-платформы и локальные копии программы-роботы:
  - Навык эхо на JavaScript версии 3: [Skills/v3-skill-bot](#).
  - Навык заказа версии 3 для JavaScript: [Skills/v3-booking-bot-skill](#).
  - Пример потребителя навыка на JavaScript версии 4: [Skills/v4-root-bot](#).

## Сведения о ботах

В этой статье каждый бот версии 3 предназначен для использования в качестве навыка. Предоставляется также потребитель навыка версии 4, который позволяет протестировать боты, преобразованные в навыки.

- Программа `v3-skill-bot` возвращает все полученные сообщения. Работая как навык, она *завершается* при получении сообщения `end` или `stop`. Бот, который мы будем преобразовывать, основан на минимальном примере бота версии 3.
- С помощью бота `v3-booking-bot-skill` пользователь может забронировать авиабилет или номер в гостинице. Работая как навык, бот после завершения отправляет собранные сведения обратно в родительский объект.

Кроме того, в примере потребителя навыков версии 4 `v4-root-bot` показано, как можно использовать и тестировать навыки.

Чтобы использовать потребитель навыков для тестирования навыков, все 3 бота должны выполняться одновременно. Боты можно тестировать локально с помощью Bot Framework Emulator, где каждый бот использует свой локальный порт.

## Создание ресурсов Azure для ботов

Для проверки подлинности в сценарии взаимодействия ботов требуется предоставить каждому из этих ботов действительный идентификатор приложения и пароль.

1. Создайте нужное количество регистраций каналов для ботов.

2. Запишите идентификатор приложения и пароль для каждой из них.

## Преобразование ленты v3

Чтобы преобразовать существующий бот в бот-навык, нужно выполнить лишь несколько шагов, которые описаны в следующих разделах. Более подробные сведения о навыках см. [здесь](#).

- Обновите файл `.env` бота, включив в него идентификатор приложения и пароль, а также добавив свойство `root bot app ID`.
- Добавьте проверку утверждений. Таким образом вы сможете ограничить доступ к навыку, предоставляя его только пользователям и корневому боту. Дополнительные сведения о стандартных и пользовательских проверках утверждений см. [здесь](#).
- Измените контроллер сообщений бота, чтобы обрабатывать действия `endOfConversation` от корневого бота.
- Измените код бота, чтобы он возвращал действие `endOfConversation` после завершения работы навыка.
- При каждом завершении работы навыка, если он еще поддерживает беседу или некоторые ресурсы, ему необходимо очистить состояние беседы и освободить ресурсы.
- При необходимости добавьте файл манифеста. Так как потребитель навыка не всегда имеет доступ к коду этого навыка, опишите в манифесте все действия, которые ваш навык умеет получать и создавать, все его входные и выходные параметры, а также конечные точки. Текущая схема манифеста хранится в файле [skill-manifest-2.0.0.json](#).

## Преобразование эхо-бота

Пример [Skills/v3-skill-bot](#) содержит эхо-бот версии 3, преобразованный в базовый навык.

1. Создайте простой проект бота для JavaScript версии 3 и импортируйте необходимые модули.

`v3-skill-bot/app.js`

```
const restify = require('restify');
const builder = require('botbuilder');
require('dotenv').config();
```

2. Настройте локальное выполнение бота на порту 3979.

`v3-skill-bot/app.js`

```
// Setup Restify Server
const server = restify.createServer();
server.listen(process.env.port || process.env.PORT || 3979, function () {
    console.log('%s listening to %s', server.name, server.url);
});
```

3. В файл конфигурации добавьте идентификатор приложения и пароль эхо-бота. Также добавьте свойство `ROOT_BOT_APP_ID` со значением идентификатора приложения простого корневого бота.

`v3-skill-bot/.env`

```
# Bot Framework Credentials
```

```
MICROSOFT_APP_ID=
MICROSOFT_APP_PASSWORD=
ROOT_BOT_APP_ID=
```

4. Создайте для бота соединитель чата. В нашем примере используется стандартная конфигурация проверки подлинности. Присвойте параметру `enableSkills` значение `true`, чтобы этот бот мог использоваться в качестве навыка. `allowedCallers` содержит массив идентификаторов приложений для всех ботов, которым разрешено использовать этот навык. Если первое значение этого массива равно "\*", то любой робот может использовать этот навык.

#### v3-skill-bot/app.js

```
// Create chat connector for communicating with the Bot Framework Service
const connector = new builder.ChatConnector({
    appId: process.env.MICROSOFT_APP_ID,
    appPassword: process.env.MICROSOFT_APP_PASSWORD,
    enableSkills: true,
    allowedCallers: [process.env.ROOT_BOT_APP_ID]
});
```

5. Обновите обработчик сообщений, чтобы он отправлял действие `endOfConversation` при завершении работы навыка пользователем.

#### v3-skill-bot/app.js

```
// Create your bot with a function to receive messages from the user
const bot = new builder.UniversalBot(connector, function (session) {
    switch (session.message.text.toLowerCase()) {
        case 'end':
        case 'stop':
            session.endConversation();
            break;
        default:
            session.send("Echo (JS V3) You said: %s", session.message.text);
            session.send('Say "end" or "stop" and I\'ll end the conversation and back to the
parent.');
    }
}).set('storage', inMemoryStorage); // Register in memory storage
```

6. Поскольку этот бот не поддерживает состояние беседы и не создает ресурсы для беседы, в нем не требуется обрабатывать действия `endOfConversation`, которые он будет получать от потребителя навыка.
7. Примените этот манифест для эхо-бота. В качестве идентификатора приложения конечной точки укажите идентификатор приложения бота.

#### v3-skill-bot/manifest/v3-skill-bot-manifest.json

```
{
    "$schema": "https://schemas.botframework.com/schemas/skills/skill-manifest-2.0.0.json",
    "$id": "v3-skill-bot",
    "name": "Echo Skill bot",
    "version": "1.0",
    "description": "This is a sample echo skill",
    "publisherName": "Microsoft",
    "privacyUrl": "https://echoskillbot.contoso.com/privacy.html",
    "copyright": "Copyright (c) Microsoft Corporation. All rights reserved.",
    "license": "",
    "iconUrl": "https://echoskillbot.contoso.com/icon.png",
    "tags": [
        "sample",
        "echo"
    ],
    "endpoints": [
        {
            "name": "default",
            "protocol": "BotFrameworkV3",
            "description": "Default endpoint for the skill",
            "endpointUrl": "http://echoskillbot.contoso.com/api/messages",
            "msAppId": "00000000-0000-0000-0000-000000000000"
        }
    ]
}
```

См. сведения о [схеме манифеста навыка \(skill-manifest-2.0.0.json\)](#).

## Преобразование бота бронирования

Пример [Skills/v3-booking-bot-skill](#) содержит бот бронирования версии 3, преобразованный в базовый навык. До преобразования этот бот был очень похож на пример [core-MultiDialogs](#) версии 3.

- Импортируйте необходимые модули.

### v3-booking-bot-skill/app.js

```
require('dotenv').config();

var builder = require('botbuilder');
var restify = require('restify');
const skills = require('botbuilder/skills-validator');
const { allowedCallersClaimsValidator } = require('./allowedCallersClaimsValidator');
```

- Настройте локальное выполнение бота на порту 3980.

### v3-booking-bot-skill/app.js

```
// Setup Restify Server
var server = restify.createServer();
server.listen(process.env.port || process.env.PORT || 3980, function () {
    console.log('%s listening to %s', server.name, server.url);
});
```

- В файл конфигурации добавьте идентификатор приложения и пароль бота бронирования. Также добавьте свойство `ROOT_BOT_APP_ID` со значением идентификатора приложения простого корневого бота.

### v3-booking-bot-skill/.env

```
# Bot Framework Credentials
```

```
MICROSOFT_APP_ID=
MICROSOFT_APP_PASSWORD=
ROOT_BOT_APP_ID=
```

4. Создайте для бота соединитель чата. В нашем примере используется пользовательская конфигурация проверки подлинности. Присвойте параметру `enableSkills` значение `true`, чтобы этот бот мог использоваться в качестве навыка. `authConfiguration` содержит объект пользовательской конфигурации проверки подлинности, который будет использоваться для аутентификации и проверки утверждений.

#### v3-booking-bot-skill/app.js

```
// Create chat bot and listen to messages
var connector = new builder.ChatConnector({
    appId: process.env.MICROSOFT_APP_ID,
    appPassword: process.env.MICROSOFT_APP_PASSWORD,
    enableSkills: true,
    authConfiguration: new skills.AuthenticationConfiguration([], allowedCallersClaimsValidator)
});
```

#### v3-booking-bot-skill/allowedCallersClaimsValidator.js

Здесь реализовано пользовательское средство проверки утверждений, которое создает ошибку при неудачной проверке.

```

const skills = require('botbuilder/skills-validator');
const path = require('path');

// Import required bot configuration.
const ENV_FILE = path.join(__dirname, '.env');
require('dotenv').config({ path: ENV_FILE });

// Load the AppIds for the configured callers (we will only allow responses from skills we have
// configured).
// process.env.AllowedCallers is the list of parent bot IDs that are allowed to access the skill
// to add a new parent bot simply go to the .env file and add
// the parent bot's Microsoft AppId to the list under AllowedCallers, e.g.:
// AllowedCallers=195bd793-4319-4a84-a800-386770c058b2,38c74e7a-3d01-4295-8e66-43dd358920f8
const allowedCallers = [process.env.ROOT_BOT_APP_ID]; // process.env.AllowedCallers ?
process.env.AllowedCallers.split(',') : undefined;

/**
 * Sample claims validator that loads an allowed list from configuration if present
 * and checks that requests are coming from allowed parent bots.
 * @param claims An array of Claims decoded from the HTTP request's auth header
 */
const allowedCallersClaimsValidator = async (claims) => {
    if (!allowedCallers || allowedCallers.length == 0) {
        throw new Error(`DefaultAuthenticationConfiguration allowedCallers must contain at least one
element of '*' or valid MicrosoftAppId(s).`);
    }
    if (!claims || claims.length < 1) {
        throw new Error(`DefaultAuthenticationConfiguration.validateClaims.claims parameter must
contain at least one element.`);
    }
    // If allowedCallers is undefined we allow all calls
    // If allowedCallers contains '*' we allow all callers
    if (skills.SkillValidation.isSkillClaim(claims)) {

        if(allowedCallers[0] === '*') {
            return;
        }
        // Check that the appId claim in the skill request is in the list of skills configured for
this bot.
        const appId = skills.JwtTokenValidation.getappIdFromClaims(claims);
        if (allowedCallers.includes(appId)) {
            return;
        }
        throw new Error(`Received a request from a bot with an app ID of "${appId} ". To enable
requests from this caller, add the app ID to your configuration file.`);
    }
    throw new Error(`DefaultAuthenticationConfiguration.validateClaims called without a Skill claim
in claims.`);
};

module.exports.allowedCallersClaimsValidator = allowedCallersClaimsValidator;

```

5. Обновите обработчик сообщений, чтобы он отправлял действие `endOfConversation` при завершении навыка. Обратите внимание, что `session.endConversation()` очищает состояние беседы, а не просто отправляет действие `endOfConversation`.

### v3-booking-bot-skill/app.js

Реализуйте вспомогательную функцию, которая будет устанавливать в действии `endOfConversation` значения свойств `code` и `value`, а также очищать состояние беседы. Если бот управляет еще какими-то ресурсами для этой беседы, все их следует здесь освободить.

```
// Code enum can be found here: https://aka.ms/codeEnum
function endConversation(session, value = null, code = null) {
    session.send('Ending conversation from the skill...');
    // Send endOfConversation with custom code and values
    const msg = {
        value,
        code,
        type: 'endOfConversation'
    };
    session.send(msg);
    // Call endConversation() to clear state
    session.endConversation();
}
```

Когда пользователь завершает процесс, вспомогательный метод должен завершить работу навыка и вернуть собранные от пользователя данные.

```
var bot = new builder.UniversalBot(connector, [
    function (session) {

},
// Dialog has ended
function(session, result) {
    endConversation(session, result, 'completedSuccessfully');
}
]).set('storage', inMemoryStorage); // Register in memory storage
```

Вспомогательный метод так само будет вызываться, если пользователь завершит процесс досрочно.

```
bot.recognizer({
    recognize: function (context, done) {
        var intent = { score: 0.0 };
        if (context.message.text) {
            switch (context.message.text.toLowerCase()) {
                case 'help':
                    intent = { score: 1.0, intent: 'Help' };
                    break;
                case 'end':
                    intent = { score: 1.0, intent: 'End' };
                    break;
            }
        }
        done(null, intent);
    }
});
```

```
// Add global endConversation() action bound to the 'Goodbye' intent
bot.endConversationAction('endAction', "Ok... See you later.", { matches: 'End' });
```

- Если навык отменяется потребителем навыка, потребитель отправляет действие `endOfConversation`. Обрабатывайте это действие и освобождайте все ресурсы, связанные с этой беседой.

[v3-booking-bot-skill/app.js](#)

```
// Listen for endOfConversation activities from other sources
bot.on('endOfConversation', (message) => {
    bot.loadSession(message.address, (err, session) => {
        endConversation(session, null, 'completedSuccessfully');
    });
})
```

7. Примените этот манифест для бота бронирования. В качестве идентификатора приложения конечной точки укажите идентификатор приложения бота.

v3-booking-bot-skill/manifest/v3-booking-bot-skill-manifest.json

```
{
    "$schema": "https://schemas.botframework.com/schemas/skills/skill-manifest-2.0.0.json",
    "$id": "v3-booking-bot-skill",
    "name": "Booking Skill bot",
    "version": "1.0",
    "description": "This is a sample booking skill",
    "publisherName": "Microsoft",
    "privacyUrl": "https://bookingskillbot.contoso.com/privacy.html",
    "copyright": "Copyright (c) Microsoft Corporation. All rights reserved.",
    "license": "",
    "iconUrl": "https://bookingskillbot.contoso.com/icon.png",
    "tags": [
        "sample",
        "echo"
    ],
    "endpoints": [
        {
            "name": "default",
            "protocol": "BotFrameworkV3",
            "description": "Default endpoint for the skill",
            "endpointUrl": "http://bookingskillbot.contoso.com/api/messages",
            "msAppId": "00000000-0000-0000-0000-000000000000"
        }
    ]
}
```

См. сведения о [схеме манифеста навыка \(skill-manifest-2.0.0.json\)](#).

## Создание корневого бота версии 4

Этот простой корневой бот выполняет роль потребителя для двух навыков и позволяет убедиться, что шаги беседы проходят в соответствии с планом. Настройте локальное выполнение бота на порту 3978.

1. В файл конфигурации добавьте идентификатор приложения и пароль корневого бота. Добавьте идентификатор приложения для каждого навыка версии 3.

v4-root-bot/.env

```
MicrosoftAppId=
MicrosoftAppPassword=
SkillHostEndpoint=http://localhost:3978/api/skills

SkillSimpleId=v3-skill-bot
SkillSimpleAppId=
SkillSimpleEndpoint=http://localhost:3979/api/messages

SkillBookingId=v3-booking-bot-skill
SkillBookingAppId=
SkillBookingEndpoint=http://localhost:3980/api/messages
```

# Тестирование корневого бота

Скачайте и установите последнюю версию [Bot Framework Emulator](#).

1. Запустите все три бота на локальном компьютере.
2. Запустите эмулятор и подключите его к корневому боту.
3. Протестируйте работу потребителя и навыков.

Начиная с версии 4.11 вам не требуется идентификатор приложения и пароль для проверки уровня квалификации и квалификации потребителя локально в эмуляторе. Подписка Azure по-прежнему необходима для развертывания вашего навыка в Azure.

## Дополнительные сведения

### Аутентификация взаимодействия между ботами

Корневой бот и навык обмениваются данными по протоколу HTTP. Эта платформа использует токены носителя и идентификаторы приложения бота для идентификации каждого бота. Она использует объект конфигурации проверки подлинности для проверки заголовков проверки подлинности во входящих запросах. Необходимо добавить проверяющий элемент управления утверждения в конфигурацию проверки подлинности. Утверждения оцениваются после заголовка проверки подлинности. Код средства проверки должен создавать исключение или ошибку, чтобы отклонить запрос.

При создании соединителя чата добавьте в параметр настроек свойство `allowedCallers` ИЛИ `authConfiguration`, чтобы включить проверку подлинности между ботами.

Свойство `allowedCallers` применяется стандартным средством проверки утверждений для этого соединителя чата. В качестве значения оно должно содержать массив идентификаторов приложений тех ботов, которым разрешено использовать этот навык. Присвойте первому элементу значение "\*", чтобы разрешить всем роботам вызывать навык.

Чтобы применить пользовательскую функцию проверки, присвойте ее значение полю `authConfiguration`. Эта функция должна принимать массив объектов утверждений и создавать ошибку, если проверка завершается неудачно. Пример средства проверки утверждений вы можете найти на шаге 4 раздела, посвященного [преобразованию бота бронирования](#).

# REST API для Bot Framework

27.10.2020 • 2 minutes to read • [Edit Online](#)

Большинство программы-роботыных платформ создаются с помощью пакета SDK для Bot Framework, который организует робот и обрабатывает все беседы. Альтернативой использованию пакета SDK является отправка сообщений непосредственно в Bot с помощью REST API.

## Создание бота

При написании кода с помощью API-интерфейсов RESTFUL платформы Bot можно отправлять и получать сообщения с пользователями по любому каналу, настроенному в регистрации службы Azure Bot в Bot.

### TIP

Bot Framework предоставляет клиентские библиотеки, которые могут использоваться для разработки ботов на C# или Node.js. Чтобы создать бот на языке C#, используйте [пакет SDK Bot Framework для C#](#). Чтобы создать бот на языке Node.js, используйте [пакет SDK Bot Framework для Node.js](#).

Дополнительные сведения о создании программы-роботы с помощью службы см. в документации по службе [Azure Bot](#).

## Создание клиента прямой линии

Большинство таких каналов, как Facebook, Teams или временного резерва, предоставляют клиентам, но с прямой линией вы можете использовать собственное клиентское приложение для взаимодействия с программой-роботом. [Веб-чат](#) — это пример прямого клиента с открытым кодом, который можно использовать как есть или изменять или изменять при создании собственного клиента. API Direct Line реализует механизм проверки подлинности, который использует стандартные шаблоны секрета или маркера и предоставляет стабильную схему проверки подлинности, которая продолжит работать даже при изменении версии протокола бота. Дополнительные сведения о взаимодействии клиента и бота с использованием API Direct Line см. в разделе [Основные понятия](#).

Клиенты прямой строки могут находиться в разных языках и расположениях (например, в классическом приложении, а не на веб-странице). См. сведения о [Direct Line](#).

# Основные понятия — API соединителя Bot

27.03.2021 • 4 minutes to read • [Edit Online](#)

Платформа Bot и служба Azure Bot позволяют роботам обмениваться данными с пользователями в группах, Facebook и других. Каналы предоставляются в двух видах: как служба в составе службы Azure Bot и как библиотеки адаптеров для использования с пакетом SDK для Bot Framework. В этой статье рассматриваются стандартизованные каналы, включенные в службу Azure Bot.

## Каналы Bot Framework

Каналы ленты Bot позволяют роботам обмениваться сообщениями с каналами, настроенными на [портале Azure](#). В нем используются стандартные отраслевые форматы REST и JSON через HTTPS и обеспечивается проверка подлинности с помощью токенов носителя JWT. Подробные сведения о том, как использовать службу Bot Connector, см. в статье [Проверка подлинности](#) и в остальных статьях этого раздела.

### Действие

Служба соединителя обменивается данными между Bot и Channel (пользователь) путем передачи объекта [действия](#). Наиболее распространенным типом действия является **сообщение**, однако существуют другие типы действий, которые можно использовать для передачи данных различных типов в бот или канал. Дополнительные сведения о действиях в службе Bot Connector см. в статье [Общие сведения о действиях](#).

## Аутентификация

Службы Bot Framework используют для проверки подлинности токены носителя JWT. Подробные сведения о способах проверки подлинности исходящих запросов, отправляемых ботом в Bot Framework, о способах проверки подлинности входящих запросов, которые ваш бот получает от Bot Framework, и многое другое см. в статье [Проверка подлинности](#).

## Клиентские библиотеки

Bot Framework предоставляет клиентские библиотеки, которые могут использоваться для разработки ботов на C#, JavaScript или Python.

- Чтобы создать бот на языке C#, используйте [пакет SDK Bot Framework для C#](#).
- Чтобы создать бот на языке Node.js, используйте [пакет SDK Bot Framework для Node.js](#).

Помимо упрощенных вызовов API REST в Bot Framework, каждый пакет SDK Bot Framework также предоставляет полнофункциональную систему для создания диалогов с поддержкой логики беседы, встроенными запросами простых сведений (ответы "Да" или "Нет", строки, числа и возможность выбора из списка), со встроенной поддержкой мощных платформ ИИ (например, [LUIS](#)) и многими другими функциями.

### NOTE

Вместо пакета SDK вы можете создать собственную клиентскую библиотеку на любом языке, используя [файл Swagger для службы Bot Connector](#) или прямой вызов API REST из кода приложения.

## Служба Состояние бота

Служба состояний Microsoft Bot Framework устарела с 30 марта 2018 г. Ранее боты, созданные на основе службы Azure Bot или пакета SDK для Bot Builder, по умолчанию использовали подключение к этой службе, размещенной корпорацией Майкрософт, для хранения данных о состоянии бота. Теперь эти боты нужно обновить, чтобы они использовали собственное хранилище состояний.

## Дополнительные ресурсы

Дополнительные сведения о создании программы-роботы с помощью службы соединителя см. в статьях по всему этому разделу, начиная с [проверки подлинности](#). Если у вас возникли проблемы или есть предложения, касающиеся службы соединителя, см. раздел [Поддержка](#) для получения списка доступных ресурсов.

# Создание программы-робота с помощью службы соединителя Bot с помощью API соединителя Bot

21.09.2020 • 6 minutes to read • [Edit Online](#)

Служба Bot Connector позволяет боту обмениваться сообщениями с каналами, настроенными на [портале Azure](#), используя стандартные отраслевые форматы REST и JSON по протоколу HTTPS. В этом руководстве подробно объясняется, как получить маркер доступа от Bot Framework и использовать службу Bot Connector для обмена сообщениями с пользователем.

## Получение маркера доступа

### IMPORTANT

Зарегистрируйте бот в Bot Framework (если еще не сделали этого), чтобы получить идентификатор приложения и пароль. Они потребуются для получения маркера доступа.

Для взаимодействия со службой Bot Connector необходимо указать маркер доступа в заголовке `Authorization` каждого запроса API, используя следующий формат:

```
Authorization: Bearer ACCESS_TOKEN
```

Вы можете получить маркер доступа для вашего бота, выполнив запрос API.

### Запрос

Чтобы запросить маркер доступа для аутентификации запросов к службе Bot Connector, выполните приведенный ниже запрос, заменив **MICROSOFT-APP-ID** и **MICROSOFT-APP-PASSWORD** идентификатором приложения и паролем, полученными при [регистрации](#) бота в Bot Framework.

```
POST https://login.microsoftonline.com/botframework.com/oauth2/v2.0/token
Host: login.microsoftonline.com
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials&client_id=MICROSOFT-APP-ID&client_secret=MICROSOFT-APP-
PASSWORD&scope=https%3A%2F%2Fapi.botframework.com%2F.default
```

### Ответ

Если запрос выполнен успешно, вы получите ответ HTTP 200, в котором указан маркер доступа и сведения о его сроке действия.

```
{
  "token_type": "Bearer",
  "expires_in": 3600,
  "ext_expires_in": 3600,
  "access_token": "eyJhbGciOiJIUzI1Ni..."}
```

**TIP**

Дополнительные сведения об аутентификации в службе Bot Connector см. в [этой статье](#).

## Обмен сообщениями с пользователем

Диалог — это последовательность сообщений, передаваемых между пользователем и ботом.

### Получение сообщения от пользователя

Когда пользователь отправляет сообщение, Bot Framework Connector отсылает запрос POST к конечной точке, указанной вами при [регистрации](#) бота. Текст запроса — это объект [Действие](#). В представленном ниже примере приведен текст запроса, который бот получает, когда пользователь отправляет ему простое сообщение.

```
{  
    "type": "message",  
    "id": "bf3cc9a2f5de...",  
    "timestamp": "2016-10-19T20:17:52.2891902Z",  
    "serviceUrl": "https://smba.trafficmanager.net/apis",  
    "channelId": "channel's name/id",  
    "from": {  
        "id": "1234abcd",  
        "name": "user's name"  
    },  
    "conversation": {  
        "id": "abcd1234",  
        "name": "conversation's name"  
    },  
    "recipient": {  
        "id": "12345678",  
        "name": "bot's name"  
    },  
    "text": "Haircut on Saturday"  
}
```

### Ответ на сообщение пользователя

Когда конечная точка бота получит запрос `POST`, который представляет собой сообщение от пользователя (т. е. `type` = `message`), используйте сведения из этого запроса, чтобы создать объект [Действие](#) для ответа.

1. Задайте свойство `conversation` с учетом содержимого свойства `conversation` в сообщении пользователя.
2. Укажите свойство `from` с учетом содержимого свойства `recipient` в сообщении пользователя.
3. Задайте свойство `recipient` с учетом содержимого свойства `from` в сообщении пользователя.
4. Укажите свойства `text` и `attachments` соответствующим образом.

Используйте свойство `serviceUrl` во входящем запросе, чтобы [определить базовый URI](#), который бот будет применять для отправки ответа.

Чтобы отправить ответ, отправьте запрос `POST` с объектом `Activity` по адресу `/v3/conversations/{conversationId}/activities/{activityId}`, как показано в примере ниже. Текст этого запроса — это объект `Activity`, в котором пользователю предлагается выбрать доступное время встречи.

```
POST https://smbs.trafficmanager.net/apis/v3/conversations/abcd1234/activities/bf3cc9a2f5de...
Authorization: Bearer eyJhbGciOiJIUzI1Ni...
Content-Type: application/json
```

```
{
  "type": "message",
  "from": {
    "id": "12345678",
    "name": "bot's name"
  },
  "conversation": {
    "id": "abcd1234",
    "name": "conversation's name"
  },
  "recipient": {
    "id": "1234abcd",
    "name": "user's name"
  },
  "text": "I have these times available:",
  "replyToId": "bf3cc9a2f5de..."
}
```

В этом примере запрос `https://smbs.trafficmanager.net/apis` представляет базовый URI. Базовый URI для запросов, отправляемых вашим ботом, может отличаться. Дополнительные сведения о настройке базового URI см. в статье [Справочник по API](#).

#### IMPORTANT

Как показано в этом примере, заголовок `Authorization` каждого отправляемого вами запроса API, должен содержать слово `Bearer`, после которого указывается маркер доступа, [полученный от Bot Framework](#).

Чтобы отправить другое сообщение, в котором пользователь может нажатием кнопки выбрать доступное время встречи, отправьте другой запрос `POST` к той же конечной точке:

```
POST https://smbs.trafficmanager.net/apis/v3/conversations/abcd1234/activities/bf3cc9a2f5de...
Authorization: Bearer eyJhbGciOiJIUzI1Ni...
Content-Type: application/json
```

```
{  
    "type": "message",  
    "from": {  
        "id": "12345678",  
        "name": "bot's name"  
    },  
    "conversation": {  
        "id": "abcd1234",  
        "name": "conversation's name"  
    },  
    "recipient": {  
        "id": "1234abcd",  
        "name": "user's name"  
    },  
    "attachmentLayout": "list",  
    "attachments": [  
        {  
            "contentType": "application/vnd.microsoft.card.thumbnail",  
            "content": {  
                "buttons": [  
                    {  
                        "type": "imBack",  
                        "title": "10:30",  
                        "value": "10:30"  
                    },  
                    {  
                        "type": "imBack",  
                        "title": "11:30",  
                        "value": "11:30"  
                    },  
                    {  
                        "type": "openUrl",  
                        "title": "See more",  
                        "value": "http://www.contososalon.com/scheduling"  
                    }  
                ]  
            }  
        ],  
        "replyToId": "bf3cc9a2f5de..."  
    }  
}
```

## Дальнейшие действия

При работе с этим руководством вы получили маркер доступа от Bot Framework и использовали службу Bot Connector для обмена сообщениями с пользователем. Для тестирования и отладки бота можно использовать [Bot Framework Emulator](#). Чтобы предоставить доступ к боту другим пользователям, нужно [настроить](#) его для работы с одним или несколькими каналами.

# Справочник по API — соединитель Bot

27.03.2021 • 57 minutes to read • [Edit Online](#)

## NOTE

REST API не является эквивалентом пакета SDK. REST API позволяет установить стандартную связь REST, однако предпочтительный способ взаимодействия с Bot Framework — это SDK.

На платформе Bot Framework есть служба Bot Connector, которая позволяет боту обмениваться сообщениями с каналами, настроенными на портале Bot Framework. Эта служба использует стандартные отраслевые протоколы REST и JSON через HTTPS.

## Базовый универсальный код ресурса

Когда пользователь отправляет сообщение боту, входящий запрос содержит объект [Activity](#) со свойством `serviceUrl`, которое указывает конечную точку, в которую бот должен отправить ответ. Для доступа к службе Bot Connector используйте значение `serviceUrl` как базовый URI для запросов API.

Например, предположим, что бот получает следующее действие, когда пользователь отправляет сообщение боту.

```
{  
    "type": "message",  
    "id": "bf3cc9a2f5de...",  
    "timestamp": "2016-10-19T20:17:52.289190Z",  
    "serviceUrl": "https://smba.trafficmanager.net/apis",  
    "channelId": "channel's name/id",  
    "from": {  
        "id": "1234abcd",  
        "name": "user's name"  
    },  
    "conversation": {  
        "id": "abcd1234",  
        "name": "conversation's name"  
    },  
    "recipient": {  
        "id": "12345678",  
        "name": "bot's name"  
    },  
    "text": "Haircut on Saturday"  
}
```

Свойство `serviceUrl` в сообщении пользователя указывает, что боту следует отправить ответ в конечную точку `https://smba.trafficmanager.net/apis`. Это будет базовым URI для последующих запросов, которые бот будет выдавать в рамках этого общения. Если боту будет необходимо отправить пользователю упреждающее сообщение, не забудьте сохранить значение `serviceUrl`.

В следующем примере показан запрос, который бот выдает в ответ на сообщение пользователя.

```
POST https://smba.trafficmanager.net/apis/v3/conversations/abcd1234/activities/bf3cc9a2f5de...  
Authorization: Bearer eyJhbGciOiJIUzI1Ni...  
Content-Type: application/json
```

```
{  
    "type": "message",  
    "from": {  
        "id": "12345678",  
        "name": "bot's name"  
    },  
    "conversation": {  
        "id": "abcd1234",  
        "name": "conversation's name"  
    },  
    "recipient": {  
        "id": "1234abcd",  
        "name": "user's name"  
    },  
    "text": "I have several times available on Saturday!",  
    "replyToId": "bf3cc9a2f5de..."  
}
```

## Заголовки

### Заголовки запросов

Помимо стандартных заголовков HTTP-запроса каждый выдаваемый запрос API должен включать заголовок `Authorization`, который указывает маркер доступа для проверки подлинности бота.

Используйте для заголовка `Authorization` следующий формат:

```
Authorization: Bearer ACCESS_TOKEN
```

Дополнительные сведения о том, как получить маркер доступа для бота, см. в разделе [Authenticate requests from your bot to the Bot Connector service](#) (Проверка подлинности запросов от бота в службу Bot Connector).

### Заголовки ответов

Помимо стандартных заголовков HTTP-ответа каждый ответ будет содержать заголовок `X-Correlating-OperationId`. Значение этого заголовка — это идентификатор, который соответствует записи журнала Bot Framework, содержащей сведения о запросе. Когда появляется сообщение об ошибке, следует фиксировать значение этого заголовка. Если вы не можете самостоятельно устранить проблему, при отправке сообщения об ошибке включите это значение в сведения, предоставляемые в службу поддержки.

## Коды состояния HTTP

[Код состояния HTTP](#), который возвращается с каждым ответом, указывает результат соответствующего запроса.

#### NOTE

В следующей таблице описаны наиболее распространенные коды состояния HTTP. Канал создает некоторые ошибки. Для получения дополнительных сведений может потребоваться ознакомиться с документацией для разработчиков канала.

HTTP STATUS CODE (КОД СОСТОЯНИЯ HTTP)	ЗНАЧЕНИЕ
200	Запрос выполнен успешно.

HTTP STATUS CODE (КОД СОСТОЯНИЯ HTTP)	ЗНАЧЕНИЕ
201	Запрос выполнен успешно.
202	Запрос был принят для обработки.
204	Запрос успешно выполнен, но содержимое не было возвращено.
400	Запрос неправильный или имеет недопустимый формат.
401	Программа Bot еще не прошла проверку подлинности.
403	Bot не имеет разрешения на выполнение запрошенной операции.
404	Запрошенный ресурс не найден.
405	Этот протокол не поддерживает запрашиваемую операцию.
500	Произошла внутренняя ошибка сервера.
503	Служба временно недоступна.

## ошибки

Любой ответ, указывающий код состояния HTTP в диапазоне 4xx или 5xx, будет содержать в тексте ответа объект [ErrorResponse](#), который предоставляет сведения об ошибке. Если вы получите сообщение об ошибке в диапазоне 4xx, проверьте объект [ErrorResponse](#), чтобы определить причину ошибки и устранить проблему, прежде чем повторно отправлять запрос.

## Операции диалога

Используйте следующие операции для создания общения, отправки сообщений (действий) и управления содержимым общения.

ОПЕРАЦИЯ	ОПИСАНИЕ
<a href="#">Создать общение</a>	Создает новое общение.
<a href="#">Удалить действие</a>	Удаляет существующее действие.
<a href="#">Удалить участника диалога</a>	Удаляет участника из диалога.
<a href="#">Получить участников действия</a>	Возвращает участников указанного действия в указанном общении.
<a href="#">Получить участника общения</a>	Возвращает сведения об участнике диалога.
<a href="#">Получить участников общения</a>	Возвращает участников указанного общения.

ОПЕРАЦИЯ	ОПИСАНИЕ
Получение участников беседы по страницам	Возвращает участников указанной беседы по одной странице за раз.
Получить диалоги	Получает список диалогов, в которых бот принимал участие.
Ответить на действие	Отправляет действие (сообщение) в указанное общение как ответ на указанное действие.
Отправить журнал бесед	Отправляет расшифровку последних действий в диалог.
Отправить в общение	Отправляет действие (сообщение) в конец указанного общения.
Обновить действие	Обновляет существующее действие.
Отправить вложение на канал	Передает вложение непосредственно в хранилище BLOB-объектов канала.

### Создать общение

Создает новое общение.

```
POST /v3/conversations
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
Текст запроса	Объект ConversationParameters
Возвращает	Объект ConversationResourceResponse

### Удалить действие

Некоторые каналы позволяют удалить существующее действие. При успешном выполнении эта операция удаляет указанное действие из указанного общения.

```
DELETE /v3/conversations/{conversationId}/activities/{activityId}
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
Текст запроса	Недоступно
Возвращает	Код состояния HTTP, который указывает результат операции. В тексте ответа ничего не указано.

### Удалить участника диалога

Удаляет участника из диалога. Если этот участник был последним в диалоге, диалог также будет удален.

```
DELETE /v3/conversations/{conversationId}/members/{memberId}
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
<b>Текст запроса</b>	Недоступно
<b>Возвращает</b>	Код состояния HTTP, который указывает результат операции. В тексте ответа ничего не указано.

### Получить участников действия

Возвращает участников указанного действия в указанном общении.

```
GET /v3/conversations/{conversationId}/activities/{activityId}/members
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
<b>Текст запроса</b>	Недоступно
<b>Возвращает</b>	Массив объектов <a href="#">ChannelAccount</a>

### Получить диалоги

Получает список диалогов, в которых бот принимал участие.

```
GET /v3/conversations?continuationToken={continuationToken}
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
<b>Текст запроса</b>	Недоступно
<b>Возвращает</b>	Объект <a href="#">ConversationsResult</a>

### Получить участника общения

Возвращает сведения об определенном участнике определенного диалога.

```
GET /v3/conversations/{conversationId}/members/{memberId}
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
<b>Текст запроса</b>	Недоступно
<b>Возвращает</b>	Объект <a href="#">ChannelAccount</a> для участника.

### Получить участников общения

Возвращает участников указанного общения.

```
GET /v3/conversations/{conversationId}/members
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
<b>Текст запроса</b>	Недоступно

СОДЕРЖИМОЕ	ОПИСАНИЕ
<b>Возвращает</b>	Массив объектов <a href="#">ChannelAccount</a> для участников диалога.

### Получение участников беседы по страницам

Возвращает участников указанной беседы по одной странице за раз.

```
GET /v3/conversations/{conversationId}/pagedmembers?pageSize={pageSize}&continuationToken={continuationToken}
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
<b>Текст запроса</b>	Недоступно
<b>Возвращает</b>	Объект <a href="#">PagedMembersResult</a>

### Ответить на действие

Отправляет действие (сообщение) в указанное общение как ответ на указанное действие. Действие добавляется в качестве ответа на другое действие, если канал это поддерживает. Если канал не поддерживает вложенные ответы, то эта операция ведет себя как [Отправить в общение](#).

```
POST /v3/conversations/{conversationId}/activities/{activityId}
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
<b>Текст запроса</b>	Объект <a href="#">Activity</a>
<b>Возвращает</b>	Объект <a href="#">ResourceResponse</a>

### Отправить журнал бесед

Отправляет расшифровку последних действий в диалог, поэтому клиент может преобразовать их для просмотра.

```
POST /v3/conversations/{conversationId}/activities/history
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
<b>Текст запроса</b>	Объект <a href="#">Transcript</a> .
<b>Возвращает</b>	Объект <a href="#">ResourceResponse</a> .

### Отправить в общение

Отправляет действие (сообщение) в указанное общение. Действие будет добавлено в конец общения в соответствии с меткой времени или семантикой канала. Чтобы ответить на конкретное сообщение внутри общения, вместо этого используйте [Ответить на действие](#).

```
POST /v3/conversations/{conversationId}/activities
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
<b>Текст запроса</b>	Объект <a href="#">Activity</a>
<b>Возвращает</b>	Объект <a href="#">ResourceResponse</a>

### Обновить действие

Некоторые каналы позволяют изменить существующее действие, чтобы отразить новое состояние общения бота. Например, можно удалять кнопки из сообщения в общении после того, как пользователь нажмет одну из кнопок. При успешном выполнении эта операция обновляет указанное действие в рамках указанного общения.

```
PUT /v3/conversations/{conversationId}/activities/{activityId}
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
<b>Текст запроса</b>	Объект <a href="#">Activity</a>
<b>Возвращает</b>	Объект <a href="#">ResourceResponse</a>

### Отправить вложение на канал

Передает вложение указанного общения непосредственно в хранилище BLOB-объектов канала. Это позволяет хранить данные в соответствующем хранилище.

```
POST /v3/conversations/{conversationId}/attachments
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
<b>Текст запроса</b>	Объект <a href="#">AttachmentData</a> .
<b>Возвращает</b>	Объект <a href="#">ResourceResponse</a> . Свойство <code>id</code> определяет идентификатор вложения, который может использоваться с операциями <a href="#">получение сведений о вложениях</a> и <a href="#">получение вложение</a> .

## Операции вложения

Используйте следующие операции для получения сведений о вложениях и двоичных данных для самого файла.

ОПЕРАЦИЯ	ОПИСАНИЕ
<a href="#">Получить сведения о вложении</a>	Получает сведения об указанном вложении, включая имя файла, тип файла и список доступных представлений (например, исходные представления или эскизы).
<a href="#">Получить вложение</a>	Получает указанное представление указанного вложения как двоичное содержимое.

### Получить сведения о вложении

Получает сведения об указанном вложении, включая имя файла, тип и список доступных представлений (например, исходные представления или эскизы).

```
GET /v3/attachments/{attachmentId}
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
<b>Текст запроса</b>	Недоступно
<b>Возвращает</b>	Объект <a href="#">AttachmentInfo</a>

### Получить вложение

Получает указанное представление указанного вложения как двоичное содержимое.

```
GET /v3/attachments/{attachmentId}/views/{viewId}
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
<b>Текст запроса</b>	Недоступно
<b>Возвращает</b>	Двоичное содержимое, отображающее указанное представление указанного вложения

## Операции состояния (нерекомендуемые)

Служба состояний Microsoft Bot Framework устарела с 30 марта 2018 г. Ранее боты, созданные на основе службы Azure Bot или пакета SDK для Bot Builder, по умолчанию использовали подключение к этой службе, размещенной корпорацией Майкрософт, для хранения данных о состоянии бота. Теперь эти боты нужно обновить, чтобы они использовали собственное хранилище состояний.

ОПЕРАЦИЯ	ОПИСАНИЕ
<code>Set User Data</code>	Сохраняет данные состояния для конкретного пользователя на канале.
<code>Set Conversation Data</code>	Сохраняет данные состояния для конкретного общения на канале.
<code>Set Private Conversation Data</code>	Сохраняет данные состояния для конкретного пользователя в контексте указанного общения на канале.
<code>Get User Data</code>	Извлекает данные состояния, сохраненные ранее для конкретного пользователя во всех сессиях общения на канале.
<code>Get Conversation Data</code>	Извлекает данные состояния, сохраненные ранее для конкретного общения на канале.
<code>Get Private Conversation Data</code>	Извлекает данные состояния, сохраненные ранее для конкретного пользователя в контексте указанного общения на канале.

ОПЕРАЦИЯ	ОПИСАНИЕ
Delete State For User	Удаляет данные состояния, сохраненные ранее для конкретного пользователя.

## схема

Схема действия Bot Framework определяет объекты и свойства, которые бот может использовать для взаимодействия с пользователем.

ОБЪЕКТ	ОПИСАНИЕ
Объект Activity	Определяет сообщения, которыми обмениваются бот и пользователь.
Объект AnimationCard	Определяет карту, которая может воспроизводить файлы GIF с анимацией или короткие видео.
Объект Attachment	Определяет дополнительные сведения для включения в сообщение. Вложение может быть файлом мультимедиа (например, аудио, видео, изображением, файлом) или форматированной карточкой.
Объект AttachmentData	Описывает данные вложения.
Объект AttachmentInfo	Описывает вложение.
Объект AttachmentView	Определяет представление вложения.
Объект AudioCard	Определяет карту, которая может воспроизводить звуковые файлы.
Объект CardAction	Определяет действие для выполнения.
Объект CardImage	Определяет изображение для отображения на карте.
Объект ChannelAccount	Определяет бота или учетную запись пользователя на канале.
Объект ConversationAccount	Определяет общение на канале.
Объект ConversationMembers	Определяет участников беседы.
Объект ConversationParameters	Определяет параметры для создания нового общения.
Объект ConversationReference	Определяет конкретный момент в общении.
Объект ConversationResourceResponse	Определяет ответ на операцию <a href="#">Создать диалог</a> .
Объект ConversationsResult	Определяет результат вызова операции <a href="#">Получить диалоги</a> .
Объект Entity	Определяет объект сущности.

ОБЪЕКТ	ОПИСАНИЕ
Объект Error	Определяет ошибку.
Объект ErrorResponse	Определяет ответ API HTTP.
Объект Fact	Определяет пару "ключ — значение", содержащую факт.
Объект GeoCoordinates	Определяет географическое расположение с помощью координат всемирной геодезической системы (WSG84).
Объект HeroCard	Определяет карту с большим изображением, заголовком, текстом и командной кнопкой.
Объект InnerHttpError	Объект, представляющий внутреннюю ошибку HTTP.
Объект MediaEventValue	Дополнительный параметр для событий мультимедиа.
Объект MediaUrl	Определяет URL-адрес источника файла мультимедиа.
Объект Mention	Определяет пользователя или бота, упомянутого в общении.
Объект MessageReaction	Определяет ответную реакцию на сообщение.
Объект PagedMembersResult	Страница участников, возвращаемая операцией получения участников диалога по страницам.
Объект Place	Определяет место, упомянутое в общении.
Объект ReceiptCard	Определяет карту, которая содержит квитанцию о покупке.
Объект ReceiptItem	Определяет элемент строки в квитанции.
Объект ResourceResponse	Определяет ресурс.
Объект SemanticAction	Определяет ссылку на программное действие.
Объект SignInCard	Определяет карту, которая позволяет пользователю выполнить вход в службу.
Объект SuggestedActions	Определяет варианты действий, из которых пользователь может выбирать.
Объект TextHighlight	Ссылается на подстроку содержимого в другом поле.
Объект ThumbnailCard	Определяет карту с эскизом изображения, заголовком, текстом и командной кнопкой.
Объект ThumbnailUrl	Определяет URL-адрес источника изображения.

ОБЪЕКТ	ОПИСАНИЕ
Объект Transcript	Коллекция действий, отправляемых с помощью операции <a href="#">Отправить журнал бесед</a> .
Объект VideoCard	Определяет карту, которая может воспроизводить видео.

## Объект Activity

Определяет сообщения, которыми обмениваются бот и пользователь.

СВОЙСТВО	ТИП	ОПИСАНИЕ
action	строка	Действие для применения или уже примененное. Используйте свойство type, чтобы определить контекст для действия. Например, если свойству type задано contactRelationUpdate, значение свойства action будет равняться add, если пользователь добавил бота в список контактов, или remove, если пользователь удалил бота из списка контактов.
attachmentLayout	строка	Макет <b>вложенных</b> форматированных карточек, включенных в сообщение. Одно из следующих значений: carousel, list. Дополнительные сведения о форматированных карточках см. в статье <a href="#">Добавление форматированных карточек как вложения к сообщениям</a> .
attachments	Attachment[]	Массив объектов Attachment, который определяет дополнительные сведения для включения в сообщение. Каждое вложение может быть файлом (например, аудио, видео, изображением) или форматированной карточкой.
callerId	строка	Строка, содержащая IRI для идентификации объекта, который вызывает бота. Это поле не предназначено для передачи по сети. Оно заполняется ботами и клиентами на основе доступных для криптографической проверки данных, которые подтверждают подлинность вызывающих объектов (например, маркеров).

СВОЙСТВО	ТИП	ОПИСАНИЕ
channelData	объект	Объект, содержащий определяемое каналом содержимое. Некоторые каналы предоставляют функции, требующие дополнительных сведений, которые не могут быть представлены с помощью схемы вложения. В таких случаях для этого свойства задается определяемое каналом содержимое, как указано в документации канала. Дополнительные сведения см. в статье <a href="#">Реализация функциональных возможностей канала</a> .
channelId	строка	Идентификатор, однозначно определяющий канал. Задается каналом.
code	строка	Код, указывающий причину завершения общения.
<b>беседа</b>	<a href="#">ConversationAccount</a>	Объект <a href="#">ConversationAccount</a> , определяющий общение, к которому относится действие.
deliveryMode	строка	Указание, определяющее альтернативные пути доставки действия получателю. Одно из следующих значений: <b>normal</b> , <b>notification</b> .
entities	object[]	Массив объектов, представляющих сущности, которые были упомянуты в сообщении. Объекты в этом массиве могут быть любыми объектами из <a href="#">Schema.org</a> . Например, массив может содержать объекты <a href="#">Mention</a> , определяющие пользователя, который был упомянут в общении, и объекты <a href="#">Place</a> , определяющие место, которое было упомянуто в общении.
expiration	строка	Время, в течение которого действие должно считаться истекшим и не должно быть представлено получателю.
from	<a href="#">ChannelAccount</a>	Объект <a href="#">ChannelAccount</a> , указывающий отправителя сообщения.
historyDisclosed	Логическое	Флаг, указывающий, раскрывается ли журнал. Значение по умолчанию — <b>false</b> .

Свойство	Тип	Описание
идентификатор	строка	Идентификатор, однозначно идентифицирующий действие на канале.
importance	строка	Определяет важность действия. Одно из следующих значений: <code>low</code> , <code>normal</code> , <code>high</code> .
inputHint	строка	Значение, указывающее, принимает, ожидает или игнорирует бот входные данные пользователя после доставки сообщения клиенту. Одно из следующих значений: <code>acceptingInput</code> , <code>expectingInput</code> или <code>ignoringInput</code> .
label	строка	Описательная метка для действия.
listenFor	string[]	Список фраз и ссылок, которые должны прослушиваться в системах подготовки речи и языка.
locale	строка	Языковой стандарт для языка, который должен использоваться для отображения текста в сообщении, в формате <code>&lt;language&gt;-&lt;country&gt;</code> . Канал использует это свойство для указания языка пользователя, чтобы бот мог указывать отображаемые строки на этом языке. Значение по умолчанию — <code>en-US</code> .
localTimestamp	строка	Дата и время отправки сообщения по местному часовому поясу, выраженные в формате <a href="#">ISO-8601</a> .
localTimezone	строка	Содержит имя локального часового пояса сообщения, представленного в формате базы данных часового пояса IANA. Например, <code>America/Los_Angeles</code> .
membersAdded	ChannelAccount[]	Массив объектов <code>ChannelAccount</code> , представляющий список пользователей, которые присоединились к общению. Присутствует, только когда <b>тип</b> действия равен <code>conversationUpdate</code> и пользователи присоединились к общению.

СВОЙСТВО	ТИП	ОПИСАНИЕ
membersRemoved	ChannelAccount[]	Массив объектов ChannelAccount, представляющий список пользователей, которые вышли из общения. Присутствует, только когда <b>тип</b> действия — conversationUpdate и пользователи вышли из общения.
name	строка	Имя операции для вызова или имя события.
reactionsAdded	MessageReaction[]	Коллекция реакций, добавленных в диалог.
reactionsRemoved	MessageReaction[]	Коллекция реакций, удаленных из диалога.
recipient	ChannelAccount	Объект ChannelAccount, указывающий получателя сообщения.
relatesTo	ConversationReference	Объект ConversationReference, который определяет конкретный момент в общении.
replyToId	строка	Идентификатор сообщения, ответом на которое является это сообщение. Чтобы ответить на сообщение, отправленное пользователем, присвойте этому свойству значение идентификатора сообщения пользователя. Не все каналы поддерживают цепочки ответов. В таких случаях канал игнорирует это свойство и использует упорядоченную по времени семантику (метку времени) для добавления сообщения в диалог.
semanticAction	SemanticAction	Объект SemanticAction, представляющий ссылку на программное действие.
serviceUrl	строка	URL-адрес, указывающий конечную точку службы канала. Задается каналом.
speak	строка	Текст, который боту требуется произнести на канале с поддержкой речевых функций. Чтобы контролировать различные характеристики речи бота, такие как голос, скорость, громкость, произношение и тон, укажите это свойство в формате <a href="#">языка разметки синтеза речи (SSML)</a> .

СВОЙСТВО	ТИП	ОПИСАНИЕ
suggestedActions	SuggestedActions	Объект SuggestedActions, определяющий варианты действий, из которых пользователь может выбирать.
summary	строка	Сводка данных, которая содержит сообщение. Например, для сообщения, которое отправляется на канал электронной почты, это свойство может указывать первые 50 символов электронного сообщения.
text	строка	Текст сообщения, отправляемого от пользователя к боту или от бота к пользователю. См. документацию канала по ограничениям, накладываемым на содержимое этого свойства.
textFormat	строка	Формат <b>текста</b> сообщения. Одно из следующих значений: <code>markdown</code> , <code>plain</code> или <code>xml</code> . Дополнительные сведения о формате текста см. в статье <a href="#">Создание сообщений</a> .
textHighlights	TextHighlight[]	Коллекция фрагментов текста для выделения, если действие содержит значение <code>replyToId</code> .
timestamp	строка	Дата и время отправки сообщения в часовом поясе UTC, выраженные в формате <a href="#">ISO-8601</a> .
topicName	строка	Раздел общения, к которому относится действие.
type	строка	Тип действия. Одно из следующих значений: <code>message</code> , <code>contactRelationUpdate</code> , <code>conversationUpdate</code> , <code>typing</code> , <code>endOfConversation</code> , <code>event</code> , <code>invoke</code> , <code>deleteUserData</code> , <code>messageUpdate</code> , <code>messageDelete</code> , <code>installationUpdate</code> , <code>messageReaction</code> , <code>suggestion</code> , <code>trace</code> , <code>handoff</code> . Дополнительные сведения о типах действий см. в статье <a href="#">Общие сведения о действиях</a> .
value	объект	Открытое значение.
valueType	строка	Тип объекта значения действия.

[Вернуться к таблице "Схема"](#)

## Объект AnimationCard

Определяет карту, которая может воспроизводить файлы GIF с анимацией или короткие видео.

СВОЙСТВО	ТИП	ОПИСАНИЕ
aspect	Логическое	Пропорции эскиза или заполнителя мультимедиа. Допустимые значения: 16:9 и 4:3.
autoloop	Логическое	Флаг, указывающий, следует ли повторять воспроизведение списка анимированных изображений в формате GIF при завершении последнего. Для автоматического повтора воспроизведения анимации присвойте этому свойству значение <code>true</code> , в противном случае — <code>false</code> . Значение по умолчанию — <code>true</code>
autostart	Логическое	Флаг, указывающий, следует ли автоматически воспроизводить анимацию при отображении карты. Для автоматического воспроизведения анимации присвойте этому свойству значение <code>true</code> , в противном случае — <code>false</code> . Значение по умолчанию — <code>true</code>
buttons	<a href="#">CardAction[]</a>	Массив объектов <code>CardAction</code> , позволяющий пользователю выполнять одно или несколько действий. Канал определяет количество кнопок, которые можно указать.
duration	строка	Длина мультимедийных данных в <a href="#">формате длительности ISO 8601</a> .
image	<a href="#">ThumbnailUrl</a>	Объект <code>ThumbnailUrl</code> , указывающий изображение, отображаемое на карте.
media	<a href="#">MediaUrl[]</a>	Массив объектов <code>MediaUrl</code> . Если это поле содержит более одного URL-адреса, это значит, что каждый URL-адрес представляет альтернативный формат одного содержимого.
shareable	Логическое	Флаг, указывающий, может ли анимации быть предоставлен общий доступ. Если анимация может находиться в общем доступе, присвойте этому свойству значение <code>true</code> , в противном случае — <code>false</code> . Значение по умолчанию — <code>true</code>

СВОЙСТВО	ТИП	ОПИСАНИЕ
subtitle	строка	Подзаголовок для отображения под заголовком карты.
text	строка	Описание или запрос для отображения под заголовком или подзаголовком карты.
title	строка	Заголовок карты.
value	объект	Дополнительный параметр для этой карты.

[Вернуться к таблице "Схема"](#)

### Объект Attachment

Определяет дополнительные сведения для включения в сообщение. Вложение может быть файлом (например, изображением, аудио или видео) или форматированной карточкой.

СВОЙСТВО	ТИП	ОПИСАНИЕ
content	объект	Содержимое вложения. Если вложение — это форматированная карточка, укажите для этого свойства объект форматированной карты. Это свойство и свойство <code>contentUrl</code> являются взаимоисключающими.

СВОЙСТВО	ТИП	ОПИСАНИЕ
contentType	строка	<p>Тип мультимедиа содержимого во вложении. Для файлов мультимедиа задайте этому свойству значение известных типов мультимедиа, например <code>image/png</code>, <code>audio/wav</code> или <code>video/mp4</code>. Для форматированных карточек задайте этому свойству один из следующих типов, определяемых поставщиком:</p> <ul style="list-style-type: none"> <li>• <code>application/vnd.microsoft.card.adaptive</code>: функциональная карточка, которая может содержать произвольное сочетание текста, речи, изображений, кнопок и полей для ввода. Для свойства <code>content</code> укажите объект <a href="#">AdaptiveCard</a>.</li> <li>• <code>application/vnd.microsoft.card.animation</code>: функциональная карточка для воспроизведения анимации. Для свойства <code>content</code> укажите объект <a href="#">AnimationCard</a>.</li> <li>• <code>application/vnd.microsoft.card.audio</code>: функциональная карточка для воспроизведения аудио-файлов. Для свойства <code>content</code> укажите объект <a href="#">AudioCard</a>.</li> <li>• <code>application/vnd.microsoft.card.hero</code>: карточка имиджевого баннера. Для свойства <code>content</code> укажите объект <a href="#">HeroCard</a>.</li> <li>• <code>application/vnd.microsoft.card.receipt</code>: карточка квитанции. Для свойства <code>content</code> укажите объект <a href="#">ReceiptCard</a>.</li> <li>• <code>application/vnd.microsoft.card.signin</code>: карточка входа пользователя. Для свойства <code>content</code> укажите объект <a href="#">SignInCard</a>.</li> <li>• <code>application/vnd.microsoft.card.thumbnail</code>: карточка эскиза. Для свойства <code>content</code> укажите объект <a href="#">ThumbnailCard</a>.</li> <li>• <code>application/vnd.microsoft.card.video</code>: функциональная карточка для воспроизведения видео. Для свойства <code>content</code> укажите объект <a href="#">VideoCard</a>.</li> </ul>

свойство	тип	описание
contentUrl	строка	URL-адрес для содержимого вложения. Например, если вложение — это изображение, укажите для свойства <code>contentUrl</code> URL-адрес, представляющий расположение этого изображения. Поддерживаемые протоколы: HTTP, HTTPS, а также протоколы передачи файлов и данных.
name	строка	Имя вложения.
thumbnailUrl	строка	URL-адрес эскиза, который канал может использовать, если поддерживает использование альтернативных, меньших форм свойств <code>content</code> или <code>contentUrl</code> . Например, если задать свойству <code>contentType</code> значение <code>application/word</code> , а свойству <code>contentUrl</code> — расположение документа Word, можно включить эскиз изображения, представляющего документ. Канал будет отображать эскиз изображения вместо документа. При щелчке изображения канал откроет документ.

[Вернуться к таблице "Схема"](#)

### Объект AttachmentData

Описывает данные вложения.

свойство	тип	описание
name	строка	Имя вложения.
originalBase64	строка	Содержимое вложения.
thumbnailBase64	строка	Содержимое эскиза вложения.
type	строка	Тип содержимого вложения.

[Вернуться к таблице "Схема"](#)

### Объект AttachmentInfo

Метаданные для вложения.

свойство	тип	описание
name	строка	Имя вложения.
type	строка	Тип содержимого вложения.

СВОЙСТВО	ТИП	ОПИСАНИЕ
представления;	AttachmentView[]	Массив объектов AttachmentView, представляющий доступные представления для вложения.

[Вернуться к таблице "Схема"](#)

### Объект AttachmentView

Определяет представление вложения.

СВОЙСТВО	ТИП	ОПИСАНИЕ
size	number	Размер файла.
viewId	строка	Идентификатор представления.

[Вернуться к таблице "Схема"](#)

### Объект AudioCard

Определяет карту, которая может воспроизводить звуковые файлы.

СВОЙСТВО	ТИП	ОПИСАНИЕ
aspect	строка	Пропорции эскиза, указанные в свойстве image. Допустимые значения: 16:9 и 4:3.
autoloop	Логическое	Флаг, указывающий, следует ли повторять воспроизведение списка аудиофайлов при завершении последнего. Для автоматического повторного воспроизведения аудиофайлов присвойте этому свойству значение true, в противном случае — false. Значение по умолчанию — true
autostart	Логическое	Флаг, указывающий, следует ли автоматически воспроизводить звук при отображении карты. Для автоматического воспроизведения звука присвойте этому свойству значение true, в противном случае — false. Значение по умолчанию — true
buttons	CardAction[]	Массив объектов CardAction, позволяющий пользователю выполнять одно или несколько действий. Канал определяет количество кнопок, которые можно указать.
duration	строка	Длина мультимедийных данных в <a href="#">формате длительности ISO 8601</a> .

СВОЙСТВО	ТИП	ОПИСАНИЕ
image	ThumbnailUrl	Объект ThumbnailUrl, указывающий изображение, отображаемое на карте.
media	MediaUrl[]	Массив объектов MediaUrl. Если это поле содержит более одного URL-адреса, это значит, что каждый URL-адрес представляет альтернативный формат одного содержимого.
shareable	Логическое	Флаг, указывающий, может ли быть предоставлен общий доступ к аудиофайлам. Если аудиофайлы могут находиться в общем доступе, присвойте этому свойству значение <code>true</code> , в противном случае — <code>false</code> . Значение по умолчанию — <code>true</code>
subtitle	строка	Подзаголовок для отображения под заголовком карты.
text	строка	Описание или запрос для отображения под заголовком или подзаголовком карты.
title	строка	Заголовок карты.
value	объект	Дополнительный параметр для этой карты.

[Вернуться к таблице "Схема"](#)

### Объект CardAction

Определяет доступное для щелчка действие с кнопкой.

СВОЙСТВО	ТИП	ОПИСАНИЕ
channelData	строка	Относящиеся к каналу данные, связанные с этим действием.
displayText	строка	Текст, отображаемый в канале чата при нажатии кнопки.
image	строка	URL-адрес изображения, которое будет отображаться на кнопке рядом с текстовой меткой.
text	строка	Текст для действия.
title	строка	Текстовое описание, которое отображается на кнопке.

СВОЙСТВО	ТИП	ОПИСАНИЕ
type	строка	Тип действия для выполнения. Список допустимых значений см. в статье <a href="#">Добавление форматированных карточек как вложения к сообщениям</a> .
value	объект	Дополнительный параметр для этого действия. Поведение этого свойства будет меняться в зависимости от <b>типа</b> действия. Дополнительные сведения см. в статье <a href="#">Добавление форматированных карточек как вложения к сообщениям</a> .

[Вернуться к таблице "Схема"](#)

### Объект CardImage

Определяет изображение для отображения на карте.

СВОЙСТВО	ТИП	ОПИСАНИЕ
alt	строка	Описание изображения. Следует включить описание для поддержки специальных возможностей.
tap	CardAction	Объект CardAction, указывающий действие, выполняемое при нажатии или щелчке изображения.
url	строка	URL-адрес источника изображения или двоичный код base64 изображения (например, <code>data:image/png;base64,iVBORw0KGgo...</code> ).

[Вернуться к таблице "Схема"](#)

### Объект ChannelAccount

Определяет бота или учетную запись пользователя на канале.

СВОЙСТВО	ТИП	ОПИСАНИЕ
aadObjectId	строка	Идентификатор объекта этой учетной записи в Azure Active Directory.
идентификатор	строка	Уникальный идентификатор пользователя или бота в этом канале.
name	строка	Понятное имя бота или пользователя.

Свойство	Тип	Описание
role	строка	Роль сущности, которая находится за учетной записью. <code>user</code> или <code>bot</code> .

[Вернуться к таблице "Схема"](#)

### Объект ConversationAccount

Определяет общение на канале.

Свойство	Тип	Описание
aadObjectId	строка	Идентификатор объекта этой учетной записи в Azure Active Directory (AAD).
conversationType	строка	Указывает тип диалога в каналах, которые могут определять разные типы (например, групповое или личное общение).
<b>идентификатор</b>	строка	Идентификатор, определяющий общение. Идентификатор является уникальным для каждого канала. Если в канале начинается диалог, канал определяет этот идентификатор. В противном случае бот присваивает этому свойству идентификатор, который получает в ответе при запуске диалога (см. инструкции по <a href="#">началу диалога</a> ).
isGroup	Логическое	Флаг, указывающий, содержит ли общение более двух участников во время создания действия. Задайте значение <code>true</code> , если это групповой общий, в противном случае — <code>false</code> . Значение по умолчанию — <code>false</code> .
name	строка	Отображаемое имя, которое может использоваться для идентификации общения.
role	строка	Роль сущности, которая находится за учетной записью. <code>user</code> или <code>bot</code> .
tenantId	строка	Идентификатор клиента диалога.

[Вернуться к таблице "Схема"](#)

### Объект ConversationMembers

Определяет участников беседы.

СВОЙСТВО	ТИП	ОПИСАНИЕ
идентификатор	строка	Идентификатор разговора.
members	ChannelAccount[]	Список участников в этом диалоге.

[Вернуться к таблице "Схема"](#)

### Объект ConversationParameters

Определяет параметры для создания диалога.

СВОЙСТВО	ТИП	ОПИСАНИЕ
activity	Действие	Начальное сообщение, отправляемое в создаваемый диалог.
bot	ChannelAccount	Сведения об учетной записи канала, требуемые для отправки сообщения в бот.
channelData	объект	Связанные с каналом полезные данные для создания диалога.
isGroup	Логическое	Указывает, является ли диалог групповым.
members	ChannelAccount[]	Сведения об учетной записи канала, требуемые для отправки сообщения каждому пользователю.
tenantId	строка	Идентификатор клиента, в котором создается диалог.
topicName	строка	Тема диалога. Это свойство используется только в том случае, если канал его поддерживает.

[Вернуться к таблице "Схема"](#)

### Объект ConversationReference

Определяет конкретный момент в общении.

СВОЙСТВО	ТИП	ОПИСАНИЕ
activityId	строка	Идентификатор, однозначно определяющий действие, на которое ссылается данный объект.
bot	ChannelAccount	Объект ChannelAccount, идентифицирующий бота в общении, на который ссылается данный объект.

СВОЙСТВО	ТИП	ОПИСАНИЕ
channelId	строка	Идентификатор, однозначно определяющий канал в общении, на которое ссылается данный объект.
<b>беседа</b>	ConversationAccount	Объект ConversationAccount, определяющий общение, на которое ссылается данный объект.
serviceUrl	строка	URL-адрес, указывающий конечную точку службы канала в общении, на которое ссылается данный объект.
user	ChannelAccount	Объект ChannelAccount, идентифицирующий пользователя в общении, на которое ссылается данный объект.

[Вернуться к таблице "Схема"](#)

### Объект ConversationResourceResponse

Определяет ответ на операцию [Создать диалог](#).

СВОЙСТВО	ТИП	ОПИСАНИЕ
activityId	строка	Идентификатор действия (при его отправке).
<b>идентификатор</b>	строка	Идентификатор ресурса.
serviceUrl	строка	Конечная точка службы, в которой могут выполняться операции, связанные с диалогом.

[Вернуться к таблице "Схема"](#)

### Объект ConversationsResult

Определяет результат операции [Получить диалоги](#).

СВОЙСТВО	ТИП	ОПИСАНИЕ
<b>Беседы</b>	ConversationMembers[]	Участники каждого диалога.
continuationToken	строка	Токен продолжения, который может использоваться в дальнейших вызовах операции <a href="#">Получить диалоги</a> .

[Вернуться к таблице "Схема"](#)

### Объект Entity

Объект метаданных, относящийся к действию.

СВОЙСТВО	ТИП	ОПИСАНИЕ
type	строка	Тип этой сущности (RFC 3987 IRI).

[Вернуться к таблице "Схема"](#)

#### Объект ошибки

Объект, представляющий сведения об ошибке.

СВОЙСТВО	ТИП	ОПИСАНИЕ
code	строка	Код ошибки.
innerHttpError	InnerHttpError	Объект, представляющий внутреннюю ошибку HTTP.
message	строка	Текстовое описание ошибки.

[Вернуться к таблице "Схема"](#)

#### Объект ErrorResponse

Определяет ответ API HTTP.

СВОЙСТВО	ТИП	ОПИСАНИЕ
error	Error	Объект Error, содержащий сведения об ошибке.

[Вернуться к таблице "Схема"](#)

#### Объект Fact

Определяет пару "ключ — значение", содержащую факт.

СВОЙСТВО	ТИП	ОПИСАНИЕ
key	строка	Имя факта. Например, Check-in. Ключ используется как метка при отображении значения факта.
value	строка	Значение факта. Например, 10 октября 2016 г.

[Вернуться к таблице "Схема"](#)

#### Объект GeoCoordinates

Определяет географическое расположение с помощью координат всемирной геодезической системы (WSG84).

СВОЙСТВО	ТИП	ОПИСАНИЕ
elevation	number	Высота расположения.
latitude	number	Широта расположения.

СВОЙСТВО	ТИП	ОПИСАНИЕ
longitude	number	Долгота расположения.
name	строка	Название расположения.
type	строка	Тип этого объекта. Всегда имеет значение <code>GeoCoordinates</code> .

[Вернуться к таблице "Схема"](#)

### Объект HeroCard

Определяет карту с большим изображением, заголовком, текстом и командной кнопкой.

СВОЙСТВО	ТИП	ОПИСАНИЕ
buttons	<code>CardAction[]</code>	Массив объектов <code>CardAction</code> , позволяющий пользователю выполнять одно или несколько действий. Канал определяет количество кнопок, которые можно указать.
images	<code>CardImage[]</code>	Массив объектов <code>CardImage</code> , указывающий изображение для отображения на карте. Карта для имиджевого баннера содержит только одно изображение.
subtitle	строка	Подзаголовок для отображения под заголовком карты.
tap	<code>CardAction</code>	Объект <code>CardAction</code> , указывающий действие, выполняемое при нажатии или щелчке карты. Это может быть то же действие, которое присвоено одной из кнопок, или другое действие.
text	строка	Описание или запрос для отображения под заголовком или подзаголовком карты.
title	строка	Заголовок карты.

[Вернуться к таблице "Схема"](#)

### Объект InnerHttpError

Объект, представляющий внутреннюю ошибку HTTP.

СВОЙСТВО	ТИП	ОПИСАНИЕ
statusCode	number	Код состояния HTTP из невыполненного запроса.
body	объект	Текст из невыполненного запроса.

[Вернуться к таблице "Схема"](#)

### Объект MediaEventValue

Дополнительный параметр для событий мультимедиа.

свойство	тип	описание
cardValue	объект	Параметр обратного вызова, указанный в поле <code>value</code> карты мультимедиа, являющийся источником этого события.

[Вернуться к таблице "Схема"](#)

### Объект MediaUrl

Определяет URL-адрес источника файла мультимедиа.

свойство	тип	описание
profile	строка	Указание, описывающее содержимое мультимедиа.
url	строка	URL-адрес источника файла мультимедиа.

[Вернуться к таблице "Схема"](#)

### Объект Mention

Определяет пользователя или бота, упомянутого в общении.

свойство	тип	описание
mentioned	ChannelAccount	Объект <code>ChannelAccount</code> , указывающий пользователя или бота, который ранее был упомянут. Обратите внимание, что некоторые каналы, такие как Slack, назначают новые имена для каждого общения, так что вполне возможно, что упомянутое имя бота (в свойстве сообщения <code>recipient</code> ) может отличаться от дескриптора, который был указан при <a href="#">регистрации</a> бота. Тем не менее идентификаторы учетных записей как для нового, так и для указанного имени бота будут совпадать.
text	строка	Пользователь или бот, упомянутый в общении. Например, если сообщение « @ColorBot выберите новый цвет», это свойство будет иметь значение <b>@ колорбот</b> . Не все каналы устанавливают это свойство.

Свойство	Тип	Описание
type	строка	Тип этого объекта. Всегда имеет значение Mention.

[Вернуться к таблице "Схема"](#)

### Объект MessageReaction

Определяет ответную реакцию на сообщение.

Свойство	Тип	Описание
type	строка	Тип реакции. like или plusOne.

[Вернуться к таблице "Схема"](#)

### Объект PagedMembersResult

Страница участников, возвращаемая операцией [получения участников диалога по страницам](#).

Свойство	Тип	Описание
continuationToken	строка	Маркер продолжения, который можно использовать в дальнейших вызовах операции <a href="#">получения участников диалога по страницам</a> .
members	ChannelAccount[]	Массив участников диалога.

[Вернуться к таблице "Схема"](#)

### Объект Place

Определяет место, упомянутое в общении.

Свойство	Тип	Описание
address	объект	Адрес места. Это свойство может иметь тип string или быть сложным объектом типа PostalAddress.
geo	GeoCoordinates	Объект GeoCoordinates, указывающий координаты географического расположения.
hasMap	объект	Карта места. Это свойство может иметь тип string (URL-адрес) или быть сложным объектом типа Map.
name	строка	Название места.
type	строка	Тип этого объекта. Всегда имеет значение Place.

[Вернуться к таблице "Схема"](#)

### Объект ReceiptCard

Определяет карту, которая содержит квитанцию о покупке.

СВОЙСТВО	ТИП	ОПИСАНИЕ
buttons	CardAction[]	Массив объектов CardAction, позволяющий пользователю выполнять одно или несколько действий. Канал определяет количество кнопок, которые можно указать.
facts	Fact[]	Массив объектов Fact, указывающий данные о покупке. Например, в списке фактов квитанции о пребывании в гостинице могут быть дата заселения и дата выселения. Канал определяет количество фактов, которые можно указать.
items	ReceiptItem[]	Массив объектов ReceiptItem, указывающий приобретенные единицы.
tap	CardAction	Объект CardAction, указывающий действие, выполняемое при нажатии или щелчке карты. Это может быть то же действие, которое присвоено одной из кнопок, или другое действие.
tax	строка	Строка в валютном формате, указывающая величину налога, накладываемого на покупку.
title	строка	Заголовок, отображаемый в верхней части уведомления.
total	строка	Строка в формате валюты, указывающая общую стоимость покупки, включая все соответствующие налоги.
vat	строка	Строка в формате валюты, которая определяет величину налога на добавленную стоимость (НДС) со стоимости покупки.

[Вернуться к таблице "Схема"](#)

### Объект ReceiptItem

Определяет элемент строки в квитанции.

СВОЙСТВО	ТИП	ОПИСАНИЕ
image	CardImage	Объект CardImage, указывающий эскиз изображения для отображения рядом с элементом строки.

СВОЙСТВО	ТИП	ОПИСАНИЕ
----------	-----	----------

price	строка	Строка в формате валюты, указывающая общую стоимость всех приобретенных единиц.
quantity	строка	Числовая строка, указывающая количество приобретенных единиц.
subtitle	строка	Подзаголовок, отображаемый под заголовком элемента строки.
tap	CardAction	Объект CardAction, указывающий действие, выполняемое при нажатии или щелчке элемента строки.
text	строка	Описание элемента строки.
title	строка	Название элемента строки.

[Вернуться к таблице "Схема"](#)

#### Объект ResourceResponse

Определяет ответ, содержащий идентификатор ресурса.

СВОЙСТВО	ТИП	ОПИСАНИЕ
идентификатор	строка	Идентификатор, уникально идентифицирующий ресурс.

[Вернуться к таблице "Схема"](#)

#### Объект SemanticAction

Определяет ссылку на программное действие.

СВОЙСТВО	ТИП	ОПИСАНИЕ
entities	объект	Объект, где значение каждого свойства является объектом Entity.
идентификатор	строка	Идентификатор этого действия.
state	строка	Состояние этого действия. Допустимые значения: start, continue, done.

[Вернуться к таблице "Схема"](#)

#### Объект SignInCard

Определяет карту, которая позволяет пользователю выполнить вход в службу.

СВОЙСТВО	ТИП	ОПИСАНИЕ
buttons	CardAction[]	Массив объектов CardAction, позволяющий пользователю войти в службу. Канал определяет количество кнопок, которые можно указать.
text	строка	Описание или запрос для включения на карте входа.

[Вернуться к таблице "Схема"](#)

### Объект SuggestedActions

Определяет варианты действий, из которых пользователь может выбирать.

СВОЙСТВО	ТИП	ОПИСАНИЕ
actions	CardAction[]	Массив объектов CardAction, определяющий предлагаемые действия.
to	string[]	Массив строк, содержащий идентификаторы получателей, которым должны отображаться предлагаемые действия.

[Вернуться к таблице "Схема"](#)

### Объект TextHighlight

Ссылается на подстроку содержимого в другом поле.

СВОЙСТВО	ТИП	ОПИСАНИЕ
occurrence	number	Вхождение текстового поля в тексте, на который указывает ссылка (при наличии повторов).
text	строка	Определяет фрагмент текста для выделения.

[Вернуться к таблице "Схема"](#)

### Объект ThumbnailCard

Определяет карту с эскизом изображения, заголовком, текстом и командной кнопкой.

СВОЙСТВО	ТИП	ОПИСАНИЕ
buttons	CardAction[]	Массив объектов CardAction, позволяющий пользователю выполнять одно или несколько действий. Канал определяет количество кнопок, которые можно указать.

СВОЙСТВО	ТИП	ОПИСАНИЕ
images	CardImage[]	Массив объектов CardImage, указывающий эскизы изображений для отображения на карте. Канал определяет количество эскизов изображений, которые можно указать.
subtitle	строка	Подзаголовок для отображения под заголовком карты.
tap	CardAction	Объект CardAction, указывающий действие, выполняемое при нажатии или щелчке карты. Это может быть то же действие, которое присвоено одной из кнопок, или другое действие.
text	строка	Описание или запрос для отображения под заголовком или подзаголовком карты.
title	строка	Заголовок карты.

[Вернуться к таблице "Схема"](#)

### Объект ThumbnailUrl

Определяет URL-адрес источника изображения.

СВОЙСТВО	ТИП	ОПИСАНИЕ
alt	строка	Описание изображения. Следует включить описание для поддержки специальных возможностей.
url	строка	URL-адрес источника изображения или двоичный код base64 изображения (например, <code>data:image/png;base64,iVBORw0KGgo...</code> ).

[Вернуться к таблице "Схема"](#)

### Объект Transcript

Коллекция действий, отправляемых с помощью операции [Отправить журнал бесед](#).

СВОЙСТВО	ТИП	ОПИСАНИЕ
действия	массиве	Массив объектов Activity. Необходимо, чтобы у них был уникальный идентификатор и метка времени.

[Вернуться к таблице "Схема"](#)

### Объект VideoCard

Определяет карту, которая может воспроизводить видео.

СВОЙСТВО	ТИП	ОПИСАНИЕ
aspect	строка	Пропорции видео. Например, 16:9 или 4:3.
autoloop	Логическое	Флаг, указывающий, следует ли повторять воспроизведение списка видео при завершении последнего. Для автоматического повторного воспроизведения видео присвойте этому свойству значение <b>true</b> , в противном случае — <b>false</b> . Значение по умолчанию — <b>true</b>
autoplay	Логическое	Флаг, указывающий, следует ли автоматически воспроизводить видео при отображении карты. Для автоматического воспроизведения видео присвойте этому свойству значение <b>true</b> , в противном случае — <b>false</b> . Значение по умолчанию — <b>true</b>
buttons	<a href="#">CardAction[]</a>	Массив объектов <a href="#">CardAction</a> , позволяющий пользователю выполнять одно или несколько действий. Канал определяет количество кнопок, которые можно указать.
duration	строка	Длина мультимедийных данных в <a href="#">формате длительности ISO 8601</a> .
image	<a href="#">ThumbnailUrl</a>	Объект <a href="#">ThumbnailUrl</a> , указывающий изображение, отображаемое на карте.
media	<a href="#">MediaUrl[]</a>	Массив <a href="#">MediaUrl</a> . Если это поле содержит более одного URL-адреса, это значит, что каждый URL-адрес представляет альтернативный формат одного содержимого.
shareable	Логическое	Флаг, указывающий, предоставлен ли видео общий доступ. Если видео могут находиться в общем доступе, присвойте этому свойству значение <b>true</b> , в противном случае — <b>false</b> . Значение по умолчанию — <b>true</b>
subtitle	строка	Подзаголовок для отображения под заголовком карты.
text	строка	Описание или запрос для отображения под заголовком или подзаголовком карты.

СВОЙСТВО	ТИП	ОПИСАНИЕ
title	строка	Заголовок карты.
value	объект	Дополнительный параметр для этой карты.

[Вернуться к таблице "Схема"](#)

# Проверка подлинности с помощью API соединителя Bot

27.03.2021 • 23 minutes to read • [Edit Online](#)

Бот взаимодействует со службой Bot Connector по протоколу HTTP через защищенный канал (SSL/TLS). Запрос, отправляемый ботом в службу Bot Connector, должен содержать сведения, с помощью которых служба будет проверять удостоверение бота. Точно так же, когда служба Bot Connector отправляет запрос боту, она должна включить в него сведения, с помощью которых бот проверит удостоверение службы. В этой статье описываются технологии проверки подлинности и требования к проверке подлинности уровня службы, выполняемой между ботом и службой Bot Connector. При написании собственного кода для проверки подлинности необходимо реализовать описанные в этой статье процедуры безопасности, чтобы бот мог обмениваться сообщениями со службой Bot Connector.

## IMPORTANT

Очень важно сделать это правильно. После выполнения всех действий в этой статье снижается вероятность прочтения злоумышленником сообщений, отправляемых боту, уменьшается риск отправки сообщений, олицетворяющих ботов, и кражи секретных ключей.

Если вы используете [пакет SDK Bot Framework для .NET](#) или [пакет SDK Bot Framework для Node.js](#), выполнять описанные в этой статье процедуры безопасности не требуется, так как эту задачу автоматически реализует пакет SDK. Просто настройте проект с использованием идентификатора приложения и пароля, полученных для бота во время [регистрации](#), а все остальное сделает пакет SDK.

## WARNING

В декабре 2016 г. в протоколе безопасности Bot Framework версии 3.1 были представлены изменения для нескольких значений, которые используются во время создания и проверки маркеров. В конце осени 2017 г. был представлен протокол безопасности Bot Framework версии 3.2 с внесенными изменениями для значений, которые используются во время создания и проверки маркеров. Дополнительные сведения см. в разделе об [изменениях в протоколе безопасности](#).

## Технологии проверки подлинности

Для установления доверительных отношений между ботом и службой Bot Connector используются следующие четыре технологии проверки подлинности.

ТЕХНОЛОГИЯ	ОПИСАНИЕ
SSL/TLS	SSL/TLS используется для всех подключений между службами. Сертификаты x.509v3 используются для идентификации всех служб HTTPS. <b>Клиенты всегда должны проверять сертификаты службы, чтобы убедиться в их надежности и допустимости.</b> (В рамках этой схемы сертификаты клиента не используются.)
OAuth 2.0	OAuth 2.0 создает маркер безопасности с использованием службы входа в учетную запись Azure Active Directory (Azure AD) версии 2 для. С помощью этого маркера которого бот может отправлять сообщения. Этот маркер работает между службами. Вход пользователя не требуется.

ТЕХНОЛОГИЯ	ОПИСАНИЕ
JSON Web Token (JWT)	JSON Web Token используется для кодирования маркеров, отправляемых ботам и из них. <b>Клиенты должны полностью проверять все получаемые маркеры JWT</b> в соответствии с требованиями, описанными в этой статье.
Метаданные OpenID	Служба Bot Connector публикует список допустимых маркеров, используемых для подписи собственных маркеров JWT в метаданных OpenID в хорошо известной статической конечной точке.

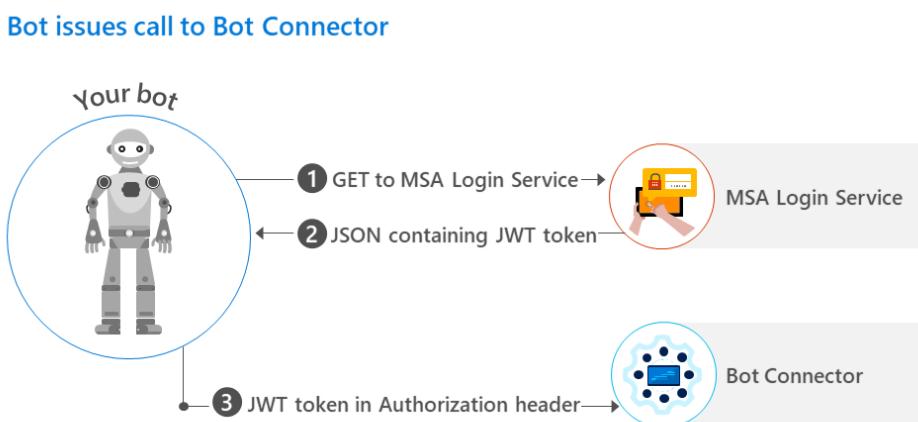
В этой статье описывается, как использовать эти технологии с помощью стандартных форматов HTTPS и JSON. Специальные пакеты SDK не требуются, однако могут быть полезны вспомогательные функции для OpenID и т. д.

## Проверка подлинности запросов от бота к службе Bot Connector

Для взаимодействия со службой Bot Connector необходимо указать маркер доступа в заголовке `Authorization` каждого запроса API, используя следующий формат:

```
Authorization: Bearer ACCESS_TOKEN
```

На этой схеме показаны действия для проверки подлинности бота в службе:



### Шаг 1. Запрос маркера доступа из службы входа в учетную запись Azure AD версии 2

#### IMPORTANT

Если это еще не сделано, [зарегистрируйте бот](#) в Bot Framework, чтобы получить идентификатор приложения и пароль. Идентификатор приложения и пароль бота потребуются для запроса маркера доступа.

Чтобы запросить маркер доступа из службы входа, выполните приведенный ниже запрос, заменив **MICROSOFT-APP-ID** и **MICROSOFT-APP-PASSWORD** идентификатором приложения и паролем бота, которые вы получили при [регистрации](#) бота в Bot Framework.

```
POST https://login.microsoftonline.com/botframework.com/oauth2/v2.0/token
Host: login.microsoftonline.com
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials&client_id=MICROSOFT-APP-ID&client_secret=MICROSOFT-APP-
PASSWORD&scope=https%3A%2F%2Fapi.botframework.com%2F.default
```

## Шаг 2. Получение маркера JWT из ответа службы входа в учетную запись Azure AD версии 2

Если приложение авторизовано службой входа, в тексте ответа JSON будет указан маркер доступа, его тип и срок действия (в секундах).

При добавлении маркера в заголовок `Authorization` запроса необходимо использовать точное значение, которое указано в этом ответе (т. е. не экранировать и не кодировать значение маркера). Маркер доступа действителен до окончания срока его действия. Чтобы предотвратить влияние окончания срока действия маркера на производительность бота, маркер можно кэшировать и заранее обновить.

В этом примере показан ответ службы входа в учетную запись Azure AD версии 2.

```
HTTP/1.1 200 OK
...
... (other headers)
```

```
{
  "token_type": "Bearer",
  "expires_in": 3600,
  "ext_expires_in": 3600,
  "access_token": "eyJhbGciOiJIUzI1Ni..."
}
```

## Шаг 3. Указание маркера JWT в заголовке авторизации запросов

При отправке запроса API в службу Bot Connector укажите маркер доступа в заголовке `Authorization` каждого запроса API, используя следующий формат:

```
Authorization: Bearer ACCESS_TOKEN
```

Все запросы, отправляемые в службу Bot Connector, должны содержать маркер доступа в заголовке `Authorization`. Если маркер имеет правильный формат, является действующим и был создан службой входа в учетную запись Azure AD версии 2, служба Bot Connector авторизует запрос. Чтобы убедиться, что маркер принадлежит боту, который отправил запрос, выполняются дополнительные проверки.

В следующем примере показано, как указать маркер доступа в заголовке `Authorization` запроса.

```
POST https://smbs.trafficmanager.net/apis/v3/conversations/12345/activities
Authorization: Bearer eyJhbGciOiJIUzI1Ni...
(JSON-serialized Activity message goes here)
```

### IMPORTANT

Маркер JWT следует задавать только в заголовке `Authorization` запросов, отправляемых в службу Bot Connector. Не отправляйте маркер через незащищенные каналы и не включайте его в HTTP-запросы, отправляемые в другие службы. Маркер JWT, полученный из службы входа в учетную запись Azure AD версии 2, аналогичен паролю, и при работе с ним нужно соблюдать меры безопасности. Пользователь, обладающий маркером, может использовать его для выполнения операций от имени вашего бота.

## Запрос от бота к службе Bot Connector: пример компонентов JWT

```

header:
{
  typ: "JWT",
  alg: "RS256",
  x5t: "<SIGNING KEY ID>",
  kid: "<SIGNING KEY ID>"
},
payload:
{
  aud: "https://api.botframework.com",
  iss: "https://sts.windows.net/d6d49420-f39b-4df7-a1dc-d59a935871db/",
  nbf: 1481049243,
  exp: 1481053143,
  appid: "<YOUR MICROSOFT APP ID>",
  ... other fields follow
}

```

#### NOTE

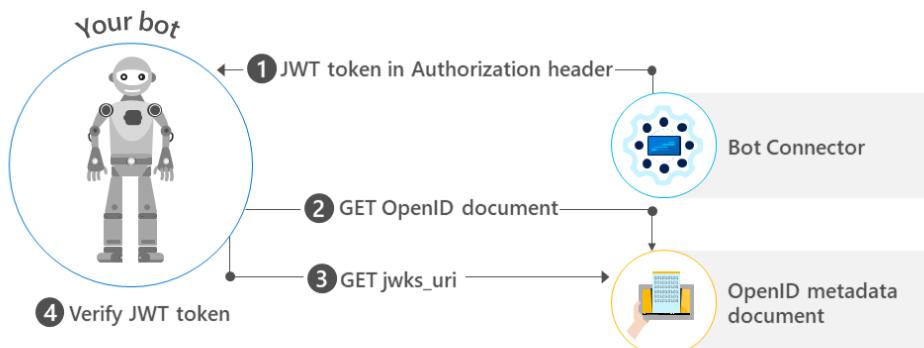
На практике могут использоваться другие поля. Создайте и проверьте все маркеры JWT, как указано выше.

## Проверка подлинности запросов от службы Bot Connector к боту

Когда служба Bot Connector отправляет запрос боту, она указывает подписанный маркер JWT в заголовке `Authorization` запроса. Бот может проверить подлинность вызовов от службы Bot Connector, проверив подлинность подписанного маркера JWT.

На этой схеме показаны действия для проверки подлинности службы в боте:

#### Bot Connector issues call to bot



#### Шаг 2. Получение документа метаданных OpenID

Документ метаданных OpenID указывает расположение второго документа, в котором приведены допустимые ключи подписи службы Bot Connector. Чтобы получить документ метаданных OpenID, выполните следующий запрос по протоколу HTTPS:

```
GET https://login.botframework.com/v1/.well-known/openidconfiguration
```

#### TIP

Это статический URL-адрес, который можно жестко задать в приложении.

В следующем примере показан документ метаданных OpenID, возвращаемый в ответ на запрос `GET`.

Свойство `jwks_uri` указывает расположение документа, который содержит допустимые ключи подписи службы Bot Connector.

```
{  
    "issuer": "https://api.botframework.com",  
    "authorization_endpoint": "https://invalid.botframework.com",  
    "jwks_uri": "https://login.botframework.com/v1/.well-known/keys",  
    "id_token_signing_alg_values_supported": [  
        "RS256"  
    ],  
    "token_endpoint_auth_methods_supported": [  
        "private_key_jwt"  
    ]  
}
```

### Шаг 3. Получение списка допустимых ключей подписи

Чтобы получить список допустимых ключей подписи, отправьте запрос `GET` по протоколу HTTPS на URL-адрес, указанный в свойстве `jwks_uri` в документе метаданных OpenID. Пример:

```
GET https://login.botframework.com/v1/.well-known/keys
```

В тексте ответа указан документ в [формате JWK](#) и также содержатся дополнительные свойства для каждого ключа: `endorsements`.

#### TIP

Список ключей является стабильным и может быть кэширован, но новые ключи могут быть добавлены в любое время. Чтобы убедиться, что программа-робот имеет актуальную копию документа перед использованием этих ключей, все экземпляры программы-робота должны **обновлять свой локальный кэш документа по крайней мере каждые 24 часа**.

Свойство `endorsements` в каждом ключе содержит одну или несколько строк подтверждения, которые можно использовать для аутентификации идентификатора канала, указанного в свойстве `channelId` в объекте [Действие](#) входящего запроса. Список идентификаторов каналов, требующих подтверждения, настраивается в каждом боте. По умолчанию в список будут входить все опубликованные идентификаторы каналов, однако разработчики ботов могут переопределить выбранные значения идентификаторов.

### Шаг 4. Проверка маркера JWT

Чтобы проверить подлинность маркера, отправленного службой Bot Connector, необходимо извлечь его из заголовка `Authorization` запроса, проанализировать, проверить его содержимое и подпись.

Библиотеки анализа JWT доступны для целого ряда платформ, и большинство из них предоставляет безопасные и надежные возможности анализа маркеров JWT, несмотря на то, что обычно эти библиотеки требуется настраивать для указания правильных значений в характеристиках маркера (например, издаватель, аудитория и проч.). При анализе маркера необходимо настроить библиотеку анализа или написать собственную процедуру проверки маркера, чтобы маркер соответствовал приведенным ниже требованиям.

1. Маркер был отправлен в заголовке HTTP `Authorization` со схемой носителя.
2. Маркер имеет допустимый формат JSON, соответствующий [стандарту JWT](#).
3. Маркер содержит утверждение "issuer" со значением <https://api.botframework.com>.
4. Маркер содержит утверждение "audience" со значением, соответствующим идентификатору приложения Майкрософт для бота.
5. Маркер является действующим. Отраслевая разница в показаниях часов составляет 5 минут.
6. Маркер имеет допустимую криптографическую подпись с ключом, приведенным в документе с ключами OpenID, который был получен на [шаге 3](#), и алгоритмом подписи, который указан в свойстве `id_token_signing_alg_values_supported` документа метаданных OpenID, который был получен на [шаге 2](#).
7. Маркер содержит утверждение `serviceUrl` со значением, которое соответствует свойству `serviceUrl` в корне объекта [Действие](#) входящего запроса.

Если требуется подтверждение для идентификатора канала:

- любой объект `Activity`, отправляемый в бот с этим идентификатором канала, должен сопровождаться маркером JWT, который подписан с использованием подтверждения для этого канала;
- если подтверждение отсутствует, бот должен отклонить запрос, возвратив код состояния HTTP 403 (**Запрещено**) .

#### IMPORTANT

Все эти требования являются важными, особенно требования 4 и 6. При невозможности выполнить все эти требования бот будет открыт для атак, что может привести к раскрытию его маркера JWT.

Разработчики должны следить за тем, чтобы проверка маркера JWT, отправляемого в бот, всегда была включена.

#### Запрос от службы Bot Connector к боту: пример компонентов JWT

```
header:  
{  
    typ: "JWT",  
    alg: "RS256",  
    x5t: "<SIGNING KEY ID>",  
    kid: "<SIGNING KEY ID>"  
},  
payload:  
{  
    aud: "<YOU MICROSOFT APP ID>",  
    iss: "https://api.botframework.com",  

```

#### NOTE

На практике могут использоваться другие поля. Создайте и проверьте все маркеры JWT, как указано выше.

## Проверка подлинности запросов от Bot Framework Emulator к боту

#### WARNING

В конце осени 2017 г. был представлен протокол безопасности Bot Framework версии 3.2. Новая версия содержит новое значение "issuer" в маркерах, которыми обмениваются Bot Framework Eumalator и бот. Чтобы подготовиться к этому изменению, ознакомьтесь с приведенными ниже действиями по проверке значений "issuer" в версиях 3.2 и 3.1.

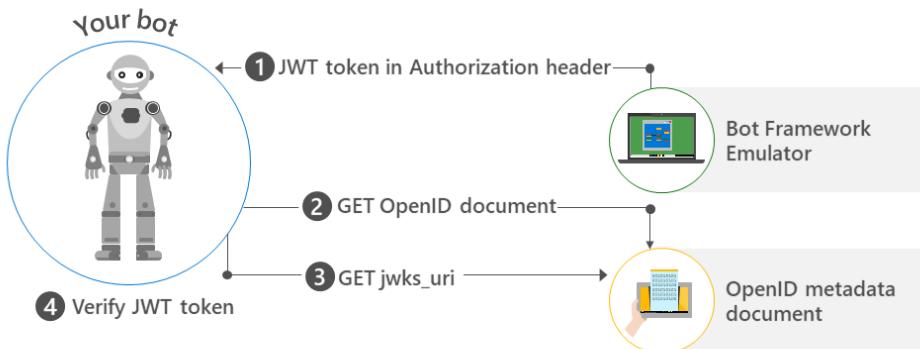
[Bot Framework Emulator](#) — это инструмент для настольных систем, используемый для тестирования работоспособности бота. Несмотря на то, что в Bot Framework Emulator применяются те же [технологии проверки подлинности](#), которые были описаны выше, это средство не может олицетворять реальную службу Bot Connector. Вместо этого он использует идентификатор приложения Майкрософт и пароль Microsoft App, которые указываются при подключении эмулятора к роботу для создания маркеров, идентичных тем, которые создает Bot. Когда эмулятор отправляет запрос в робот, он указывает маркер JWT в `Authorization` заголовке запроса, используя собственные учетные данные Bot для проверки подлинности запроса.

Если вы реализуете библиотеку проверки подлинности и хотите принимать запросы от Bot Framework Emulator, необходимо добавить этот путь дополнительной проверки. Путь является структурным аналогом пути проверки [соединителя > Bot](#), но вместо документа OpenID Connect соединителя Bot

используется документ MSA.

На этой схеме показаны шаги для проверки подлинности с помощью эмулятора в Bot:

### Bot Framework Emulator issues call to bot



#### Шаг 2. Получение документа метаданных OpenID MSA

Документ метаданных OpenID указывает расположение второго документа, в котором перечислены допустимые ключи подписи. Чтобы получить документ метаданных OpenID MSA, отправьте следующий запрос по протоколу HTTPS:

```
GET https://login.microsoftonline.com/botframework.com/v2.0/.well-known/openid-configuration
```

В следующем примере показан документ метаданных OpenID, возвращаемый в ответ на запрос `GET`.

Свойство `jwks_uri` указывает расположение документа, который содержит допустимые ключи подписи.

```
{  
    "authorization_endpoint": "https://login.microsoftonline.com/common/oauth2/v2.0/authorize",  
    "token_endpoint": "https://login.microsoftonline.com/common/oauth2/v2.0/token",  
    "token_endpoint_auth_methods_supported": ["client_secret_post", "private_key_jwt"],  
    "jwks_uri": "https://login.microsoftonline.com/common/discovery/v2.0/keys",  
    ...  
}
```

#### Шаг 3. Получение списка допустимых ключей подписи

Чтобы получить список допустимых ключей подписи, отправьте запрос `GET` по протоколу HTTPS на URL-адрес, указанный в свойстве `jwks_uri` в документе метаданных OpenID. Пример:

```
GET https://login.microsoftonline.com/common/discovery/v2.0/keys  
Host: login.microsoftonline.com
```

В тексте ответа документ указывается в [формате JWK](#).

#### Шаг 4. Проверка маркера JWT

Чтобы проверить подлинность токена, отправленного эмулятором, необходимо извлечь маркер из `Authorization` заголовка запроса, выполнить синтаксический анализ маркера, проверить его содержимое и проверить его подпись.

Библиотеки анализа JWT доступны для целого ряда платформ, и большинство из них предоставляет безопасные и надежные возможности анализа маркеров JWT, несмотря на то, что обычно эти библиотеки требуется настраивать для указания правильных значений в характеристиках маркера (например, издатель, аудитория и проч.). При анализе маркера необходимо настроить библиотеку анализа или написать собственную процедуру проверки маркера, чтобы маркер соответствовал приведенным ниже требованиям.

1. Маркер был отправлен в заголовке HTTP `Authorization` со схемой носителя.
2. Маркер имеет допустимый формат JSON, соответствующий [стандарту JWT](#).
3. Маркер содержит утверждение "issuer" со значением  
`https://sts.windows.net/d6d49420-f39b-4df7-a1dc-d59a935871db/` или  
`https://sts.windows.net/f8cdef31-a31e-4b4a-93e4-5f571e91255a/`. (Проверка обоих значений "issuer" гарантирует, что эти значения проверяются в протоколе безопасности версий 3.1 и 3.2.)
4. Маркер содержит утверждение "audience" со значением, соответствующим идентификатору приложения Майкрософт для бота.
5. Маркер содержит утверждение "appid" со значением, соответствующим идентификатору приложения Майкрософт для бота.
6. Маркер является действующим. Отраслевая разница в показаниях часов составляет 5 минут.
7. Маркер имеет допустимый криптографическую подпись с ключом, приведенным в документе с ключами OpenID, который был получен на [шаге 3](#).

#### NOTE

Требование 5 характерно для пути проверки эмулятора.

Если маркер не соответствует всем требованиям, бот должен завершить запрос, возвратив код состояния **HTTP 403 (Запрещено)**.

#### IMPORTANT

Все эти требования являются важными, особенно требования 4 и 7. При невозможности выполнить все эти требования бот будет открыт для атак, что может привести к раскрытию его маркера JWT.

#### Запрос от эмулятора к боту: пример компонентов JWT

```
header:  
{  
  typ: "JWT",  
  alg: "RS256",  
  x5t: "<SIGNING KEY ID>",  
  kid: "<SIGNING KEY ID>"  
},  
payload:  
{  
  aud: "<YOUR MICROSOFT APP ID>",  
  iss: "https://sts.windows.net/d6d49420-f39b-4df7-a1dc-d59a935871db/",  

```

#### NOTE

На практике могут использоваться другие поля. Создайте и проверьте все маркеры JWT, как указано выше.

## Изменения протокола безопасности

#### WARNING

Поддержка протокола безопасности версии 3.0 была прекращена **31 июля 2017 г.** Если вы написали собственный код проверки подлинности (т. е. не использовали для создания бота пакет SDK Bot Framework), следует обновить протокол безопасности до версии 3.1 путем обновления приложения для использования приведенных ниже значений версии 3.1.

## Проверка подлинности бота в службе Bot Connector

URL-адрес входа OAuth

ВЕРСИЯ ПРОТОКОЛА	ДОПУСТИМОЕ ЗНАЧЕНИЕ
3.1 и 3.2	<a href="https://login.microsoftonline.com/botframework.com/oauth2/v2.0/token">https://login.microsoftonline.com/botframework.com/oauth2/v2.0/token</a>

Область OAuth

ВЕРСИЯ ПРОТОКОЛА	ДОПУСТИМОЕ ЗНАЧЕНИЕ
3.1 и 3.2	<a href="https://api.botframework.com/.default">https://api.botframework.com/.default</a>

## Проверка подлинности службы Bot Connector в боте

Документ метаданных OpenID

ВЕРСИЯ ПРОТОКОЛА	ДОПУСТИМОЕ ЗНАЧЕНИЕ
3.1 и 3.2	<a href="https://login.botframework.com/v1/.well-known/openidconfiguration">https://login.botframework.com/v1/.well-known/openidconfiguration</a>

Издатель JWT

ВЕРСИЯ ПРОТОКОЛА	ДОПУСТИМОЕ ЗНАЧЕНИЕ
3.1 и 3.2	<a href="https://api.botframework.com">https://api.botframework.com</a>

## Проверка подлинности эмулятора в боте

URL-адрес входа OAuth

ВЕРСИЯ ПРОТОКОЛА	ДОПУСТИМОЕ ЗНАЧЕНИЕ
3.1 и 3.2	<a href="https://login.microsoftonline.com/botframework.com/oauth2/v2.0/token">https://login.microsoftonline.com/botframework.com/oauth2/v2.0/token</a>

Область OAuth

ВЕРСИЯ ПРОТОКОЛА	ДОПУСТИМОЕ ЗНАЧЕНИЕ
3.1 и 3.2	Идентификатор приложения-робота (Microsoft) + /.default

Аудитория JWT

ВЕРСИЯ ПРОТОКОЛА	ДОПУСТИМОЕ ЗНАЧЕНИЕ
3.1 и 3.2	Идентификатор приложения для программы-робота Microsoft

Издатель JWT

ВЕРСИЯ ПРОТОКОЛА	ДОПУСТИМОЕ ЗНАЧЕНИЕ
3.1	<a href="https://sts.windows.net/d6d49420-f39b-4df7-a1dc-d59a935871db/">https://sts.windows.net/d6d49420-f39b-4df7-a1dc-d59a935871db/</a>
3.2	<a href="https://sts.windows.net/f8cdef31-a31e-4b4a-93e4-5f571e91255a/">https://sts.windows.net/f8cdef31-a31e-4b4a-93e4-5f571e91255a/</a>

Документ метаданных OpenID

ВЕРСИЯ ПРОТОКОЛА	ДОПУСТИМОЕ ЗНАЧЕНИЕ
3.1 и 3.2	<a href="https://login.microsoftonline.com/botframework.com/v2.0/.well-known/openid-configuration">https://login.microsoftonline.com/botframework.com/v2.0/.well-known/openid-configuration</a>

## Дополнительные ресурсы

- [Устранение неполадок проверки подлинности Bot Framework](#)
- [Принципы использования действий в Bot Framework](#)
- [JSON Web Token \(JWT\) draft-jones-json-web-token-07](#)
- [JSON Web Signature \(JWS\) draft-jones-json-web-signature-04](#)
- [Документ RFC 7517 для JSON Web Key \(JWK\)](#)

# Создание сообщений с помощью API соединителя Bot

21.09.2020 • 4 minutes to read • [Edit Online](#)

Бот будет отправлять объекты [Действие](#) типа `message` для передачи информации пользователям, а также будет получать действия `message` от пользователей. Некоторые сообщения могут состоять из простого текста, в то время как другие могут содержать более богатое содержимое, такое как [произносимый текст](#), [предлагаемые действия](#), [мультиданные вложения](#), [форматированные карточки](#) и [данные по каналу](#). В этой статье описаны некоторые из наиболее часто используемых свойств сообщений.

## Текст сообщения и форматирование

Текст сообщения можно форматировать с помощью `plain`, `markdown` или `xml`. Формат по умолчанию для свойства `textFormat` — `markdown`, который интерпретирует текст с использованием стандартов форматирования Markdown. Уровень поддержки формата текста меняется в зависимости от каналов.

### TIP

Описание функций, поддерживаемых в каждом канале, см. в статье [Разделенные на категории действия по каналам](#).

Свойство `textFormat` объекта [Действие](#) может использоваться для указания формата текста. Например, чтобы создать простое сообщение, которое содержит только обычный текст, задайте для свойства `textFormat` объекта [Activity](#) значение `plain`, для свойства `text` — значение содержимого сообщения, а для свойства `locale` — языковой стандарт отправителя.

## Вложения

Свойство `attachments` объекта [Действие](#) может использоваться для отправки простых мультиданных вложений (изображений, аудио, видео, файлов) и форматированных карточек. Дополнительные сведения см. в разделах [Добавление мультиданных вложений в сообщения](#) и [Добавление вложений в виде форматированных карточек в сообщения](#).

## Сущности

Свойство `entities` объекта [Действие](#) представляет собой массив открытых объектов [schema.org](#), который позволяет осуществлять обмен контекстно-зависимыми метаданными между каналом и ботом.

### Сущности Mention

Многие каналы поддерживают способность бота или пользователя "упоминать" кого-то в контексте общения. Чтобы упомянуть пользователя в сообщении, заполните свойство `entities` сообщения объектом [Mention](#).

### Сущность Place

Чтобы передать в сообщение [сведения, связанные с расположением](#), заполните свойство `entities` сообщения объектом [Place](#).

## Данные канала

Свойство `channelData` объекта [Действие](#) может использоваться для реализации функциональных возможностей канала. Дополнительные сведения см. в разделе [Реализация функциональных возможностей канала](#).

## Преобразование текста в речь

Свойство `speak` объекта [Действие](#) может использоваться для указания текста, который будет произноситься ботом по каналу с поддержкой речевых функций, а свойство `inputHint` объекта [Activity](#) может использоваться для воздействия на состояние микрофона клиента. Дополнительные сведения см. в разделах [Добавление речи в сообщения](#) и [Добавление подсказок ввода к сообщениям](#).

## Предлагаемые действия

Свойство `suggestedActions` объекта [Действие](#) может использоваться для представления кнопок, которых пользователь может коснуться для ввода данных. В отличие от кнопок, которые появляются в функциональных карточках (и которые остаются видимыми и доступными для пользователя даже после касания), кнопки, отображаемые в области предлагаемых действий, исчезнут, как только будет сделан выбор. Дополнительные сведения см. в разделе [Добавление предлагаемых действий к сообщениям](#).

## Дополнительные ресурсы

- [Справочник по каналам](#)
- [Общие сведения о действиях](#)
- [Отправка и получение сообщений](#)
- [Добавление мультимедийных вложений в сообщения](#)
- [Добавление вложений в виде форматированных карточек в сообщения](#)
- [Добавление речи в сообщения](#)
- [Добавление подсказок для ввода в сообщения](#)
- [Добавление предлагаемых действий в сообщения](#)
- [Реализация возможностей для определенных каналов](#)

# Отправка и получение сообщений с помощью API соединителя Bot

21.09.2020 • 7 minutes to read • [Edit Online](#)

Служба Bot Connector позволяет боту взаимодействовать через различные каналы связи, такие как электронная почта, Slack и др. Она обеспечивает обмен данными между ботом и пользователем, ретранслируя [действия](#) между ботом и каналом в двустороннем порядке. Каждое действие содержит сведения, используемые для маршрутизации сообщения в соответствующее место назначения, а также сведения о создателе, контексте и получателе сообщения. В этой статье описывается, как использовать службу Bot Connector для обмена действиями [сообщений](#) между ботом и пользователем через канал.

## Ответ на сообщение

### Создание ответа

Когда пользователь отправляет боту сообщение, этот бот получает сообщение в виде объекта [Действие](#) с типом `message`. Чтобы создать ответ на сообщение пользователя, создайте объект [Activity](#) и задайте ему следующие свойства:

СВОЙСТВО	ЗНАЧЕНИЕ
диалог	Присвойте этому свойству значение свойства <code>conversation</code> из сообщения пользователя.
из	Присвойте этому свойству значение свойства <code>recipient</code> из сообщения пользователя.
локаль	Присвойте этому свойству значение свойства <code>locale</code> из сообщения пользователя, если оно указано.
recipient	Присвойте этому свойству значение свойства <code>from</code> из сообщения пользователя.
replyToId	Присвойте этому свойству значение свойства <code>id</code> из сообщения пользователя.
type	Присвойте этому свойству значение <code>message</code> .

Затем задайте свойства, которые определяют передаваемую пользователю информацию. Например, свойство `text` позволяет указать текст, отображаемый в сообщении, свойство `speak` — произносимый ботом текст, а свойство `attachments` — включаемые в сообщение вложения мультимедиа или расширенные карточки. Дополнительные сведения о часто используемых свойствах сообщения см. в руководстве по [созданию сообщений](#).

### Отправка отчета

Используйте свойство `serviceUrl` во входящем действии, чтобы [определить базовый URI](#), который бот будет применять для отправки ответа.

Чтобы отправить ответ, выполните такой запрос:

```
POST /v3/conversations/{conversationId}/activities/{activityId}
```

В URI этого запроса замените `{conversationId}` значением `id` из объекта `conversation` для действия (ответа), а также замените `{activityId}` значением свойства `replyToId` для действия (ответа). В текст запроса поместите объект [Действие](#), созданный для представления ответа.

Следующий пример демонстрирует запрос, который отправляет простой текстовый ответ на сообщение пользователя. В этом примере запрос <https://smba.trafficmanager.net/apis> представляет базовый URI. Базовый URI для запросов, отправляемых вашим ботом, может отличаться. Дополнительные сведения о настройке базового URI см. в статье [Справочник по API](#).

```
POST https://smba.trafficmanager.net/apis/v3/conversations/abcd1234/activities/5d5cdc723
Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json
```

```
{
  "type": "message",
  "from": {
    "id": "12345678",
    "name": "Pepper's News Feed"
  },
  "conversation": {
    "id": "abcd1234",
    "name": "Convo1"
  },
  "recipient": {
    "id": "1234abcd",
    "name": "SteveW"
  },
  "text": "My bot's reply",
  "replyToId": "5d5cdc723"
}
```

## Отправка сообщения (не ответа)

Как правило, бот отправляет сообщения только в ответ на сообщения пользователя. Но иногда боту нужно отправить в диалоге сообщение, не являющееся прямым ответом на сообщение пользователя. Например, бот может начать беседу на новую тему или попрощаться с пользователем в конце беседы.

Чтобы отправить сообщение, не являющееся прямым ответом на сообщение пользователя, выполните такой запрос:

```
POST /v3/conversations/{conversationId}/activities
```

В URI запроса замените `{conversationId}` значением идентификатора беседы.

В текст запроса поместите объект [Действие](#), созданный для представления сообщения.

#### NOTE

Bot Framework не накладывает каких-либо ограничений на число сообщений, которые может отправлять бот. Тем не менее большинство каналов принудительно применяет ограничения регулирования, запрещая ботам отправлять большое число сообщений за короткий период времени. Кроме того, если бот отправляет несколько сообщений за короткий промежуток времени, канал не всегда может обрабатывать сообщения в правильной последовательности.

## Начало общения

Возможны ситуации, когда боту требуется инициировать беседу с одним или несколькими пользователями. Чтобы начать беседу с пользователем по определенному каналу, бот должен иметь сведения о своей учетной записи и об учетной записи пользователя в соответствующем канале.

#### TIP

Если в будущем боту может потребоваться снова начать беседу с пользователями, сохраните в кэш данные о соответствующих учетных записях и другие важные сведения, например персональные настройки и языковой стандарт, а также URL-адрес службы (чтобы создать базовый URI для запроса на создание беседы).

Чтобы начать беседу, выполните такой запрос:

```
POST /v3/conversations
```

В текст запроса поместите объект [ConversationParameters](#), содержащий сведения об учетной записи бота и учетной записи пользователя или пользователей, которых нужно включить в беседу.

#### NOTE

Не все каналы поддерживают групповые беседы. Изучите документацию по используемому каналу, чтобы узнать о поддержке групповых бесед и ограничениях на число участников в них.

В следующем примере показан запрос, который начинает беседу: В этом примере запрос

`https://smiba.trafficmanager.net/apis` представляет базовый URI. Базовый URI для запросов, отправляемых вашим ботом, может отличаться. Дополнительные сведения о настройке базового URI см. в статье [Справочник по API](#).

```
POST https://smiba.trafficmanager.net/apis/v3/conversations
Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json
```

```
{  
  "bot": {  
    "id": "12345678",  
    "name": "bot's name"  
  },  
  "isGroup": false,  
  "members": [  
    {  
      "id": "1234abcd",  
      "name": "recipient's name"  
    }  
  ],  
  "topicName": "News Alert"  
}
```

Если беседа с указанными пользователями создана, ответ будет содержать идентификатор этой беседы.

```
{  
  "id": "abcd1234"  
}
```

Теперь бот может использовать идентификатор беседы для [отправки сообщений](#) пользователям в этой беседе.

## Дополнительные ресурсы

- [Создание сообщений](#)
- [Принципы использования действий в Bot Framework](#)

# Добавление вложений мультимедиа в сообщения с помощью API соединителя Bot

27.10.2020 • 5 minutes to read • [Edit Online](#)

Боты и каналы обычно обмениваются текстовыми строками, но некоторые каналы также поддерживают обмен вложениями, что позволяет боту отправлять пользователям сообщения с более широким набором возможностей. Например, бот может отправлять вложения мультимедиа (такие как изображения, видео, звук и файлы) и [форматированные карточки](#). В этой статье описывается добавление вложений мультимедиа в сообщения с помощью службы соединителя ботов.

## TIP

Описание функций, поддерживаемых в каждом канале, см. в статье [Разделенные на категории действия по каналам](#).

## Добавление мультимедийного вложения

Чтобы добавить мультимедийное вложение в сообщение, создайте объект [Вложение](#), задайте свойство `name`, задайте для свойства `contentUrl` URL-адрес файла мультимедиа, а для свойства `contentType` — соответствующий тип мультимедиа (например, `image/png`, `audio/wav`, `video/mp4`). Затем в объекте [Действие](#), представляющем сообщение, укажите объект `Attachment` в массиве `attachments`.

Следующий пример демонстрирует запрос, который отправляет сообщение, содержащее текст и одно вложенное изображение. В этом примере запрос `https://smba.trafficmanager.net/apis` представляет базовый URI. Базовый URI для запросов, отправляемых вашим ботом, может отличаться. Дополнительные сведения о настройке базового URI см. в статье [Справочник по API](#).

```
POST https://smba.trafficmanager.net/apis/v3/conversations/abcd1234/activities/5d5cdc723
Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json
```

```
{
    "type": "message",
    "from": {
        "id": "12345678",
        "name": "sender's name"
    },
    "conversation": {
        "id": "abcd1234",
        "name": "conversation's name"
    },
    "recipient": {
        "id": "1234abcd",
        "name": "recipient's name"
    },
    "text": "Here's a picture of the duck I was telling you about.",
    "attachments": [
        {
            "contentType": "image/png",
            "contentUrl": "https://aka.ms/DuckOnARock",
            "name": "duck-on-a-rock.jpg"
        }
    ],
    "replyToId": "5d5cdc723"
}
```

Для каналов, которые поддерживают встроенные двоичные файлы изображений, можно задать свойство `contentUrl` объекта `Attachment` как двоичный файл base64 изображения (например, `data:image/png;base64,iVBORw0KGgo...`). Канал будет отображать изображение или URL-адрес изображения рядом со строкой текста сообщения.

```
{
    "type": "message",
    "from": {
        "id": "12345678",
        "name": "sender's name"
    },
    "conversation": {
        "id": "abcd1234",
        "name": "conversation's name"
    },
    "recipient": {
        "id": "1234abcd",
        "name": "recipient's name"
    },
    "text": "Here's a picture of the duck I was telling you about.",
    "attachments": [
        {
            "contentType": "image/png",
            "contentUrl": "data:image/png;base64,iVBORw0KGgo...",
            "name": "duck-on-a-rock.jpg"
        }
    ],
    "replyToId": "5d5cdc723"
}
```

В сообщение можно вложить файл видео или звуковой файл, используя ту же процедуру, которая описана выше для файла изображения. В зависимости от канала аудио- и видеоданные могут воспроизводиться внутри сообщения или отображаться в виде ссылки.

#### NOTE

Бот также может получать сообщения, содержащие мультимедийные вложения. Например, сообщение, полученное ботом, может содержать вложение, если канал позволяет пользователю отправлять фотографию для анализа или документ для сохранения.

## Добавление вложения AudioCard

Добавление вложения **AudioCard** или **VideoCard** ничем не отличается от добавления вложения мультимедиа. Например, следующий пример кода JSON демонстрирует добавление карточки аудиофайла в мультимедийное вложение.

```
{  
    "type": "message",  
    "from": {  
        "id": "12345678",  
        "name": "sender's name"  
    },  
    "conversation": {  
        "id": "abcd1234",  
        "name": "conversation's name"  
    },  
    "recipient": {  
        "id": "1234abcd",  
        "name": "recipient's name"  
    },  
    "attachments": [  
        {  
            "contentType": "application/vnd.microsoft.card.audio",  
            "content": {  
                "title": "Allegro in C Major",  
                "subtitle": "Allegro Duet",  
                "text": "No Image, No Buttons, Autoloop, Autostart, Sharable",  
                "duration": "PT2M55S",  
                "media": [  
                    {  
                        "url": "https://contoso.com/media/AllegrofromDuetinCMajor.mp3"  
                    }  
                ],  
                "shareable": true,  
                "autoloop": true,  
                "autostart": true,  
                "value": {  
                    // Supplementary parameter for this card  
                }  
            }  
        }  
    ],  
    "replyToId": "5d5cdc723"  
}
```

Как только канал получит это вложение, он начнет воспроизведение звукового файла. Если пользователь взаимодействует со звуковым файлом, например нажимая кнопку **Пауза**, канал отправит обратный вызов боту с примерно следующим запросом JSON:

```
{
  ...
  "type": "event",
  "name": "media/pause",
  "value": {
    "url": // URL for media
    "cardValue": {
      // Supplementary parameter for this card
    }
  }
}
```

Имя события мультимедиа `media/pause` появится в поле `activity.name`. Список всех имен событий мультимедиа см. в таблице ниже.

СОБЫТИЕ	ОПИСАНИЕ
<code>media/next</code>	Клиент перешел к следующему файлу мультимедиа.
<code>media/pause</code>	Клиент приостановил воспроизведение мультимедиа.
<code>media/play</code>	Клиент запустил воспроизведение мультимедиа.
<code>media/previous</code>	Клиент перешел к предыдущему файлу мультимедиа.
<code>media/resume</code>	Клиент возобновил воспроизведение мультимедиа.
<code>media/stop</code>	Клиент остановил воспроизведение мультимедиа.

## Дополнительные ресурсы

- [Создание сообщений](#)
- [Отправка и получение сообщений](#)
- [Добавление вложений в виде форматированных карточек в сообщения](#)
- [Принципы использования действий в Bot Framework](#)
- [Принципы использования карточек в Bot Framework](#)

# Добавление в сообщения полнофункциональных вложений карты с помощью API соединителя Bot

27.03.2021 • 11 minutes to read • [Edit Online](#)

Некоторые каналы позволяют вашему роботу отправить пользователям *привлекательные карты*.

Широкая карта — это вложение, которое содержит интерактивные элементы, такие как кнопки, текст, изображения, аудио, видео и т. д.

В этой статье описывается создание обширных карточек, которые можно вложить в сообщение.

Инструкции по добавлению вложения в сообщение см. в разделе [Добавление вложений к сообщениям](#).

## Типы функциональных карточек

Форматированная карточка содержит название, описание, ссылку и изображения. Сообщение может содержать несколько форматированных карточек, которые отображаются в виде списка или карусели. Сейчас Bot Framework поддерживает восемь типов форматированных карточек.

ТИП КАРТОЧКИ	ОПИСАНИЕ
<a href="#">AdaptiveCard</a>	Настраиваемая карточка, которая может содержать любое сочетание текста, речи, изображений, кнопок и полей для ввода. См. описание <a href="#">поддержки для каждого канала</a> .
<a href="#">AnimationCard</a>	Карточка, которая может воспроизводить GIF-файлы с анимацией или короткие видеоролики.
<a href="#">AudioCard</a>	Карточка, которая может воспроизводить звуковой файл.
<a href="#">HeroCard</a> ;	Карточка, которая обычно содержит одно большое изображение, одну или несколько кнопок и текст.
<a href="#">ThumbnailCard</a> .	Карточка, которая обычно содержит один эскиз, одну или несколько кнопок и текст.
<a href="#">ReceiptCard</a> ;	Карточка, с помощью которой бот выдает квитанцию пользователю. Обычно она содержит список элементов, включаемых в квитанцию, налог, а также общую информацию и другой текст.
<a href="#">SignInCard</a>	Карточка, в которой бот запрашивает вход пользователя. Обычно она содержит текст и одну или несколько кнопок, которые можно нажать, чтобы начать процесс входа.
<a href="#">VideoCard</a>	Карточка, которая может воспроизводить видео.

**TIP**

Описание функций, поддерживаемых в каждом канале, см. в статье [Разделенные на категории действия по каналам](#).

## Обработка событий в форматированных карточках

Для обработки событий в форматированных карточках используйте объекты `CardAction`, чтобы указать, что должно происходить, когда пользователь нажимает кнопку или касается сегмента карточки. Каждый объект `CardAction` содержит следующие свойства.

СВОЙСТВО	ТИП	ОПИСАНИЕ
channelData	строка	Относящиеся к каналу данные, связанные с этим действием.
displayText	строка	Текст, отображаемый в канале чата при нажатии кнопки.
text	строка	Текст для действия.
type	строка	тип действия (одно из значений, указанных в таблице ниже)
title	строка	название кнопки
Изображение	строка	URL-адрес изображения для кнопки
value	строка	значение, необходимое для выполнения указанного типа действия

**NOTE**

Для создания кнопок в адаптивных карточках используются не объекты `CardAction`, а схема, которая определяется адаптивными карточками. О том, как добавить кнопку в адаптивную карточку, см. в разделе [Добавление адаптивной карточки в сообщение](#).

Чтобы правильно работать, назначьте тип действия каждому элементу, который можно щелкнуть, на карточке Hero. В этой таблице перечислены и описаны доступные типы действий и требуемый формат для связанного свойства `messageBack`. Действие карточки имеет более обобщенное значение, чем другие действия с картой. Дополнительные сведения о типах действий карты см. в разделе [действие карты](#) [схемы действий](#) `messageBack`.

ТИП	ОПИСАНИЕ	ЗНАЧЕНИЕ
вызывает	Инициирует телефонный звонок.	Целевое назначение телефонного звонка в следующем формате: <code>tel:123123123123</code> .
downloadFile	Скачивает файл.	URL-адрес для скачивания файла.

ТИП	ОПИСАНИЕ	ЗНАЧЕНИЕ
imBack	Отправляет боту сообщение и отображает полученный ответ в чате.	Текст отправляемого сообщения.
messageback	Представляет текстовый ответ, отправляемый через систему разговора.	Необязательное программное значение, включаемое в создаваемые сообщения.
openUrl	Открывает URL-адрес в окне встроенного браузера.	URL-адрес, который нужно открыть.
playAudio	Воспроизводит звук.	URL-адрес для воспроизведения звука.
playVideo	Воспроизводит видео.	URL-адрес для воспроизведения видео.
postBack	Отправляет боту сообщение, но не всегда отображает полученный ответ в чате.	Текст отправляемого сообщения.
showImage	Отображает изображение.	URL-адрес для отображения изображения.
signin	Инициирует процесс входа OAuth.	URL-адрес потока OAuth, который нужно запустить.

## Добавление имиджевой карточки в сообщение

Добавление к сообщению вложения карточки с широкими возможностями:

1. Создайте объект, представляющий [тип карты](#), которую необходимо добавить в сообщение.
2. Создайте объект [вложения] [] :
  - Задайте `contentType` для свойства Тип носителя карточки.
  - Задайте `content` для его свойства объект, который вы создали для представления карточки.
3. Добавьте `Attachment` объект в `attachments` массив сообщения.

### TIP

Сообщения, содержащие вложения в виде форматированных карточек, обычно не указывают `text`.

На некоторых каналах можно добавлять несколько форматированных карточек в массив `attachments` сообщения. Это может быть полезно в сценариях, где необходимо предоставить пользователю несколько параметров. Например, если бот позволяет пользователям забронировать номера в гостинице, то он может предоставить список форматированных карточек, в которых будут показаны типы доступных номеров. Каждая карточка может содержать изображение и список удобств, соответствующих типу комнаты, а пользователь может выбрать тип комнаты, нажав на кнопку или щелкнув карточку.

**TIP**

Чтобы отобразить несколько форматированных карточек в виде списка, задайте свойству `attachmentLayout` объекта `Действие` значение `list`. Чтобы отобразить несколько форматированных карточек в виде карусели, задайте свойству `attachmentLayout` объекта `Activity` значение `carousel`. Если канал не поддерживает формат карусели, форматированные карточки будут отображены в виде списка, даже если свойству `attachmentLayout` задано значение `carousel`.

В следующем примере показано полное сообщение, содержащее одно вложение карточки Hero. В этом примере запрос `https://smba.trafficmanager.net/apis` представляет базовый URI. Базовый URI для запросов, отправляемых вашим ботом, может отличаться. Дополнительные сведения о настройке базового URI см. в статье [Справочник по API](#).

```
POST https://smba.trafficmanager.net/apis/v3/conversations/abcd1234/activities/5d5cdc723
Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json
```

```
{
  "type": "message",
  "from": {
    "id": "12345678",
    "name": "sender's name"
  },
  "conversation": {
    "id": "abcd1234",
    "name": "conversation's name"
  },
  "recipient": {
    "id": "1234abcd",
    "name": "recipient's name"
  },
  "attachments": [
    {
      "contentType": "application/vnd.microsoft.card.hero",
      "content": {
        "title": "title goes here",
        "subtitle": "subtitle goes here",
        "text": "descriptive text goes here",
        "images": [
          {
            "url": "https://aka.ms/DuckOnARock",
            "alt": "picture of a duck",
            "tap": {
              "type": "playAudio",
              "value": "url to an audio track of a duck call goes here"
            }
          }
        ],
        "buttons": [
          {
            "type": "playAudio",
            "title": "Duck Call",
            "value": "url to an audio track of a duck call goes here"
          },
          {
            "type": "openUrl",
            "title": "Watch Video",
            "image": "https://aka.ms/DuckOnARock",
            "value": "url goes here of the duck in flight"
          }
        ]
      }
    ],
    "replyToId": "5d5cdc723"
  }
}
```

## Добавление адаптивной карточки в сообщение

Адаптивная карточка может содержать любое сочетание текста, речи, изображений, кнопок и полей для ввода. Адаптивные карточки создаются в формате JSON (см. [здесь](#)), что позволяет получить больший контроль над содержимым и форматом карточек.

Ознакомьтесь со сведениями на веб-сайте [адаптивных карточек](#), чтобы получить представление о схеме адаптивных карточек, изучить элементы адаптивных карточек и просмотреть примеры JSON, которые можно использовать для создания карточек различного состава и уровня сложности. А воспользовавшись интерактивным визуализатором, вы сможете разрабатывать соответствующие полезные нагрузки и просматривать выходные данные карточки.

В следующем примере показан один объект адаптивного вложения карты, представляющий назначение

работы. Чтобы отправить его пользователю, необходимо добавить его в качестве вложения в сообщение.

```
{  
    "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",  
    "type": "AdaptiveCard",  
    "version": "1.0",  
    "body": [  
        {  
            "type": "Container",  
            "items": [  
                {  
                    "type": "TextBlock",  
                    "text": "Publish Adaptive Card schema",  
                    "weight": "bolder",  
                    "size": "medium"  
                },  
                {  
                    "type": "ColumnSet",  
                    "columns": [  
                        {  
                            "type": "Column",  
                            "width": "auto",  
                            "items": [  
                                {  
                                    "type": "Image",  
                                    "url": "https://pbs.twimg.com/profile_images/3647943215/d7f12830b3c17a5a9e4afcc370e3a37e_400x400.jpeg",  
                                    "size": "small",  
                                    "style": "person"  
                                }  
                            ]  
                        },  
                        {  
                            "type": "Column",  
                            "width": "stretch",  
                            "items": [  
                                {  
                                    "type": "TextBlock",  
                                    "text": "Matt Hidinger",  
                                    "weight": "bolder",  
                                    "wrap": true  
                                },  
                                {  
                                    "type": "TextBlock",  
                                    "spacing": "none",  
                                    "text": "Created {{DATE(2017-02-14T06:08:39Z, SHORT)}}",  
                                    "isSubtle": true,  
                                    "wrap": true  
                                }  
                            ]  
                        }  
                    ]  
                },  
                {  
                    "type": "Container",  
                    "items": [  
                        {  
                            "type": "TextBlock",  
                            "text": "Now that we have defined the main rules and features of the format, we need to produce a schema and publish it to GitHub. The schema will be the starting point of our reference documentation.",  
                            "wrap": true  
                        },  
                        {  
                            "type": "FactSet",  
                            "facts": [
```

```

    "facts": [
      {
        "title": "Board:",
        "value": "Adaptive Card"
      },
      {
        "title": "List:",
        "value": "Backlog"
      },
      {
        "title": "Assigned to:",
        "value": "Matt Hidinger"
      },
      {
        "title": "Due date:",
        "value": "Not set"
      }
    ]
  ],
  "actions": [
    {
      "type": "Action.ShowCard",
      "title": "Comment",
      "card": {
        "type": "AdaptiveCard",
        "body": [
          {
            "type": "Input.Text",
            "id": "comment",
            "isMultiline": true,
            "placeholder": "Enter your comment"
          }
        ],
        "actions": [
          {
            "type": "Action.Submit",
            "title": "OK"
          }
        ]
      }
    },
    {
      "type": "Action.OpenUrl",
      "title": "View",
      "url": "https://adaptivecards.io"
    }
  ]
}

```

Результирующая карточка содержит заголовок, сведения о пользователе, создавшем карточку (имя и аватар), время создания карточки, описание назначения задания и сведения, связанные с назначением. Кроме того, доступны две кнопки, позволяющие просмотреть назначение задания или добавить комментарий.

## Publish Adaptive Card schema



Matt Hidinger

Created Mon, Feb 13, 2017

Now that we have defined the main rules and features of the format, we need to produce a schema and publish it to GitHub. The schema will be the starting point of our reference documentation.

**Board:** Adaptive Card

**List:** Backlog

**Assigned to:** Matt Hidinger

**Due date:** Not set

[Comment](#)

[View](#)

## Дополнительные ресурсы

- [Создание сообщений](#)
- [Отправка и получение сообщений](#)
- [Добавление мультимедийных вложений в сообщения](#)
- [Справочник по использованию каналов](#)
- [Принципы использования действий в Bot Framework](#)
- [Схема карт инфраструктуры Bot](#)

# Добавление речи в сообщения с помощью API соединителя Bot

27.10.2020 • 3 minutes to read • [Edit Online](#)

При создании бота для канала с поддержкой речевых функций, такого как Cortana, можно сформировать сообщения, в которых указывается произносимый ботом текст. Можно также попытаться повлиять на состояние микрофона клиента, задав [подсказку для ввода](#), чтобы указать, что бот принимает, ожидает или игнорирует ввод данных пользователем.

Вы можете настроить бот так, чтобы клиентские приложения могли обмениваться данными с ним через [канал Direct Line Speech](#).

## Указание текста, произносимого ботом

Чтобы указать текст, который бот будет произносить по каналу с поддержкой речи, задайте свойство `speak` в объекте [Activity](#), который представляет ваше сообщение. Можно установить свойство `speak` в текстовую строку или строку, которая отформатирована как [Speech Synthesis Markup Language \(SSML\)](#) (Язык разметки синтеза речи) — язык разметки на основе XML, который позволяет управлять различными характеристиками речи бота, такими как голос, скорость, громкость, произношение, тон и другое. Если канал не поддерживает такое поведение, сообщение доставляется как текст.

В следующем примере запроса показана отправка сообщения, которое задает текст для отображения и текст для произнесения и указывает, что бот [принимает входные данные пользователя](#). Он определяет свойство `speak`, используя формат [SSML](#), чтобы указать, что слово "sure" должно быть произнесено с умеренным количеством акцентов. В этом примере запроса Direct Line представляет базовый URI.

Базовый URI для запросов, отправляемых вашим ботом, может отличаться. Дополнительные сведения о настройке базового URI см. в статье [Справочник по API](#).

```
POST https://smba.trafficmanager.net/apis/v3/conversations/abcd1234/activities/5d5cdc723
Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json
```

```
{
  "type": "message",
  "from": {
    "id": "12345678",
    "name": "sender's name"
  },
  "conversation": {
    "id": "abcd1234",
    "name": "conversation's name"
  },
  "recipient": {
    "id": "1234abcd",
    "name": "recipient's name"
  },
  "text": "Are you sure that you want to cancel this transaction?",
  "speak": "<speak version=\"1.0\" xmlns=\"http://www.w3.org/2001/10/synthesis\" xml:lang=\"en-US\">Are
you <emphasis level=\"moderate\">sure</emphasis> that you want to cancel this transaction?</speak>",
  "inputHint": "expectingInput",
  "replyToId": "5d5cdc723"
}
```

## Подсказки для ввода

При отправке сообщения в канале, поддерживающем распознавание речи, можно выразить предполагаемое состояние микрофона клиента, включив в него подсказку, указывающую, принимает ли программа-Bot ввод, ожидает или игнорирует введенные пользователем данные. Дополнительные сведения см. в статье [Добавление подсказок для ввода в сообщения](#).

## Дополнительные ресурсы

- [Создание сообщений](#)
- [Отправка и получение сообщений](#)
- [Добавление подсказок для ввода в сообщения](#)
- [Язык разметки синтеза речи \(Speech Synthesis Markup Language, SSML\)](#)

# Добавление входных подсказок к сообщениям с помощью API соединителя Bot

27.03.2021 • 4 minutes to read • [Edit Online](#)

Задав подсказку для ввода, можно указать, какое действие бот выполнит после того, как сообщение будет доставлено клиенту: примет, будет ожидать или пропустит ввод данных пользователем. При использовании каналов с поддержкой этого поля это позволяет клиентам соответствующим образом задавать состояние элементов управления ввода данных пользователем. Например, если подсказка для ввода указывает, что бот пропускает ввод данных пользователем, клиент может закрыть микрофон и отключить поле ввода, чтобы предотвратить ввод данных пользователем.

## Принятие ввода данных

Чтобы указать, что бот пассивно принимает входные данные, но не ожидает от пользователя ответа, задайте для свойства `inputHint` значение `acceptingInput` в объекте [Действие](#), который представляет сообщение. В результате во многих каналах поле ввода будет включено, а микрофон закрыт, но доступен пользователю. Например, Кортана откроет микрофон, чтобы принять входные данные пользователя, если пользователь будет удерживать кнопку микрофона.

В следующем примере показан запрос, который отправляет сообщение, и указывает, что бот готов принять входные данные. В этом примере запроса Direct Line представляет базовый URI. Базовый URI для запросов, отправляемых вашим ботом, может отличаться. Дополнительные сведения о настройке базового URI см. в статье [Справочник по API](#).

```
POST https://smba.trafficmanager.net/apis/v3/conversations/abcd1234/activities/5d5cdc723
Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json
```

```
{
  "type": "message",
  "from": {
    "id": "12345678",
    "name": "sender's name"
  },
  "conversation": {
    "id": "abcd1234",
    "name": "conversation's name"
  },
  "recipient": {
    "id": "1234abcd",
    "name": "recipient's name"
  },
  "text": "Here's a picture of the house I was telling you about.",
  "inputHint": "acceptingInput",
  "replyToId": "5d5cdc723"
}
```

## Ожидание ввода данных

Чтобы указать, что бот активно ожидает от пользователя ответа, задайте для свойства `inputHint` значение `expectingInput` в объекте [Действие](#), который представляет сообщение. При использовании

каналов с поддержкой такого поведения поле ввода клиента будет недоступным, а микрофон — выключен.

В следующем примере показан запрос, который отправляет сообщение и указывает, что бот ожидает ввода. В этом примере запроса Direct Line представляет базовый URI. Базовый URI для запросов, отправляемых вашим ботом, может отличаться. Дополнительные сведения о настройке базового URI см. в статье [Справочник по API](#).

```
POST https://smba.trafficmanager.net/apis/v3/conversations/abcd1234/activities/5d5cdc723
Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json
```

```
{
  "type": "message",
  "from": {
    "id": "12345678",
    "name": "sender's name"
  },
  "conversation": {
    "id": "abcd1234",
    "name": "conversation's name"
  },
  "recipient": {
    "id": "1234abcd",
    "name": "recipient's name"
  },
  "text": "What is your favorite color?",
  "inputHint": "expectingInput",
  "replyToId": "5d5cdc723"
}
```

## Пропуск ввода данных

Чтобы указать, что бот не готов принимать входные данные от пользователя, задайте для свойства `inputHint` значение `ignoringInput` в объекте [Действие](#), который представляет сообщение. При использовании каналов с поддержкой такого поведения поле ввода клиента будет недоступным, а микрофон — выключен.

В следующем примере показан запрос, который отправляет сообщение, и указывает, что бот игнорирует входные данные. В этом примере запроса Direct Line представляет базовый URI. Базовый URI для запросов, отправляемых вашим ботом, может отличаться. Дополнительные сведения о настройке базового URI см. в статье [Справочник по API](#).

```
POST https://smba.trafficmanager.net/apis/v3/conversations/abcd1234/activities/5d5cdc723
Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json
```

```
{  
    "type": "message",  
    "from": {  
        "id": "12345678",  
        "name": "sender's name"  
    },  
    "conversation": {  
        "id": "abcd1234",  
        "name": "conversation's name"  
    },  
    "recipient": {  
        "id": "1234abcd",  
        "name": "recipient's name"  
    },  
    "text": "Please hold while I perform the calculation.",  
    "inputHint": "ignoringInput",  
    "replyToId": "5d5cdc723"  
}
```

## Дополнительные ресурсы

- [Создание сообщений](#)
- [Отправка и получение сообщений](#)

# Добавление предлагаемых действий в сообщения с помощью API соединителя Bot

27.10.2020 • 2 minutes to read • [Edit Online](#)

Предлагаемые действия позволяют боту представлять кнопки, с помощью которых пользователь может вводить данные. Предлагаемые действия появляются рядом со средством создания. Они удобны тем, что пользователи могут отвечать на вопросы или выбирать нужные варианты простым касанием кнопки вместо того, чтобы вводить ответ с помощью клавиатуры. В отличие от кнопок, которые появляются в функциональных карточках (и которые остаются видимыми и доступными для пользователя даже после касания), кнопки, отображаемые в области предлагаемых действий, исчезнут, как только будет сделан выбор. Это происходит для того, чтобы пользователь не касался устаревших кнопок в диалоге. Кроме того, это упрощает разработку ботов (так как для этого сценария учетная запись не требуется).

## Отправка предлагаемых действий

Чтобы добавить предлагаемые действия к сообщению, задайте свойство `suggestedActions` для объекта [Действие](#), чтобы указать список объектов [CardAction](#), представляющих кнопки, которые будут показаны пользователю.

Следующий запрос отправляет пользователю сообщение, которое предлагает три действия. В этом примере запрос `https://smba.trafficmanager.net/apis` представляет базовый URI. Базовый URI для запросов, отправляемых вашим ботом, может отличаться. Дополнительные сведения о настройке базового URI см. в статье [Справочник по API](#).

```
POST https://smba.trafficmanager.net/apis/v3/conversations/abcd1234/activities/5d5cdc723
Authorization: Bearer ACCESS_TOKEN
Content-Type: application/json
```

```
{  
    "type": "message",  
    "from": {  
        "id": "12345678",  
        "name": "sender's name"  
    },  
    "conversation": {  
        "id": "abcd1234",  
        "name": "conversation's name"  
    },  
    "recipient": {  
        "id": "1234abcd",  
        "name": "recipient's name"  
    },  
    "text": "I have colors in mind, but need your help to choose the best one.",  
    "inputHint": "expectingInput",  
    "suggestedActions": {  
        "actions": [  
            {  
                "type": "imBack",  
                "title": "Blue",  
                "value": "Blue"  
            },  
            {  
                "type": "imBack",  
                "title": "Red",  
                "value": "Red"  
            },  
            {  
                "type": "imBack",  
                "title": "Green",  
                "value": "Green"  
            }  
        ]  
    },  
    "replyToId": "5d5cdc723"  
}
```

Когда пользователь выбирает одно из предложенных действий, бот получает сообщение от пользователя, которое содержит значение `value` соответствующего действия.

## Дополнительные ресурсы

- [Создание сообщений](#)
- [Отправка и получение сообщений](#)

# Реализация функций, зависящих от канала, с помощью API соединителя Bot

27.03.2021 • 8 minutes to read • [Edit Online](#)

Некоторые каналы предоставляют функции, которые невозможно реализовать, используя только [текст сообщений и вложения](#). Чтобы реализовать функции, связанные с каналами, вы можете передать в канал собственные метаданные через свойство `channelData` объекта `[Activity[]]`. Например, с помощью свойства `channelData` бот может передать в Telegram команду отправки наклейки или потребовать, чтобы Office 365 отправил сообщение электронной почты.

В этой статье описано, как реализовать функции, связанные с каналами, на основе свойства `channelData` в действии сообщения.

CHANNEL	ФУНКЦИОНАЛЬНОСТЬ
Email	Отправка и получение сообщений электронной почты, которые содержат текст, тему и метаданные о важности.
Slack	Отправка сообщений Slack с полным контролем.
Facebook	Отправка уведомлений Facebook из кода приложения.
Telegram	Выполнение действий, реализованных в Telegram, таких как публикация голосового напоминания или наклейки.
Kik	Отправка и получение сообщений Kik.

## NOTE

Значение свойства `channelData` объекта `Activity` — это объект JSON. Структура этого объекта JSON будет разной в зависимости от канала и реализованных функций, как описано ниже.

## Создание пользовательского сообщения электронной почты

Чтобы создать сообщение электронной почты, присвойте свойству `channelData` объекта `Activity` объект JSON, содержащий следующие свойства.

СВОЙСТВО	ОПИСАНИЕ
<code>htmlBody</code>	HTML-код, используемый для текста сообщения.
<code>subject</code>	Тема сообщения.
<code>importance</code>	Флаг важности, используемый для сообщения: <code>low</code> , <code>normal</code> или <code>high</code> .

СВОЙСТВО	ОПИСАНИЕ
toRecipients	Строка адреса электронной почты, разделенная точками с запятой (;) для добавления к полю "Кому" сообщения.
ccRecipients	Строка адреса электронной почты, разделенная точками с запятой (;) для добавления к полю "Копия" сообщения.
bccRecipients	Строка адреса электронной почты, разделенная точками с запятой (;) для добавления к полю "Скрытая копия" сообщения.

В этом фрагменте кода демонстрируется свойство `channelData` для сообщения электронной почты.

```
"channelData":  
{  
    "htmlBody": "<html><body style = /"font-family: Calibri; font-size: 11pt;/" >This is more than awesome.  
  </body></html>",  
    "subject": "Super awesome message subject",  
    "importance": "high",  
    "ccRecipients": "Yasemin@adatum.com;Temel@adventure-works.com"  
}
```

## Создание сообщение Slack с полным контролем

Чтобы создать сообщение Slack с полным контролем, присвойте свойству `channelData` объекта `Activity` объект JSON, который определяет [сообщения](#), [вложения](#) и (или) [кнопки](#) Slack.

### NOTE

Для включения поддержки кнопок в сообщениях Slack необходимо включить **интерактивные сообщения** при [подключении бота](#) к каналу Slack.

В этом фрагменте кода демонстрируется свойство `channelData` для пользовательского сообщения Slack.

```

"channelData": {
    "text": "Now back in stock! :tada:",
    "attachments": [
        {
            "title": "The Further Adventures of Slackbot",
            "author_name": "Stanford S. Strickland",
            "author_icon": "https://api.slack.com/img/api/homepage_custom_integrations-2x.png",
            "image_url": "http://i.imgur.com/OJkaVOI.jpg?1"
        },
        {
            "fields": [
                {
                    "title": "Volume",
                    "value": "1",
                    "short": true
                },
                {
                    "title": "Issue",
                    "value": "3",
                    "short": true
                }
            ]
        },
        {
            "title": "Synopsis",
            "text": "After @episod pushed exciting changes to a devious new branch back in Issue 1, Slackbot notifies @don about an unexpected deploy..."
        },
        {
            "fallback": "Would you recommend it to customers?",
            "title": "Would you recommend it to customers?",
            "callback_id": "comic_1234_xyz",
            "color": "#3AA3E3",
            "attachment_type": "default",
            "actions": [
                {
                    "name": "recommend",
                    "text": "Recommend",
                    "type": "button",
                    "value": "recommend"
                },
                {
                    "name": "no",
                    "text": "No",
                    "type": "button",
                    "value": "bad"
                }
            ]
        }
    ]
}

```

Когда пользователь нажмет кнопку в сообщении Slack, бот получит ответное сообщение, в котором свойство `channelData` содержит объект JSON `payload`. Этот объект `payload` определяет содержимое исходного сообщения, нажатую кнопку и идентификатор пользователя, который нажал эту кнопку.

В этом фрагменте кода показан пример свойства `channelData` в сообщении, которое бот получает при нажатии кнопки в сообщении Slack.

```

"channelData": {
    "payload": {
        "actions": [
            {
                "name": "recommend",
                "value": "yes"
            }
        ],
        ...
    },
    "original_message": "...",
    "response_url": "https://hooks.slack.com/actions/..."
}
}

```

Бот может ответить на это сообщение [обычным способом](#) или отправить ответ напрямую на конечную точку, которая определена свойством `response_url` объекта `payload`. Сведения том, как и в каких случаях следует отправлять ответ для `response_url`, см. в документации по [кнопкам Slack](#).

## Создание оповещения Facebook

Чтобы создать оповещение Facebook, присвойте свойству `channelData` объекта `Activity` объект JSON, содержащий следующие свойства.

СВОЙСТВО	ОПИСАНИЕ
<code>notification_type</code>	Тип уведомления (например, <code>REGULAR</code> , <code>SILENT_PUSH</code> , <code>NO_PUSH</code> ).
<code>attachment</code>	Вложение, которое содержит изображение, видео или другой тип мультимедиа, уведомление или другое шаблонное вложение, например квитанция.

### NOTE

Дополнительные сведения о формате и содержании свойств `notification_type` и `attachment` см. в документации по [API Facebook](#).

В этом фрагменте кода демонстрируется свойство `channelData` для вложения Facebook.

```

"channelData": {
    "notification_type": "NO_PUSH",
    "attachment": {
        "type": "template",
        "payload": {
            "template_type": "receipt",
            ...
        }
    }
}

```

## Создание сообщения Telegram

Чтобы создать сообщение, которое реализует специальные действия Telegram, например предоставление в совместный доступ голосового напоминания или наклейки, присвойте свойству `channelData` объекта `Activity` объект JSON, который определяет следующие свойства.

СВОЙСТВО	ОПИСАНИЕ
method	Вызываемый метод API Telegram Bot.
параметры	Параметры указанного метода.

Поддерживаются следующие методы Telegram:

- answerInlineQuery
- editMessageCaption
- editMessageReplyMarkup
- editMessageText
- forwardMessage
- kickChatMember
- sendAudio
- sendChatAction
- sendContact
- sendDocument
- sendLocation
- sendMessage
- sendPhoto
- sendSticker
- sendVenue
- sendVideo
- sendVoice
- unbanChateMember

Дополнительные сведения об этих методах Telegram и их параметрах см. в документации по [API Telegram Bot](#).

#### NOTE

- Параметр `chat_id` применяется во всех методах Telegram. Если вы не укажете `chat_id` в числе параметров, платформа автоматически создаст идентификатор.
- Чтобы не передавать содержимое файла прямо в запросе, включите ссылку на файл через URL-адрес и тип носителя, как показано в следующем примере.
- В каждом сообщении, которое бот получает из канала Telegram, свойство `channelData` содержит отправленное ранее сообщение.

В этом фрагменте кода показан пример свойства `channelData`, которое определяет один метод Telegram.

```
"channelData": {
    "method": "sendSticker",
    "parameters": {
        "sticker": {
            "url": "https://domain.com/path/gif",
            "mediaType": "image/gif",
        }
    }
}
```

В этом фрагменте кода показан пример свойства `channelData`, которое определяет массив методов

Telegram.

```
"channelData": [
  {
    "method": "sendSticker",
    "parameters": {
      "sticker": {
        "url": "https://domain.com/path/gif",
        "mediaType": "image/gif",
      }
    }
  },
  {
    "method": "sendMessage",
    "parameters": {
      "text": "<b>This message is HTML formatted.</b>",
      "parse_mode": "HTML"
    }
  }
]
```

## Создание собственного сообщения Kik

Чтобы создать собственное сообщение Kik, присвойте свойству `channelData` объекта `Activity` объект JSON, содержащий следующие свойства.

СВОЙСТВО	ОПИСАНИЕ
отправляемых из облака на устройство	Массив сообщений Kik. См. дополнительные сведения о <a href="#">формате сообщений Kik</a> .

В этом фрагменте кода демонстрируется свойство `channelData` для собственного сообщения Kik.

```
"channelData": {
  "messages": [
    {
      "chatId": "c6dd8165...",
      "type": "link",
      "to": "kikhandle",
      "title": "My Webpage",
      "text": "Some text to display",
      "url": "http://botframework.com",
      "picUrl": "http://lorempixel.com/400/200/",
      "attribution": {
        "name": "My App",
        "iconUrl": "http://lorempixel.com/50/50/"
      },
      "noForward": true,
      "kikJsData": {
        "key": "value"
      }
    }
  ]
}
```

## Дополнительные ресурсы

- [Создание сообщений](#)
- [Отправка и получение сообщений](#)
- [Принципы использования действий в Bot Framework](#)

- Справочник по каналам

# Управление данными о состоянии с помощью REST API

27.03.2021 • 2 minutes to read • [Edit Online](#)

Боты обычно используют хранилище для отслеживания того, в каком месте диалога находится пользователь, а также для сбора сведений о взаимоотношениях пользователя с ботом. Пакет SDK для Bot Framework управляет состоянием пользователя и диалога автоматически для разработчиков ботов.

Изначально в Bot Framework включалась служба состояний для хранения этих данных. Но большинство современных ботов (и все последние выпуски пакета SDK для Bot Framework) используют хранилище, управляемое непосредственно разработчиком бота, а не централизованно управляемую службу.

Поддержка центральной службы состояний Bot Framework прекращена 30 марта 2018 г. См. сведения о том, что [вам нужно знать в связи с прекращением поддержки службы состояний Bot Framework](#).

# Основные понятия Direct Line API 3.0

27.03.2021 • 3 minutes to read • [Edit Online](#)

С помощью API для Direct Line вы можете реализовать обмен данными между ботом и своим клиентским приложением. В этой статье представлены основные понятия Direct Line API 3.0 и сведения о соответствующих ресурсах для разработчиков. Вы можете создать клиент, используя пакет SDK, REST API или Web Chat.

## Аутентификация

Аутентификация запросов Direct Line API 3.0 может осуществляться с использованием **секрета**, который можно получить на странице конфигурации канала Direct Line на [портале Azure](#), или с помощью **токена**, который можно получить в среде выполнения. Дополнительные сведения см. в разделе [Authenticate to the Speech API](#) (Аутентификация в API речи).

## Начало общения

Общения Direct Line открываются клиентами явным образом и могут выполняться, пока бот и клиент участвуют в них и имеют действительные учетные данные. Дополнительные сведения см. в статье [Начало общения](#).

## Отправка сообщений

С помощью Direct Line API 3.0 клиент может отправлять боту сообщения, выполняя запросы `HTTP POST`. В каждом запросе клиент может отправить одно сообщение. Дополнительные сведения см. в руководстве по [отправке действия боту](#).

## Получение сообщений

Клиент может получать сообщения от бота с помощью Direct Line API 3.0 через поток `WebSocket` либо путем отправки запросов `HTTP GET`. Используя любой из этих методов, клиент может получать сразу несколько сообщений от бота как часть `ActivitySet`. Дополнительные сведения см. в статье [Receive activities from the bot](#) (Получение действий от бота).

## Ресурсы для разработчиков

### Клиентские библиотеки

Bot Framework предоставляет клиентские библиотеки, которые позволяют легко получить доступ к Direct Line API 3.0 с помощью C# и Node.js.

- Чтобы использовать клиентскую библиотеку .NET в проекте Visual Studio, установите `Microsoft.Bot.Connector.DirectLine` [пакет NuGet](#).
- Чтобы использовать клиентскую библиотеку в проекте Node.js, установите библиотеку `botframework-directlinejs` с помощью [NPM](#) (или [загрузите](#) источник).

### Элемент управления веб-чата

Bot Framework предоставляет элемент управления, который позволяет внедрить бот, использующий Direct Line, в клиентское приложение. Дополнительные сведения см. в статье о [Microsoft Bot Framework WebChat control](#) (Элемент управления WebChat в Microsoft Bot Framework).

# Проверка подлинности в API прямой линии 3,0

27.03.2021 • 16 minutes to read • [Edit Online](#)

Клиент может проверить подлинность запросов к API Direct Line версии 3.0 с помощью **секрета**, который [можно получить на странице конфигурации канала Direct Line](#) на портале Bot Framework, или с помощью **маркера**, который можно получить во время выполнения. Секрет или маркер необходимо указать в заголовке `Authorization` каждого запроса, используя следующий формат:

```
Authorization: Bearer SECRET_OR_TOKEN
```

## Секреты и маркеры

**Секрет** Direct Line — это главный ключ, который можно использовать для доступа ко всем диалогам, связанным с соответствующим ботом. **Секрет** также можно использовать для получения **маркера**. Срок действия секрета не ограничен.

**Маркер** Direct Line — это ключ, который можно использовать для доступа к одному диалогу. Срок действия маркера ограничен, но его можно продлить.

При принятии решения о том, следует ли использовать **секретный ключ** или **токен** и когда это нужно делать, нужно учитывать вопросы безопасности. Секретный ключ должен предоставляться с какой-то целью (намеренно) и с осторожностью. Собственно, это поведение по умолчанию, благодаря чему Direct Line может определять, является ли клиент допустимым. Но в целом, если вы пытаетесь сохранить данные пользователя, обеспечение безопасности является критически важным. См. сведения о [вопросах безопасности](#).

При создании приложения между службами проще всего указать **секрет** в заголовке `Authorization` запросов API Direct Line. Если вы создаете приложение, в котором клиент запускается в веб-браузере или в мобильном приложении, вы можете обменять секрет на маркер (который будет действительным только для одного диалога и перестанет действовать, если не будет продлен) и указать **маркер** в заголовке `Authorization` запросов API Direct Line. Выберите наиболее подходящую вам модель безопасности.

### NOTE

Учетные данные клиента Direct Line отличаются от учетных данных вашего бота. Это позволяет проверить ключи независимо друг от друга и предоставить маркеры клиента, не раскрывая пароль бота.

## Получение секрета Direct Line

Вы можете [получить секрет Direct Line](#) на странице настройки канала Direct Line для вашего бота на портале Azure:

## Configure Direct Line



+ Add new site	KB_Client_1	<input type="checkbox"/> Disable
KB_Client_1	<b>Secret keys</b> Hide   Regenerate xxxxxxxxxxxxxxxxxxxxxxxxxxxx Show   Regenerate	
	<b>Version</b> Select which versions of the Direct Line protocol are enabled on this site. More information about these versions can be found in the <a href="#">Direct Line reference documentation</a> .	

## Создание маркера Direct Line

Чтобы создать маркер Direct Line, который можно использовать для доступа к одному диалогу, необходимо сначала получить секрет Direct Line на странице настройки канала Direct Line на [портале Azure](#). Затем выполните следующий запрос, чтобы обменять секрет Direct Line на маркер Direct Line:

```
POST https://directline.botframework.com/v3/directline/tokens/generate
Authorization: Bearer SECRET
```

В заголовке `Authorization` этого запроса замените `SECRET` значением секрета Direct Line.

Ниже приведены примеры фрагментов кода для запроса на создание маркера и соответствующего ответа.

### Запрос

```
POST https://directline.botframework.com/v3/directline/tokens/generate
Authorization: Bearer RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qbOF5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0
```

Полезные данные запроса, куда входят параметры маркера, необязательны, но рекомендуются. При создании маркера, который можно отправить обратно в службу Direct Line, предоставьте следующие полезные данные для более безопасного подключения. Благодаря этим значениям Direct Line сможет выполнять дополнительную проверку идентификатора и имени пользователя, препятствуя изменению этих значений злоумышленниками. Также эти значения упрощают для Direct Line отправку действия *обновления диалога*, что позволяет создать обновление беседы немедленно при присоединении пользователя. Если эта информация не предоставляется, Direct Line сможет обновить беседу только после того, как пользователь отправит в нее какое-либо содержимое.

```
{
  "user": {
    "id": "string",
    "name": "string"
  },
  "trustedOrigins": [
    "string"
  ]
}
```

ПАРАМЕТР	ТИП	ОПИСАНИЕ
<code>user.id</code>	строка	Необязательный параметр. Идентификатор пользователя в конкретном канале, который кодируется в маркере. Для пользователя Direct Line это значение начинается с <code>d1_</code> . Вы можете создать уникальный идентификатор пользователя для каждой беседы, а для повышения безопасности следует исключить возможность угадать этот идентификатор.
<code>user.name</code>	строка	Необязательный параметр. Понятное отображаемое имя пользователя, которое кодируется в маркере.
<code>trustedOrigins</code>	массив строк	Необязательный параметр. Список доверенных доменов для внедрения в маркер. Здесь перечисляются домены, которые могут размещать клиент веб-чата для бота. Этот список должен совпадать с тем, который указан на странице настройки Direct Line для бота.

## Ответ

Если запрос выполнен успешно, ответ содержит `token` для одного диалога, а значение `expires_in` указывает количество секунд до истечения срока действия маркера. Чтобы продолжить использование маркера, необходимо [продлить маркер](#) до того, как его срок действия истечет.

```
HTTP/1.1 200 OK
[other headers]
```

```
{
  "conversationId": "abc123",
  "token": "RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qbOF5xnR2vtCx7CZj0LdjAPGfiCpg4Fv0y8qbOF5xPGfiCpg4Fv0y8qqbOF5x8qbOF5xn",
  "expires_in": 1800
}
```

## Сравнение операций создания маркера и начала диалога

Операция создания маркера (`POST /v3/directline/tokens/generate`) аналогична операции [начала диалога](#) (

`POST /v3/directline/conversations`) в том, что обе операции возвращают `token`, который можно использовать для доступа к одному диалогу. Тем не менее, в отличие от операции начала диалога, операция создания маркера не начинает диалог, не связывается с ботом и не создает URL-адрес потоковой передачи WebSocket.

Если вы хотите передать маркер клиентам, а также желаете, чтобы они начали диалог, используйте операцию создания маркера. Если вы хотите начать диалог немедленно, используйте операцию [начала диалога](#).

## Продление маркера Direct Line

Маркер Direct Line можно продлевать неограниченное количество раз, до тех пор, пока срок его действия не истек. Продлить маркер с истекшим сроком действия невозможно. Чтобы продлить маркер Direct Line, выполните следующий запрос:

```
POST https://directline.botframework.com/v3/directline/tokens/refresh
Authorization: Bearer TOKEN_TO_BE_REFRESHED
```

В заголовке `Authorization` этого запроса замените `TOKEN_TO_BE_REFRESHED` на маркер Direct Line, который вы хотите обновить.

Ниже приведены примеры фрагментов кода для запроса на обновление маркера и соответствующего ответа.

### Запрос

```
POST https://directline.botframework.com/v3/directline/tokens/refresh
Authorization: Bearer
CurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0y8qb0F5xPGfiCpg4Fv0y8qqb0F5x8qb0F5xn
```

### Ответ

Если запрос выполнен успешно, ответ содержит новый `token`, который действителен для того же диалога, что и предыдущий маркер, а значение `expires_in` указывает количество секунд до истечения срока нового действия маркера. Чтобы продолжить использование нового маркера, необходимо [продлить маркер](#) до того, как его срок действия истечет.

```
HTTP/1.1 200 OK
[other headers]
```

```
{
  "conversationId": "abc123",
  "token": "RCurR_XV9ZA.cwA.BKA.y8qb0F5xPGfiCpg4Fv0y8qqb0F5x8qb0F5xniaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0",
  "expires_in": 1800
}
```

## Проверка подлинности службы Azure Bot

Сведения, представленные в этом разделе, основаны на инструкциях по [добавлению проверки подлинности к боту с помощью службы Azure Bot](#).

**Функция проверки подлинности службы Azure Bot** позволяет проверять подлинность пользователей и **получать маркеры доступа** от различных поставщиков удостоверений, например *Azure Active Directory*, *GitHub*, *Uber* и т. д. Кроме того, настроить проверку подлинности можно для

пользовательского поставщика удостоверений OAuth2. Все это позволяет использовать только **один фрагмент кода проверки подлинности**, который будет работать для всех поддерживаемых поставщиков удостоверений и всех каналов. Чтобы использовать эти возможности, выполните следующие действия:

1. Статически настройте `settings` в боте, который содержит сведения о регистрации приложения у поставщика удостоверений.
2. Используйте `oauthcard`, указав сведения о приложении, которые используются для входа пользователя и были указаны на предыдущем шаге.
3. Получите маркеры доступа с помощью API **службы Azure Bot**.

## Вопросы безопасности

При использовании проверки подлинности службы Azure Bot для [Web Chat](#) важно учитывать несколько моментов, связанных с безопасностью.

1. **Олицетворение.** Здесь олицетворение означает действие злоумышленника, в результате которого бот считает его другим пользователем. В Web Chat злоумышленник может олицетворять кого-то, изменив **идентификатор пользователя** в активном экземпляре канала. Чтобы избежать этого, разработчики должны сделать так, чтобы **идентификатор пользователя нельзя было угадать**.

Если включить **расширенную проверку подлинности**, служба Azure Bot сможет обнаруживать и отклонять любые изменения идентификатора пользователя. Это означает, что идентификатор пользователя (`Activity.From.Id`) в сообщениях, отправляемых боту из Direct Line, всегда будет тем же, с которым вы инициализировали экземпляр Web Chat. Обратите внимание, что для использования этой функции идентификатор пользователя должен начинаться с `d1_`.

### NOTE

Если идентификатор `User.Id` указывается при смене секрета для маркера, этот `User.Id` внедряется в маркер. В DirectLine все отправляемые боту сообщения обязательно содержат этот идентификатор в виде значения `From.Id` действия. Если клиент отправляет сообщение в DirectLine с другим идентификатором `From.Id`, он будет заменен на **идентификатор в маркере** перед перенаправлением сообщения в бот. Поэтому нельзя использовать другой идентификатор пользователя после инициализации секрета канала с этим идентификатором.

2. **Удостоверения пользователей.** Необходимо учитывать, что при работе используется два удостоверения пользователей:
  - a. Удостоверение пользователя в канале.
  - b. Удостоверение пользователя в поставщике удостоверений, который заинтересован в роботе.Когда программа-робот запрашивает у пользователя А в канале вход в поставщик удостоверений Р, процесс входа должен гарантировать, что пользователь А подписывается на Р. Если другому пользователю б разрешено входить в систему, то пользователь А сможет получить доступ к ресурсу пользователя б через робот. Web Chat предоставляет два механизма, обеспечивающих надлежащий процесс входа, как описано далее.
  - a. Раньше по завершении входа пользователю предоставлялся сформированный случайным образом код из шести цифр (магический код). Пользователь должен был ввести этот код в диалоге, который инициировал завершение процесса входа. Как правило, такой механизм неудобно использовать. Кроме того, он обычно подвергается фишинговым атакам. Злоумышленник может обмануть путем вынудить другого пользователя выполнить вход, чтобы получить магический код путем фишинга.

b. Из-за проблем с предыдущим подходом мы решили не использовать в службе Azure Bot магический код. Служба Azure Bot гарантирует, что процесс входа может выполняться только **в том же сеансе браузера**, в котором запущен экземпляр Web Chat. Чтобы включить эту защиту в качестве программы-робота, необходимо начать Web Chat с **прямым маркером**, который содержит **список доверенных доменов, в которых может размещаться клиент Интернет-чата**. Ранее этот токен можно получить, только передав незарегистрированный необязательный параметр в API Direct Line. Но теперь благодаря расширенным параметрам проверки подлинности можно настроить статический список доверенных доменов (источников) на странице настройки Direct Line.

См. сведения о [включении проверки подлинности для бота с помощью службы Azure Bot](#).

### Примеры кода

Следующий контроллер .NET использует включенные расширенные параметры проверки подлинности и возвращает маркер Direct Line и идентификатор пользователя.

```

public class HomeController : Controller
{
    public async Task<ActionResult> Index()
    {
        var secret = GetSecret();

        HttpClient client = new HttpClient();

        HttpRequestMessage request = new HttpRequestMessage(
            HttpMethod.Post,
            $"https://directline.botframework.com/v3/directline/tokens/generate");

        request.Headers.Authorization = new AuthenticationHeaderValue("Bearer", secret);

        var userId = $"dl_{Guid.NewGuid()}";

        request.Content = new StringContent(
            JsonConvert.SerializeObject(
                new { User = new { Id = userId } }),
            Encoding.UTF8,
            "application/json");

        var response = await client.SendAsync(request);
        string token = String.Empty;

        if (response.IsSuccessStatusCode)
        {
            var body = await response.Content.ReadAsStringAsync();
            token = JsonConvert.DeserializeObject<DirectLineToken>(body).token;
        }

        var config = new ChatConfig()
        {
            Token = token,
            UserId = userId
        };

        return View(config);
    }
}

public class DirectLineToken
{
    public string conversationId { get; set; }
    public string token { get; set; }
    public int expires_in { get; set; }
}

public class ChatConfig
{
    public string Token { get; set; }
    public string UserId { get; set; }
}

```

Следующий контроллер JavaScript использует включенные расширенные параметры проверки подлинности и возвращает маркер Direct Line и идентификатор пользователя.

```
var router = express.Router(); // get an instance of the express Router

// Get a directline configuration (accessed at GET /api/config)
const userId = "dl_" + createUniqueId();

router.get('/config', function(req, res) {
  const options = {
    method: 'POST',
    uri: 'https://directline.botframework.com/v3/directline/tokens/generate',
    headers: {
      'Authorization': 'Bearer ' + secret
    },
    json: {
      User: { Id: userId }
    }
  };

  request.post(options, (error, response, body) => {
    if (!error && response.statusCode < 300) {
      res.json({
        token: body.token,
        userId: userId
      });
    } else {
      res.status(500).send('Call to retrieve token from Direct Line failed');
    }
  });
});
```

## Дополнительные ресурсы

- [Основные понятия](#)
- [Подключение бота к Direct Line](#)
- [Добавление аутентификации для бота с помощью службы Azure Bot](#)

# Начать беседу в API прямой линии 3,0

27.03.2021 • 3 minutes to read • [Edit Online](#)

Общения Direct Line открываются клиентами явным образом и могут выполняться, пока бот и клиент участвуют в них и имеют действительные учетные данные. Бот и клиент могут отправлять сообщения, пока общение остается открытым. К определенному диалогу могут подключаться несколько клиентов, и каждый из них может участвовать в нем от имени нескольких пользователей.

## Открытие нового общения

Чтобы открыть новый диалог со стороны клиента, передайте запрос POST в конечную точку /v3/directline/conversations.

```
POST https://directline.botframework.com/v3/directline/conversations
Authorization: Bearer SECRET_OR_TOKEN
```

В следующих фрагментах кода представлен пример запроса и ответа для запуска диалога.

### Запрос

```
POST https://directline.botframework.com/v3/directline/conversations
Authorization: Bearer RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0y8qb0F5xPGfiCpg4Fv0y8qqb0F5x8qb0F5xn
```

### Ответ

Если запрос выполнен, ответ будет содержать идентификатор общения, маркер и значение, указывающее количество секунд до истечения срока действия маркера, и URL-адрес потока, который может быть использован пользователем для [получения действий через потоковую передачу WebSocket](#).

```
HTTP/1.1 201 Created
[other headers]
```

```
{
  "conversationId": "abc123",
  "token": "RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0y8qb0F5xPGfiCpg4Fv0y8qqb0F5x8qb0F5xn",
  "expires_in": 1800,
  "streamUrl": "https://directline.botframework.com/v3/directline/conversations/abc123/stream?token=RCurR_XV9ZA.cwA..."
```

Как правило, запрос на запуск диалога используется для открытия нового диалога, а код состояния HTTP 201 возвращается в случае успешного запуска нового диалога. Однако, если клиент отправляет запрос на запуск диалога с прямым токеном строки в `Authorization` заголовке, который ранее использовался для запуска диалога с помощью операции запуска диалога, возвращается код состояния HTTP 200, указывающий, что запрос приемлем, но диалог не был создан (как уже существовало).

#### TIP

На подключение к URL-адресу потока через протокол WebSocket выделяется 60 секунд. Чтобы создать новый URL-адрес потока, если в течение этого времени не удастся установить подключение, можно [повторно подключиться к общению](#).

## Запуск диалога и создание токена

Операция запуска диалога (`POST /v3/directline/conversations`) аналогична операции [создания токена](#) (`POST /v3/directline/tokens/generate`) в том, что обе операции возвращают объект `token`, который можно использовать для доступа к одному диалогу. Однако операция запуска диалога также запускает диалог, обращается к роботу и создает URL-адрес потока WebSocket, в то время как операция создания токена не выполняет ни одно из этих действий.

Если вы планируете немедленно начать диалог с клиентом, используйте операцию начать беседу. Если вы планируете распространить маркер на клиенты и хотите, чтобы они инициировали диалог, используйте операцию [создания токена](#).

## Дополнительные ресурсы

- [Основные понятия](#)
- [Аутентификация](#)
- [Receive activities via WebSocket stream](#) (Получение действий через поток по протоколу WebSocket)
- [Повторное подключение к общению](#)

# Повторное подключение к беседе в API прямой линии 3,0

21.09.2020 • 2 minutes to read • [Edit Online](#)

Если клиент использует [интерфейс WebSocket](#) для получения сообщений, но теряет подключение, может возникнуть необходимость повторного подключения. В этом сценарии клиент должен создать новый URL-адрес потока по протоколу WebSocket, который можно использовать для повторного подключения к диалогу.

## Создание нового URL-адреса потока по протоколу WebSocket

Чтобы создать новый URL-адрес потока по протоколу WebSocket, который можно использовать для повторного подключения к существующему диалогу, выполните этот запрос:

```
GET https://directline.botframework.com/v3/directline/conversations/{conversationId}?watermark={watermark_value}  
Authorization: Bearer SECRET_OR_TOKEN
```

В этом URI запроса замените `{conversationId}` идентификатором диалога, а `{watermark_value}` — значением водяного знака (если доступен параметр `watermark`). Параметр `watermark` не обязателен. Если в URI запроса указан параметр `watermark`, диалог воспроизводится начиная с водяного знака, за счет чего ни одно сообщение не будет потеряно. Если в URI запроса параметр `watermark` пропущен, будут воспроизведены только те сообщения, которые были получены после выполнения запроса о повторном подключении.

Ниже приведены примеры фрагментов кода для запроса о повторном подключении и ответа на него.

### Запрос

```
GET https://directline.botframework.com/v3/directline/conversations/abc123?watermark=0000a-42  
Authorization: Bearer  
RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0y8qb0F5xPGfiCpg4Fv0y8qqb0F5x8qb0F5xn
```

### Ответ

Если запрос выполнен, ответ будет содержать идентификатор диалога, маркер и новый URL-адрес потока по протоколу WebSocket.

```
HTTP/1.1 200 OK  
[other headers]
```

```
{  
  "conversationId": "abc123",  
  "token":  
    "RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0y8qb0F5xPGfiCpg4Fv0y8qqb0F5x8qb0F5xn",  
  "streamUrl": "https://directline.botframework.com/v3/directline/conversations/abc123/stream?  
watermark=000a-4&t=RCurR_XV9ZA.cwA..."  
}
```

## Повторное подключение к диалогу

Клиент должен использовать новый URL-адрес потока по протоколу WebSocket, чтобы [повторно подключиться к диалогу](#) в течение 60 секунд. Если в течение этого времени не удается установить подключение, клиент должен выполнить новый запрос на повторное подключение, чтобы создать новый URL-адрес потока.

Если параметр Enhanced authentication option (Расширенная проверка подлинности) включен в Direct Line, вы можете получить ошибку с кодом 400 — MissingProperty (свойство не найдено), информирующую о том, что маркер, присоединенный к запросу, не настроен правильно.

## Дополнительные ресурсы

- [Основные понятия](#)
- [Аутентификация](#)
- [Receive activities via WebSocket stream](#) (Получение действий через поток по протоколу WebSocket)

# Отправка действия в программу Bot в API прямой линии 3,0

21.09.2020 • 7 minutes to read • [Edit Online](#)

С помощью протокола Direct Line 3.0 клиенты и боты могут обмениваться различными типами [действий](#), в том числе действиями **сообщений**, **ввода** и настраиваемыми действиями, которые поддерживает бот. В каждом запросе клиент может отправить одно действие.

## Отправка действия

Чтобы отправить действие боту, клиент должен создать объект [Действие](#) для определения действия, а затем выполнить запрос `POST` к

`https://directline.botframework.com/v3/directline/conversations/{conversationId}/activities`, указав объект `Activity` в теле запроса.

Ниже приведены примеры фрагментов кода для запроса отправки действия и соответствующего ответа.

### Запрос

```
POST https://directline.botframework.com/v3/directline/conversations/abc123/activities
Authorization: Bearer RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0
Content-Type: application/json
[other headers]
```

```
{
  "locale": "en-EN",
  "type": "message",
  "from": {
    "id": "user1"
  },
  "text": "hello"
}
```

### Ответ

После того как действие доставлено боту, служба отвечает с кодом состояния HTTP, который отражает код состояния бота. Если в боте возникает ошибка, в ответ на запрос отправки действия клиент получает ответ HTTP 502 ("Недопустимый шлюз").

#### NOTE

Это может быть вызвано тем, что правильный маркер не использовался. Для отправки действия можно использовать только маркер, полученный для операции *Начать беседу*.

Если запрос POST выполнен успешно, ответ содержит полезные данные JSON, указывающие идентификатор действия, которое было отправлено боту.

```
HTTP/1.1 200 OK
[other headers]
```

```
{  
    "id": "0001"  
}
```

## Общее время запроса и ответа отправки действия

Общее время для отправки запроса POST в общение Direct Line является суммой следующих значений:

- транзитное время передачи HTTP-запроса с клиента в службу Direct Line;
- время внутренней обработки в Direct Line (обычно меньше 120 мс);
- транзитное время передачи из службы Direct Line в бот;
- время обработки в боте;
- транзитное время обратной передачи ответа HTTP клиенту.

## Отправка вложений боту

В некоторых случаях клиенту может потребоваться отправить боту вложения, такие как изображения или документы. Отправка вложений осуществляется либо путем [указания URL-адресов](#) вложений в объекте [Действие](#), отправляемом клиентом с помощью

`POST /v3/directline/conversations/{conversationId}/activities`, либо путем [передачи вложений](#) с помощью  
`POST /v3/directline/conversations/{conversationId}/upload`.

## Отправка вложений по URL-адресу

Чтобы отправить одно или несколько вложений в составе объекта [Действие](#) с помощью

`POST /v3/directline/conversations/{conversationId}/activities`, нужно просто включить один или несколько объектов [Вложение](#) в объект Activity и задать свойству `contentUrl` каждого объекта Attachment значения HTTP, HTTPS или URI `data` вложения.

## Отправка вложений путем передачи

Иногда на устройстве клиента могут находиться изображения или документы, которые нужно отправить боту, но URL-адреса, соответствующие этим файлам, отсутствуют. В этом случае клиент может выполнить запрос `POST /v3/directline/conversations/{conversationId}/upload`, чтобы отправить вложения боту путем передачи. Формат и содержимое запроса зависят от того, сколько вложений отправляет клиент, — [одно](#) или [несколько](#).

### Отправка одного вложения путем передачи

Чтобы отправить одно вложение путем передачи, выполните следующий запрос.

```
POST https://directline.botframework.com/v3/directline/conversations/{conversationId}/upload?userId={userId}  
Authorization: Bearer SECRET_OR_TOKEN  
Content-Type: TYPE_OF_ATTACHMENT  
Content-Disposition: ATTACHMENT_INFO  
[other headers]  
  
[file content]
```

В этом URI запроса замените `{conversationId}` на идентификатор общения, а `{userId}` — на идентификатор пользователя, который отправляет сообщение. Параметр `userId` является обязательным. В заголовках запроса задайте для `Content-Type` тип вложения, а для `Content-Disposition` задайте имя файла вложения.

Ниже приведены примеры фрагментов кода для запроса отправки одного вложения и соответствующего

ответа.

## Запрос

```
POST https://directline.botframework.com/v3/directline/conversations/abc123/upload?userId=user1
Authorization: Bearer RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0
Content-Type: image/jpeg
Content-Disposition: name="file"; filename="badjokeeel.jpg"
[other headers]

[JPEG content]
```

## Ответ

Если запрос выполнен успешно, после завершения передачи боту отправляется **сообщение** с действием. А ответ, полученный клиентом, будет содержать идентификатор отправленного действия.

```
HTTP/1.1 200 OK
[other headers]
```

```
{
  "id": "0003"
}
```

## Отправка нескольких вложений путем передачи

Чтобы отправить несколько вложений путем передачи, отправьте составной запрос `POST` к конечной точке `/v3/directline/conversations/{conversationId}/upload`. Задайте `multipart/form-data` в качестве заголовка `Content-Type` запроса и включите заголовок `Content-Type` и заголовок `Content-Disposition` для каждой части, чтобы указать тип и имя файла каждого вложения. В URI запроса задайте параметру `userId` значение идентификатора пользователя, который отправляет сообщение.

В запрос можно включить объект `Activity`, добавив часть, которая указывает значение `application/vnd.microsoft.activity` заголовка `Content-Type`. Если запрос содержит объект `Activity`, то перед отправкой этого объекта в него добавляются вложения, заданные другими частями полезных данных. Если в запросе отсутствует объект `Activity`, создается пустой объект `Activity`, который будет контейнером для сбора отправляемых указанных вложений.

Ниже приведены примеры фрагментов кода для запроса отправки нескольких вложений и соответствующего ответа. В этом примере запрос отправляет сообщение, содержащее текст и одно вложение с изображением. Чтобы включить несколько вложений в это сообщение, в запрос можно добавить дополнительные части.

## Запрос

```
POST https://directline.botframework.com/v3/directline/conversations/abc123/upload?userId=user1
Authorization: Bearer RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0
Content-Type: multipart/form-data; boundary=----DD4E5147-E865-4652-B662-F223701A8A89
[other headers]

-----DD4E5147-E865-4652-B662-F223701A8A89
Content-Type: image/jpeg
Content-Disposition: form-data; name="file"; filename="badjokee1.jpg"
[other headers]

[JPEG content]

-----DD4E5147-E865-4652-B662-F223701A8A89
Content-Type: application/vnd.microsoft.activity
[other headers]

{
    "type": "message",
    "from": {
        "id": "user1"
    },
    "text": "Hey I just IM'd you\n\nand this is crazy\n\nbut here's my webhook\n\nso POST me maybe"
}

-----DD4E5147-E865-4652-B662-F223701A8A89
```

#### Ответ

Если запрос выполнен успешно, после завершения передачи боту отправляется сообщение с действием. А ответ, полученный клиентом, будет содержать идентификатор отправленного действия.

```
HTTP/1.1 200 OK
[other headers]
```

```
{
    "id": "0004"
}
```

## Дополнительные ресурсы

- [Основные понятия](#)
- [Аутентификация](#)
- [Начало общения](#)
- [Повторное подключение к общению](#)
- [Получение действий от бота](#)
- [Конец общения](#)
- [Принципы использования действий в Bot Framework](#)

# Получение действий от программы-робота в API прямой линии 3,0

21.09.2020 • 8 minutes to read • [Edit Online](#)

С помощью протокола Direct Line 3.0 клиенты могут получать действия через поток `WebSocket` или извлекать действия путем выполнения запросов `HTTP GET`.

## WebSocket и запрос GET HTTP

Потоковая передача WebSocket эффективно передает сообщения клиентам, тогда как интерфейс GET позволяет клиентам точно запрашивать сообщения. Несмотря на то что механизм WebSocket часто является предпочтительным благодаря своей эффективности, механизм GET может быть полезен клиентам, которые не могут использовать WebSocket.

Служба позволяет только 1 подключение WebSocket для диалога. Direct Line может закрыть дополнительные подключения WebSocket со значением причины `collision`.

Не все [типы действий](#) доступны как через WebSocket, так и через запрос GET HTTP. В следующей таблице описана доступность различных типов действий для клиентов, использующих протокол Direct Line.

тип действия:	доступность
message	HTTP GET и WebSocket
typing	Только WebSocket
conversationUpdate	Недоступно для отправки и получения через клиент
contactRelationUpdate	Не поддерживается в Direct Line
endOfConversation	HTTP GET и WebSocket
Все другие типы действий	HTTP GET и WebSocket

## Получение действий через потоковую передачу WebSocket

Когда клиент отправляет запрос [Начало общения](#) для открытия общения с помощью бота, ответ службы включает в себя свойство `streamUrl`, которое клиент может впоследствии использовать для подключения через WebSocket. URL-адрес потока предварительно одобрен, поэтому запрос клиента на подключение через WebSocket не требует заголовка `Authorization`.

В следующем примере показан запрос, который использует `streamUrl` для подключения через WebSocket.

```
-- connect to wss://directline.botframework.com --
GET /v3/directline/conversations/abc123/stream?t=RCurR_XV9ZA.cwA...
Upgrade: websocket
Connection: upgrade
[other headers]
```

Служба возвращает код состояния HTTP 101 ("Переключение протоколов").

```
HTTP/1.1 101 Switching Protocols
[other headers]
```

## Получение сообщений

После подключения через WebSocket клиент может получать эти типы сообщений из службы Direct Line.

- Сообщение, содержащее объект [ActivitySet](#), который включает в себя одно или несколько действий и водяной знак (описание приведено ниже).
- Пустое сообщение, которое используется службой Direct Line для подтверждения того, что подключение по-прежнему действительно.
- Дополнительные типы будут определены позже. Эти типы идентифицируются свойствами в корне JSON.

`ActivitySet` содержит сообщения, отправленные ботом и всеми пользователями в общении. В следующем примере показан объект `ActivitySet`, содержащий одно сообщение.

```
{
  "activities": [
    {
      "type": "message",
      "channelId": "directline",
      "conversation": {
        "id": "abc123"
      },
      "id": "abc123|0000",
      "from": {
        "id": "user1"
      },
      "text": "hello"
    }
  ],
  "watermark": "0000a-42"
}
```

## Обработка сообщений

Клиент должен отслеживать значение свойства `watermark`, которое оно получает в каждом `ActivitySet`, чтобы иметь возможность использовать водяной знак для гарантии того, что ни одно сообщение не будет утрачено, если потерян подключение и потребуется [повторно подключиться к общению](#). Если клиент получает `ActivitySet`, где свойство `watermark` равняется `null` или отсутствует, он должен игнорировать это значение и не перезаписывать полученный ранее водяной знак.

Клиент должен игнорировать пустые сообщения, которые получает из службы Direct Line.

Клиент может отправить пустое сообщение в службу Direct Line, чтобы проверить подключение. Служба Direct Line проигнорирует пустые сообщения, получаемые от клиента.

Служба Direct Line может принудительно закрыть подключение к WebSocket при определенных условиях. Если клиент не получил действие `endOfConversation`, он может [создать новый URL-адрес потока WebSocket](#), который можно использовать для повторного подключения к общению.

Поток WebSocket содержит обновления в реальном времени и недавние сообщения (с момента вызова на подключение через WebSocket), но не включает сообщения, отправленные ранее самого последнего `POST` в `/v3/directline/conversations/{id}`. Для получения более ранних сообщений из общения используйте `HTTP GET`, как описано ниже.

# Извлечение действий с помощью запроса GET HTTP

Клиенты, которые не могут использовать WebSocket, могут извлечь действия с помощью запроса

HTTP GET .

Для получения сообщений из конкретного общения отправьте запрос `GET` в конечную точку

`/v3/directline/conversations/{conversationId}/activities`, при необходимости задавая параметр `watermark`, чтобы указать последнее сообщение, отображаемое клиенту.

В следующих фрагментах кода приведен пример запроса и ответа на получение действий общения.

Ответ на получение действий общения содержит `watermark` как свойство `ActivitySet`. Клиенты должны постранично просмотреть доступные действия, переходя по значению `watermark`, пока действия не перестанут возвращаться.

## Запрос

```
GET https://directline.botframework.com/v3/directline/conversations/abc123/activities?watermark=0001a-94
Authorization: Bearer RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0
```

## Ответ

```
HTTP/1.1 200 OK
[other headers]
```

```
{
  "activities": [
    {
      "type": "message",
      "channelId": "directline",
      "conversation": {
        "id": "abc123"
      },
      "id": "abc123|0000",
      "from": {
        "id": "user1"
      },
      "text": "hello"
    },
    {
      "type": "message",
      "channelId": "directline",
      "conversation": {
        "id": "abc123"
      },
      "id": "abc123|0001",
      "from": {
        "id": "bot1"
      },
      "text": "Nice to see you, user1!"
    }
  ],
  "watermark": "0001a-95"
}
```

## Рекомендации относительно расписания

Большинство клиентов хотят сохранить полный журнал сообщений. Несмотря на то что Direct Line — это составной протокол с потенциальными пробелами в расписании, протокол и служба предназначены для упрощения создания надежных клиентов.

- Свойство `watermark`, которое отправляется в поток WebSocket, и ответ на получение действий общения являются надежными. Клиент не пропустит ни одного сообщения, пока воспроизводит водяной знак дословно.
- Когда клиент начинает общение и подключается к потоку WebSocket, все действия, которые отправляются после команды POST, но перед открытием сокета, воспроизводятся перед новыми действиями.
- Когда клиент издает запрос "Получить активности общения" (для обновления истории), в то время как он подключен к потоку WebSocket, действия могут дублироваться по обоим каналам. Клиенты должны отслеживать все известные идентификаторы действий, чтобы иметь возможность отклонить повторяющиеся действия, если они встречаются.

Клиентам, проводящим опрос с помощью `HTTP GET`, следует выбирать интервал опроса, который соответствует их предполагаемому использованию.

- Приложения типа "служба — служба" часто используют интервал опроса, равный 5 или 10 с.
- Приложения, взаимодействующие с клиентами, часто используют интервал опроса, равный 1 с, и выдают один дополнительный запрос вскоре после каждого сообщения, отправленного клиентом (для быстрого получения ответа от бота). Минимальное значение этой задержки составляет 300 мс, но ее настоятельно рекомендуется настроить с учетом скорости бота и транзитного времени. Для любого продолжительного периода времени опрос не следует проводить чаще, чем один раз в секунду.

## Дополнительные ресурсы

- [Основные понятия](#)
- [Аутентификация](#)
- [Начало общения](#)
- [Повторное подключение к общению](#)
- [Отправка действий боту](#)
- [Конец общения](#)

# Завершение диалога в API прямой линии 3,0

27.03.2021 • 2 minutes to read • [Edit Online](#)

**EndOfConversation** — это [действие](#), означающее что канал или бот завершил диалог.

## NOTE

Хотя событие `endOfConversation` отправляется несколькими каналами, канал Cortana — это единственный канал, который его принимает. Другие каналы, включая Direct Line, не реализуют эту функцию, отклоняя или пересылая вместо этого действие — каждый канал определяет, как реагировать на действие `endOfConversation`.

## Отправка действия endOfConversation

Чтобы запросить окончание диалога с помощью канала Кортаны, отправьте запрос POST действия End of Conversation (Завершение беседы) в конечную точку обмена сообщениями канала.

### Запрос

```
POST https://directline.botframework.com/v3/directline/conversations/abc123/activities  
Authorization: Bearer RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0  
[other headers]
```

```
{  
  "type": "endOfConversation",  
  "from": {  
    "id": "user1"  
  }  
}
```

### Ответ

Если запрос выполнен успешно, ответ будет содержать идентификатор отправленного действия.

```
HTTP/1.1 200 OK  
[other headers]
```

```
{  
  "id": "0004"  
}
```

## Дополнительные ресурсы

- [Основные понятия](#)
- [Аутентификация](#)
- [Отправка действий боту](#)

# Справочник по программному интерфейсу Direct Line API 3.0

27.03.2021 • 11 minutes to read • [Edit Online](#)

С помощью API Direct Line 3.0 можно взаимодействовать с ботом в клиентском приложении. Direct Line API 3.0 использует отраслевые стандарты REST и JSON по протоколу HTTPS.

## Базовый универсальный код ресурса

Для доступа к Direct Line API 3.0 используйте этот базовый URI во всех запросах к API:

`https://directline.botframework.com`

## Заголовки

Кроме стандартных заголовков HTTP-запроса запрос Direct Line API должен включать заголовок `Authorization`, определяющий секрет или токен для аутентификации клиента, который выполняет запрос. Используйте для заголовка `Authorization` следующий формат:

`Authorization: Bearer SECRET_OR_TOKEN`

Дополнительные сведения о том, как получить секрет или токен, который клиент может использовать для аутентификации запросов Direct Line API, см. в руководстве по [аутентификации](#).

## Коды состояния HTTP

[Код состояния HTTP](#), который возвращается с каждым ответом, указывает результат соответствующего запроса.

HTTP STATUS CODE (КОД СОСТОЯНИЯ HTTP)	ЗНАЧЕНИЕ
200	Запрос выполнен успешно.
201	Запрос выполнен успешно.
202	Запрос принят для обработки.
204	Запрос успешно выполнен, но содержимое не было возвращено.
400	Запрос неправильный или имеет недопустимый формат.
401	Клиент не авторизован для выполнения запроса. Часто этот код состояния возникает, когда заголовок <code>Authorization</code> отсутствует или имеет недопустимый формат.

HTTP STATUS CODE (КОД СОСТОЯНИЯ HTTP)	ЗНАЧЕНИЕ
403	Клиенту запрещено выполнять запрошенную операцию. Если в запросе указан действительный маркер с истекшим сроком действия, свойство <code>code</code> в объекте <code>Error</code> , который возвращается в <code>ErrorResponse</code> , будет иметь значение <code>TokenExpired</code> .
404	Запрошенный ресурс не найден. Обычно этот код состояния обозначает, что в запросе указан недопустимый URI.
500	Внутренняя ошибка сервера в службе Direct Line
502	Бот недоступен или вернул ошибку. <b>Это обобщенный код ошибки.</b>

#### NOTE

Также в пути подключения WebSocket используется код состояния HTTP 101, но, вероятно, его уже обрабатывает ваш клиент WebSocket.

#### ошибки

Любой ответ, указывающий код состояния HTTP в диапазоне 4xx или 5xx, будет содержать в тексте ответа объект `ErrorResponse`, который предоставляет сведения об ошибке. Если вы получите сообщение об ошибке в диапазоне 4xx, проверьте объект `ErrorResponse`, чтобы определить причину ошибки и устранить проблему, прежде чем повторно отправлять запрос.

#### NOTE

Коды состояния HTTP и значения в свойстве `code` в объекте `ErrorResponse` являются неизменными. Значения, указанные в свойстве `message` в объекте `ErrorResponse`, могут изменяться со временем.

В следующих фрагментах кода представлены пример запроса и ответ на него, содержащий ошибку.

#### Запрос

```
POST https://directline.botframework.com/v3/directline/conversations/abc123/activities
[detail omitted]
```

#### Ответ

```
HTTP/1.1 502 Bad Gateway
[other headers]
```

```
{
  "error": {
    "code": "BotRejectedActivity",
    "message": "Failed to send activity: bot returned an error"
  }
}
```

## Операции с токенами

Используйте эти операции для создания или обновления токена, который клиент может использовать для доступа к одному диалогу.

ОПЕРАЦИЯ	ОПИСАНИЕ
Создание токена	Создает токен для нового диалога.
Обновление токена	Обновляет токен.

### Generate Token

Создает токен, действующий для одного диалога.

```
POST /v3/directline/tokens/generate
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
Текст запроса	Объект <a href="#">TokenParameters</a>
Возвращает	Объект <a href="#">Conversation</a> .

### Refresh Token

Обновляет токен.

```
POST /v3/directline/tokens/refresh
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
Текст запроса	Недоступно
Возвращает	Объект <a href="#">Conversation</a> .

## Операции диалога

Используйте эти операции, чтобы открыть диалог с ботом и обменяться сообщениями между клиентом и ботом.

ОПЕРАЦИЯ	ОПИСАНИЕ
Начало диалога	Открывает новый диалог с ботом.
Получение сведений о диалоге	Получает сведения о существующем диалоге. Эта операция создает URL-адрес потока WebSocket, который позволяет клиенту <a href="#">повторно подключиться</a> к диалогу.
Получение действий	Получает действия от бота.
Отправка действия	Отправляет действие боту.
Загрузка и отправка файлов	Загружает и отправляет файлы в виде вложений.

## Начало диалога

Открывает новый диалог с ботом.

```
POST /v3/directline/conversations
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
<b>Текст запроса</b>	Объект <a href="#">TokenParameters</a>
<b>Возвращает</b>	Объект <a href="#">Conversation</a> .

## Получение сведений о диалоге

Получает информацию о существующем диалоге и создает URL-адрес потока WebSocket, который позволяет клиенту [повторно подключиться](#) к диалогу. Вы можете включить необязательный параметр `watermark` в URI запроса, чтобы указать последнее сообщение, увиденное клиентом.

```
GET /v3/directline/conversations/{conversationId}?watermark={watermark_value}
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
<b>Текст запроса</b>	Недоступно
<b>Возвращает</b>	Объект <a href="#">Conversation</a> .

## Получение действий

Получает действия от бота для указанного диалога. Вы можете включить необязательный параметр `watermark` в URI запроса, чтобы указать последнее сообщение, увиденное клиентом.

```
GET /v3/directline/conversations/{conversationId}/activities?watermark={watermark_value}
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
<b>Текст запроса</b>	Недоступно
<b>Возвращает</b>	Объект <a href="#">ActivitySet</a> . Ответ содержит <code>watermark</code> как свойство объекта <a href="#">ActivitySet</a> . Клиенты должны постранично просмотреть доступные действия, переходя по значению <code>watermark</code> , пока действия не перестанут возвращаться.

## Отправка действия

Отправляет действие боту.

```
POST /v3/directline/conversations/{conversationId}/activities
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
<b>Текст запроса</b>	Объект <a href="#">Activity</a>

СОДЕРЖИМОЕ	ОПИСАНИЕ
<b>Возвращает</b>	Объект <a href="#">ResourceResponse</a> , который содержит свойство <code>id</code> с идентификатором действия, отправленного боту.

### Загрузка и отправка файлов

Загружает и отправляет файлы в виде вложений. Задайте параметр `userId` в URI запроса, чтобы указать идентификатор пользователя, отправляющего вложения.

```
POST /v3/directline/conversations/{conversationId}/upload?userId={userId}
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
<b>Текст запроса</b>	Для одного вложения заполните текст запроса содержимым файла. Для нескольких вложений создайте текст запроса из нескольких частей, по одной для каждого вложения, а также (необязательно) одну часть для объекта <a href="#">Действие</a> , которая будет служить контейнером для всех вложений. Дополнительные сведения см. в руководстве по <a href="#">отправке действия боту</a> .
<b>Возвращает</b>	Объект <a href="#">ResourceResponse</a> , который содержит свойство <code>id</code> с идентификатором действия, отправленного боту.

#### NOTE

Загруженные файлы удаляются через 24 часа.

## СХЕМА

Схема Direct Line 3.0 включает все объекты, которые определены в [схеме Bot Framework](#), а также объекты, которые связаны с Direct Line.

### Объект ActivitySet

Определяет набор действий.

СВОЙСТВО	ТИП	ОПИСАНИЕ
<b>действия</b>	<a href="#">Действие[]</a>	Массив объектов <a href="#">Activity</a> .
<code>watermark</code>	строка	Максимальное значение для водяного знака действий в наборе. Клиент может использовать значение <code>watermark</code> , чтобы указать последнее полученное имя сообщение, при <a href="#">получении новых действий от бота</a> или при <a href="#">создании нового URL-адреса потока WebSocket</a> .

### Объект Conversation

Определяет диалог Direct Line.

СВОЙСТВО	ТИП	ОПИСАНИЕ
conversationId	строка	Идентификатор, который уникально идентифицирует диалог, для которого действует указанный токен.
eTag	строка	ETag HTTP (тег сущности).
expires_in	number	Число секунд до истечения срока действия токена.
referenceGrammarId	строка	Идентификатор справочной грамматики для этого бота.
streamUrl	строка	URL-адрес для потока сообщений диалога.
token	строка	Токен, действующий для указанного диалога.

## Объект TokenParameters

Параметры для создания токена.

СВОЙСТВО	ТИП	ОПИСАНИЕ
eTag	строка	ETag HTTP (тег сущности).
trustedOrigins	string[]	Доверенные источники для внедрения в токен.
user	ChannelAccount	Учетная запись пользователя для внедрения в маркер.

## Действия

Для каждого объекта [Действие](#), полученного клиентом от бота через Direct Line, выполняется следующее:

- сохраняются карточки, включенные как вложения;
- URL-адреса отправленных вложений подменяются частной ссылкой;
- свойство `channelData` сохраняется без изменений.

Клиенты могут [получать](#) несколько действий от бота как набор действий ([ActivitySet](#)).

Когда клиент отправляет боту объект `Activity` через Direct Line, происходит следующее:

- свойство `type` принимает тип отправленного действия (обычно это `message`);
- свойство `from` заполняется идентификатором пользователя, выбранным клиентом;
- вложения могут содержать URL-адреса, ведущие к существующим ресурсам или ресурсам, отправленным через конечную точку вложений Direct Line;
- свойство `channelData` сохраняется без изменений.
- Общий размер действия, сериализованного и зашифрованного в формате JSON, не должен превышать 256 тысяч символов. Поэтому рекомендуется, чтобы количество действий не превышало 150 тысяч.

Если требуется больше данных, разбейте действие на несколько составных частей или рассмотрите возможность использования вложений.

В каждом запросе клиент может [отправить](#) только одно действие.

## Дополнительные ресурсы

- [Спецификация по действиям Bot Framework](#)

# Основные понятия API 1.1 для Direct Line

03.09.2020 • 2 minutes to read • [Edit Online](#)

С помощью API для Direct Line вы можете реализовать обмен данными между ботом и своим клиентским приложением.

## IMPORTANT

В этой статье раскрыты основные понятия API 1.1 для Direct Line и приведены сведения о соответствующих ресурсах для разработчиков. При создании подключения между клиентским приложением и ботом используйте [API версии 3.0 для Direct Line](#).

## Аутентификация

Аутентификация запросов Direct Line API 1.1 может осуществляться с использованием **секрета**, который можно получить на странице конфигурации канала Direct Line на [портале Azure](#), или с использованием **токена**, который можно получить в среде выполнения. Дополнительные сведения см. в разделе [Authenticate to the Speech API](#) (Аутентификация в API речи).

## Начало общения

Общения Direct Line открываются клиентами явным образом и могут выполняться, пока бот и клиент участвуют в них и имеют действительные учетные данные. Дополнительные сведения см. в статье [Начало общения](#).

## Отправка сообщений

С помощью API 1.1 для Direct Line клиент может отправлять боту сообщения, выполняя запросы `HTTP POST`. В каждом запросе клиент может отправить одно сообщение. Дополнительные сведения см. в разделе [Отправка сообщения боту](#).

## Получение сообщений

С помощью API 1.1 для Direct Line клиент может получать сообщения, выполняя опросы с запросами `HTTP GET`. В ответе на каждый запрос клиент может получить от бота несколько сообщений в виде части объекта `MessageSet`. Дополнительные сведения см. в статье [Receive messages from the bot](#) (Получение сообщений от бота).

## Ресурсы для разработчиков

### Клиентская библиотека

Bot Framework предоставляет клиентскую библиотеку, которая позволяет легко получить доступ к API для 1.1 Direct Line с помощью C#. Чтобы использовать клиентскую библиотеку в проекте Visual Studio, установите `Microsoft.Bot.Connector.DirectLine` пакет NuGet версии 1.x.

Вместо клиентской библиотеки C# вы можете создать собственную клиентскую библиотеку на любом языке, используя [файл Swagger для API 1.1 для Direct Line](#).

# Проверка подлинности в API прямой линии 1.1

21.09.2020 • 6 minutes to read • [Edit Online](#)

## IMPORTANT

В этой статье описана аутентификация в API 1.1 для Direct Line. При создании подключения между клиентским приложением и ботом используйте [API версии 3.0 для Direct Line](#).

Клиент может выполнить аутентификацию запросов к API Direct Line версии 1.1 с помощью **секрета**, который [доступен на странице конфигурации канала Direct Line](#) на портале Bot Framework, или с помощью **маркера**, который можно получить во время выполнения.

Секрет или маркер нужно указать в заголовке `Authorization` каждого запроса с помощью схемы Bearer или BotConnector.

## Схема Bearer

```
Authorization: Bearer SECRET_OR_TOKEN
```

## Схема BotConnector

```
Authorization: BotConnector SECRET_OR_TOKEN
```

## Секреты и маркеры

**Секрет** Direct Line — это главный ключ, который можно использовать для доступа ко всем диалогам, связанным с соответствующим ботом. **Секрет** также можно использовать для получения **маркера**. Срок действия секрета не ограничен.

**Маркер** Direct Line — это ключ, который можно использовать для доступа к одному диалогу. Срок действия маркера ограничен, но его можно продлить.

При создании приложения между службами проще всего указать **секрет** в заголовке `Authorization` запросов API Direct Line. Если вы создаете приложение, в котором клиент запускается в веб-браузере или в мобильном приложении, вы можете обменять секрет на маркер (который будет действительным только для одного диалога и перестанет действовать, если не будет продлен) и указать **маркер** в заголовке `Authorization` запросов API Direct Line. Выберите наиболее подходящую вам модель безопасности.

## NOTE

Учетные данные клиента Direct Line отличаются от учетных данных вашего бота. Это позволяет проверить ключи независимо друг от друга и предоставить маркеры клиента, не раскрывая пароль бота.

## Получение секрета Direct Line

Вы можете [получить секрет Direct Line](#) на странице настройки канала Direct Line для вашего бота на [портале Azure](#):

## Configure Direct Line



+ Add new site KB\_Client\_1  Disable |

KB\_Client\_1

Secret keys

Hide | Regenerate

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx Show | Regenerate

Version

Select which versions of the Direct Line protocol are enabled on this site. More information about these versions can be found in the [Direct Line reference documentation](#).

## Создание маркера Direct Line

Чтобы создать маркер Direct Line, который можно использовать для доступа к одному диалогу, необходимо сначала получить секрет Direct Line на странице настройки канала Direct Line на [портале Azure](#). Затем выполните следующий запрос, чтобы обменять секрет Direct Line на маркер Direct Line:

```
POST https://directline.botframework.com/api/tokens/conversation  
Authorization: Bearer SECRET
```

В заголовке `Authorization` этого запроса замените `SECRET` значением секрета Direct Line.

Ниже приведены примеры фрагментов кода для запроса на создание маркера и соответствующего ответа.

### Запрос

```
POST https://directline.botframework.com/api/tokens/conversation  
Authorization: Bearer RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0
```

### Ответ

Если запрос выполнен успешно, ответ будет содержать маркер, который действителен для одного диалога. Срок действия маркера истечет через 30 минут. Чтобы продолжить использование маркера, необходимо [продлить маркер](#) до того, как его срок действия истечет.

```
HTTP/1.1 200 OK  
[other headers]  
"RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0y8qb0F5xPGfiCpg4Fv0y8qqb0F5x8qb0F5xn"
```

### Сравнение операций создания маркера и начала диалога

Операция создания маркера (`POST /api/tokens/conversation`) аналогична операции [начала диалога](#) (`POST /api/conversations`) в том, что обе операции возвращают `token`, который можно использовать для доступа к одному диалогу. Тем не менее, в отличие от операции начала диалога, операция создания маркера не начинает диалог и не связывается с ботом.

Если вы хотите передать маркер клиентам, а также желаете, чтобы они начали диалог, используйте операцию создания маркера. Если вы хотите начать диалог немедленно, используйте операцию [начала диалога](#).

## Продление маркера Direct Line

Маркер Direct Line действует в течение 30 минут от момента создания. Его можно продлевать неограниченное количество раз, до тех пор, пока срок его действия не истек. Продлить маркер с истекшим сроком действия невозможно. Чтобы продлить маркер Direct Line, выполните следующий запрос:

```
POST https://directline.botframework.com/api/tokens/{conversationId}/renew
Authorization: Bearer TOKEN_TO_BE_REFRESHED
```

В URI запроса замените `{conversationId}` идентификатором диалога, для которого действует маркер, а в заголовке `Authorization` этого запроса, замените `TOKEN_TO_BE_REFRESHED` маркером Direct Line, который нужно продлить.

Ниже приведены примеры фрагментов кода для запроса на обновление маркера и соответствующего ответа.

### Запрос

```
POST https://directline.botframework.com/api/tokens/abc123/renew
Authorization: Bearer
CurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0y8qb0F5xPGfiCpg4Fv0y8qqb0F5x8qb0F5xn
```

### Ответ

Если запрос выполнен успешно, ответ содержит новый маркер, который действителен для того же диалога, что и предыдущий маркер. Срок действия нового маркера истечет через 30 минут. Чтобы продолжить использование нового маркера, необходимо [продлить маркер](#) до того, как его срок действия истечет.

```
HTTP/1.1 200 OK
[other headers]

"RCurR_XV9ZA.cwA.BKA.y8qb0F5xPGfiCpg4Fv0y8qqb0F5x8qb0F5xniaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0"
```

## Дополнительные ресурсы

- [Основные понятия](#)
- [Подключение бота к Direct Line](#)

# Начать беседу в API прямой линии 1.1

27.03.2021 • 2 minutes to read • [Edit Online](#)

## IMPORTANT

В этой статье объясняется, как начать диалог с помощью API Direct Line версии 1.1. При создании подключения между клиентским приложением и ботом используйте [API версии 3.0 для Direct Line](#).

Общения Direct Line открываются клиентами явным образом и могут выполняться, пока бот и клиент участвуют в них и имеют действительные учетные данные. Бот и клиент могут отправлять сообщения, пока общение остается открытым. К определенному диалогу могут подключаться несколько клиентов, и каждый из них может участвовать в нем от имени нескольких пользователей.

## Открытие нового общения

Чтобы с помощью бота открыть новое общение, необходимо выполнить следующий запрос.

```
POST https://directline.botframework.com/api/conversations
Authorization: Bearer SECRET_OR_TOKEN
```

Ниже приведены примеры фрагментов кода, предназначенные для запроса "начало общения" и соответствующего ответа.

### Запрос

```
POST https://directline.botframework.com/api/conversations
Authorization: Bearer
RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0y8qb0F5xPGfiCpg4Fv0y8qqb0F5x8qb0F5xn
```

### Ответ

Если запрос выполнен, ответ будет содержать идентификатор для диалога, маркер и значение, указывающее количество секунд до истечения срока действия маркера.

```
HTTP/1.1 200 OK
[other headers]
```

```
{
  "conversationId": "abc123",
  "token": "RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0y8qb0F5xPGfiCpg4Fv0y8qqb0F5x8qb0F5xn",
  "expires_in": 1800
}
```

## Сравнение операций "Начало общения" и "Создать маркер"

Операция начала диалога (`POST /api/conversations`) аналогична операции [создания маркера](#) (`POST /api/tokens/conversation`) в том смысле, что обе операции возвращают маркер `token`, который можно использовать для доступа к одному диалогу. Однако операция начала диалога также начинает

диалог и контактирует с ботом, тогда как операция создания маркера не поддерживает ни одно из этих действий.

Если требуется немедленно начать общение, используйте операцию "Начало общения". Если требуется распространить маркер среди клиентов, а также заставить их начать общение, используйте операцию [создание маркера Direct Line](#).

## Дополнительные ресурсы

- [Основные понятия](#)
- [Аутентификация](#)

# Отправка сообщения в Bot в API прямой линии

## 1.1

21.09.2020 • 7 minutes to read • [Edit Online](#)

### IMPORTANT

В этой статье описывается отправка сообщения боту с помощью Direct Line API 1.1. При создании подключения между клиентским приложением и ботом используйте [API версии 3.0 для Direct Line](#).

С помощью протокола Direct Line 1.1 клиенты могут обмениваться сообщениями с ботами. Эти сообщения преобразовываются в схему, поддерживаемую ботом (Bot Framework версии 1 или 3). В каждом запросе клиент может отправить одно сообщение.

## Отправка сообщения

Для отправки сообщения боту клиент должен создать объект [Message](#) для определения сообщения, а затем отправить запрос `POST` на адрес

`https://directline.botframework.com/api/conversations/{conversationId}/messages`, указав объект [Message](#) в теле запроса.

Ниже приведены примеры фрагментов кода для запроса отправки сообщения и соответствующего ответа.

### Запрос

```
POST https://directline.botframework.com/api/conversations/abc123/messages
Authorization: Bearer RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qbOF5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0
[other headers]
```

```
{
  "text": "hello",
  "from": "user1"
}
```

### Ответ

После доставки сообщения боту служба отвечает кодом состояния HTTP, который отражает код состояния бота. Если бот выдает ошибку, клиенту в ответ на запрос отправки сообщения возвращается ответ HTTP 500 ("Внутренняя ошибка сервера"). Если вызов POST завершен успешно, служба возвращает код состояния HTTP 204. В тексте ответа данные не возвращаются. Сообщение клиента и любые сообщения от бота можно получить через [опрос](#).

```
HTTP/1.1 204 No Content
[other headers]
```

### Общее время отправки сообщения (отправка запроса и получение ответа)

Общее время для отправки запроса POST в общение Direct Line является суммой следующих значений:

- транзитное время передачи HTTP-запроса с клиента в службу Direct Line;

- время внутренней обработки в Direct Line (обычно меньше 120 мс);
- транзитное время передачи из службы Direct Line в бот;
- время обработки в боте;
- транзитное время обратной передачи ответа HTTP клиенту.

## Отправка вложений боту

В некоторых случаях клиенту может потребоваться отправить боту вложения, такие как изображения или документы. Отправка вложений осуществляется либо путем [указания URL-адресов](#) вложений в объекте `Message`, отправляемом клиентом с помощью `POST /api/conversations/{conversationId}/messages`, либо путем [передачи вложений](#) с помощью `POST /api/conversations/{conversationId}/upload`.

## Отправка вложений по URL-адресу

Для отправки одного или нескольких вложений как часть объекта `Message` с помощью `POST /api/conversations/{conversationId}/messages`, укажите URL-адреса вложений в массиве `images` и/или массиве `attachments` сообщения.

## Отправка вложений путем передачи

Иногда на устройстве клиента могут находиться изображения или документы, которые нужно отправить боту, но URL-адреса, соответствующие этим файлам, отсутствуют. В этом случае клиент может выполнить запрос `POST /api/conversations/{conversationId}/upload`, чтобы отправить вложения боту путем передачи. Формат и содержимое запроса зависят от того, сколько вложений отправляет клиент, — [одно](#) или [несколько](#).

### Отправка одного вложения путем передачи

Чтобы отправить одно вложение путем передачи, выполните следующий запрос.

```
POST https://directline.botframework.com/api/conversations/{conversationId}/upload?userId={userId}
Authorization: Bearer SECRET_OR_TOKEN
Content-Type: TYPE_OF_ATTACHMENT
Content-Disposition: ATTACHMENT_INFO
[other headers]

[file content]
```

В этом URI запроса замените `{conversationId}` на идентификатор общения, а `{userId}` — на идентификатор пользователя, который отправляет сообщение. В заголовках запроса задайте для `Content-Type` тип вложения, а для `Content-Disposition` задайте имя файла вложения.

Ниже приведены примеры фрагментов кода для запроса отправки одного вложения и соответствующего ответа.

### Запрос

```
POST https://directline.botframework.com/api/conversations/abc123/upload?userId=user1
Authorization: Bearer RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0
Content-Type: image/jpeg
Content-Disposition: name="file"; filename="badjokeeel.jpg"
[other headers]

[JPEG content]
```

### Ответ

Если запрос выполнен успешно, после завершения передачи сообщение отправляется боту, и служба

возвращает код состояния HTTP 204.

```
HTTP/1.1 204 No Content
[other headers]
```

## Отправка нескольких вложений путем передачи

Чтобы отправить несколько вложений путем передачи, отправьте составной запрос `POST` к конечной точке `/api/conversations/{conversationId}/upload`. Задайте `multipart/form-data` в качестве заголовка `Content-Type` запроса и включите заголовок `Content-Type` и заголовок `Content-Disposition` для каждой части, чтобы указать тип и имя файла каждого вложения. В URI запроса задайте параметру `userId` значение идентификатора пользователя, который отправляет сообщение.

В запрос можно включить объект `Message`, добавив часть, которая указывает значение `application/vnd.microsoft.bot.message` заголовка `Content-Type`. Это позволяет клиенту настроить сообщение, содержащее вложения. Если запрос содержит объект `Message`, то перед отправкой этого объекта в него добавляются вложения, заданные другими частями полезных данных.

Ниже приведены примеры фрагментов кода для запроса отправки нескольких вложений и соответствующего ответа. В этом примере запрос отправляет сообщение, содержащее текст и одно вложение с изображением. Чтобы включить несколько вложений в это сообщение, в запрос можно добавить дополнительные части.

### Запрос

```
POST https://directline.botframework.com/api/conversations/abc123/upload?userId=user1
Authorization: Bearer RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0
Content-Type: multipart/form-data; boundary=----DD4E5147-E865-4652-B662-F223701A8A89
[other headers]

----DD4E5147-E865-4652-B662-F223701A8A89
Content-Type: image/jpeg
Content-Disposition: form-data; name="file"; filename="badjokeeel.jpg"
[other headers]

[JPEG content]

----DD4E5147-E865-4652-B662-F223701A8A89
Content-Type: application/vnd.microsoft.bot.message
[other headers]

{
  "text": "Hey I just IM'd you\n\nand this is crazy\n\nbut here's my webhook\n\nso POST me maybe",
  "from": "user1"
}

----DD4E5147-E865-4652-B662-F223701A8A89
```

### Ответ

Если запрос выполнен успешно, после завершения передачи сообщение отправляется боту, и служба возвращает код состояния HTTP 204.

```
HTTP/1.1 204 No Content
[other headers]
```

## Дополнительные ресурсы

- [Основные понятия](#)

- Аутентификация
- Начало общения
- Получение сообщений от бота

# Получение сообщений с помощью программы-робота в API прямой линии 1.1

21.09.2020 • 2 minutes to read • [Edit Online](#)

## IMPORTANT

В этой статье описывается получения сообщения ботом с помощью Direct Line API 1.1. При создании подключения между клиентским приложением и ботом используйте [API версии 3.0 для Direct Line](#).

Чтобы использовать протокол Direct Line 1.1 для получения сообщений, клиенты должны взаимодействовать через интерфейс `HTTP GET`.

## Получение сообщения с помощью HTTP-запроса GET

Для получения сообщений из конкретного общения отправьте запрос `GET` в конечную точку `api/conversations/{conversationId}/messages`, при необходимости задавая параметр `watermark`, чтобы указать последнее сообщение, отображаемое клиенту. Даже если в ответе JSON не было включено сообщение, он будет возвращен с обновленным значением свойства `watermark`.

Ниже приведены примеры фрагментов кода для запроса и ответа "Get Messages". Ответ Get Messages содержит свойство `watermark` В Качестве `MessageSet`. Клиенты должны просматривать доступные сообщения, перемещая значение `watermark` до тех пор, пока сообщения не перестанут возвращаться.

### Запрос

```
GET https://directline.botframework.com/api/conversations/abc123/messages?watermark=0001a-94
Authorization: Bearer RCurR_XV9ZA.cwA.BKA.iaJrC8xpy8qb0F5xnR2vtCX7CZj0LdjAPGfiCpg4Fv0
```

### Ответ

```
HTTP/1.1 200 OK
[other headers]
```

```
{
  "messages": [
    {
      "conversation": "abc123",
      "id": "abc123|0000",
      "text": "hello",
      "from": "user1"
    },
    {
      "conversation": "abc123",
      "id": "abc123|0001",
      "text": "Nice to see you, user1!",
      "from": "bot1"
    }
  ],
  "watermark": "0001a-95"
}
```

## Рекомендации относительно расписания

Несмотря на то что Direct Line — это составной протокол с потенциальными пробелами в расписании, протокол и служба предназначены для упрощения создания надежных клиентов. Свойство `watermark`, которое отправляется в ответе Get Messages, является надежным. Клиент не пропустит ни одного сообщения, пока воспроизводит водяной знак дословно.

Клиенты должны выбрать интервал опроса, который соответствует их предполагаемому использованию.

- Приложения типа "служба — служба" часто используют интервал опроса, равный 5 или 10 с.
- Приложения, взаимодействующие с клиентами, часто используют интервал опроса, равный 1 с, и выдают дополнительный запрос приблизительно через 300 мс после каждого сообщения, отправленного клиентом (для быстрого получения ответа от бота). Эта задержка в 300 мс настраивается в зависимости от времени передачи и скорости бота.

## Дополнительные ресурсы

- [Основные понятия](#)
- [Аутентификация](#)
- [Начало общения](#)
- [Отправка сообщения боту](#)

# Справочник по программному интерфейсу Direct Line API 1.1

27.03.2021 • 11 minutes to read • [Edit Online](#)

## IMPORTANT

Эта статья содержит справочные сведения для Direct Line API 1.1. При создании подключения между клиентским приложением и ботом используйте API версии 3.0 для Direct Line.

С помощью Direct Line API 1.1 можно взаимодействовать с ботом в клиентском приложении. Direct Line API 1.1 использует стандартные отраслевые REST и JSON по протоколу HTTPS.

## Базовый универсальный код ресурса

Для доступа к Direct Line API 1.1 используйте этот базовый URI для всех запросов API.

```
https://directline.botframework.com
```

## Заголовки

Кроме стандартных заголовков HTTP-запроса запрос Direct Line API должен включать заголовок `Authorization`, определяющий секрет или токен для аутентификации клиента, который выполняет запрос. Можно указать заголовок `Authorization` с помощью схем "Bearer" или "BotConnector".

### Схема Bearer

```
Authorization: Bearer SECRET_OR_TOKEN
```

### Схема BotConnector

```
Authorization: BotConnector SECRET_OR_TOKEN
```

Дополнительные сведения о том, как получить секрет или токен, который клиент может использовать для аутентификации запросов Direct Line API, см. в руководстве по [аутентификации](#).

## Коды состояния HTTP

[Код состояния HTTP](#), который возвращается с каждым ответом, указывает результат соответствующего запроса.

HTTP STATUS CODE (КОД СОСТОЯНИЯ HTTP)	ЗНАЧЕНИЕ
200	Запрос выполнен успешно.
204	Запрос успешно выполнен, но содержимое не было возвращено.

HTTP STATUS CODE (КОД СОСТОЯНИЯ HTTP)	ЗНАЧЕНИЕ
400	Запрос неправильный или имеет недопустимый формат.
401	Клиент не авторизован для выполнения запроса. Часто этот код состояния возникает, когда заголовок <code>Authorization</code> отсутствует или имеет недопустимый формат.
403	Клиенту запрещено выполнять запрошенную операцию. Часто этот код состояния возникает, потому что заголовок <code>Authorization</code> указывает недопустимый токен или секрет.
404	Запрошенный ресурс не найден. Обычно этот код состояния обозначает, что в запросе указан недопустимый URI.
500	Внутренняя ошибка сервера в службе Direct Line
502	В боте произошла ошибка. Бот недоступен или вернул ошибку. <b>Это обобщенный код ошибки.</b>

## Операции с токенами

Используйте эти операции для создания или обновления токена, который клиент может использовать для доступа к одному диалогу.

ОПЕРАЦИЯ	ОПИСАНИЕ
<a href="#">Создание токена</a>	Создает токен для нового диалога.
<a href="#">Обновление токена</a>	Обновляет токен.

### Generate Token

Создает токен, действующий для одного диалога.

```
POST /api/tokens/conversation
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
<b>Текст запроса</b>	Недоступно
<b>Возвращает</b>	Строка, представляющая токен

### Refresh Token

Обновляет токен.

```
GET /api/tokens/{conversationId}/renew
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
<b>Текст запроса</b>	Недоступно
<b>Возвращает</b>	Строка, представляющая новый токен

## Операции диалога

Используйте эти операции, чтобы открыть общение с ботом и обмениваться сообщениями между клиентом и ботом.

ОПЕРАЦИЯ	ОПИСАНИЕ
<a href="#">Начало диалога</a>	Открывает новый диалог с ботом.
<a href="#">Получение сообщений</a>	Получает сообщения от бота.
<a href="#">Send a Message</a>	Отправка сообщения боту.
<a href="#">Загрузка и отправка файлов</a>	Загружает и отправляет файлы в виде вложений.

### Начало диалога

Открывает новый диалог с ботом.

```
POST /api/conversations
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
<b>Текст запроса</b>	Недоступно
<b>Возвращает</b>	Объект <a href="#">Conversation</a> .

### Get Messages

Получает сообщения от бота для указанного общения. Задайте параметр `watermark` в URI запроса, чтобы указать последнее сообщение, увиденное клиентом.

```
GET /api/conversations/{conversationId}/messages?watermark={watermark_value}
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
<b>Текст запроса</b>	Недоступно
<b>Возвращает</b>	Объект <a href="#">MessageSet</a> . Ответ содержит <code>watermark</code> как свойство объекта <code>MessageSet</code> . Клиенты должны просматривать доступные сообщения, перемещая значение <code>watermark</code> до тех пор, пока сообщения не перестанут возвращаться.

### Send a Message

Отправка сообщения боту.

```
POST /api/conversations/{conversationId}/messages
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
<b>Текст запроса</b>	Объект <a href="#">Message</a>
<b>Возвращает</b>	В тексте ответа данные не возвращаются. Служба отвечает кодом состояния HTTP 204, если сообщение было отправлено успешно. Клиент может получить отправленное сообщение (вместе с любыми сообщениями, которые бот отправил клиенту) с помощью операции <a href="#">Получить сообщения</a> .

### Загрузка и отправка файлов

Загружает и отправляет файлы в виде вложений. Задайте параметр `userId` в URI запроса, чтобы указать идентификатор пользователя, отправляющего вложения.

```
POST /api/conversations/{conversationId}/upload?userId={userId}
```

СОДЕРЖИМОЕ	ОПИСАНИЕ
<b>Текст запроса</b>	Для одного вложения заполните текст запроса содержимым файла. Для нескольких вложений создайте текст составного запроса, содержащего одну часть для каждого вложения, а также (необязательно) одну часть для объекта <a href="#">Message</a> , который должен служить контейнером для указанного(ых) вложения(й). Дополнительные сведения см. в разделе <a href="#">Отправка сообщения боту</a> .
<b>Возвращает</b>	В тексте ответа данные не возвращаются. Служба отвечает кодом состояния HTTP 204, если сообщение было отправлено успешно. Клиент может получить отправленное сообщение (вместе с любыми сообщениями, которые бот отправил клиенту) с помощью операции <a href="#">Получить сообщения</a> .

#### NOTE

Загруженные файлы удаляются через 24 часа.

## СХЕМА

Схема Direct Line 1.1 представляет собой упрощенную копию схемы Bot Framework версии 1, которая включает следующие объекты.

### Объект "Message"

Определяет сообщение, которое клиент отправляет боту или получает от бота.

СВОЙСТВО	ТИП	ОПИСАНИЕ
----------	-----	----------

СВОЙСТВО	ТИП	ОПИСАНИЕ
идентификатор	строка	Идентификатор, который уникально идентифицирует сообщение (назначается службой Direct Line).
conversationId	строка	Идентификатор, определяющий общение.
created	строка	Дата и время создания сообщения, выраженные в формате <a href="#">ISO-8601</a> .
from	строка	Идентификатор, определяющий пользователя, который является отправителем сообщения. При создании сообщения клиенты должны установить это свойство в стабильный идентификатор пользователя. Несмотря на то что Direct Line назначит идентификатор пользователя, если он не указан, это обычно приводит к непредвиденному поведению.
text	строка	Текст сообщения, отправляемого от пользователя к боту или от бота к пользователю.
channelData	объект	Объект, содержащий определяемое каналом содержимое. Некоторые каналы предоставляют функции, требующие дополнительных сведений, которые не могут быть представлены с помощью схемы вложения. В таких случаях для этого свойства задается определяемое каналом содержимое, как указано в документации канала. Эти данные отправляются без изменений между клиентом и ботом. Это свойство необходимо присвоить составному объекту, либо оставить поле пустым. Не устанавливайте его на строку, номер или другой простой тип.
images	string[]	Массив строк, содержащий URL-адреса для образов, которые содержат сообщение. В некоторых случаях строки в этом массиве могут быть относительными URL-адресами. Если какая-либо строка в этом массиве не начинается с "http" или "https", добавьте <a href="https://directline.botframework.com">https://directline.botframework.com</a> к строке, чтобы сформировать полный URL-адрес.

СВОЙСТВО	ТИП	ОПИСАНИЕ
attachments	Attachment[]	<p>Массив объектов Attachment, которые представляют собой вложения без изображений, содержащиеся в сообщении.</p> <p>Каждый объект в массиве содержит свойство <code>url</code> и свойство <code>contentType</code>. В сообщениях, которые клиент получает от бота, свойство <code>url</code> иногда может указывать относительный URL-адрес. Для любого значения свойства <code>url</code>, которое не начинается с "http" или "https", добавьте <a href="https://directline.botframework.com">https://directline.botframework.com</a> к строке, чтобы сформировать полный URL-адрес.</p>

В следующем примере показан объект "Message", который содержит все возможные свойства. В большинстве случаев при создании сообщения клиенту нужно только предоставить свойство `from` и по меньшей мере одно свойство содержимого (например, `text`, `images`, `attachments` или `channelData`).

```
{
  "id": "CuvLPID4kDb|00000000000000000004",
  "conversationId": "CuvLPID4kDb",
  "created": "2016-10-28T21:19:51.0357965Z",
  "from": "examplebot",
  "text": "Hello!",
  "channelData": {
    "examplefield": "abc123"
  },
  "images": [
    "/attachments/CuvLPID4kDb/0.jpg?..."
  ],
  "attachments": [
    {
      "url": "https://example.com/example.docx",
      "contentType": "application/vnd.openxmlformats-officedocument.wordprocessingml.document"
    },
    {
      "url": "https://example.com/example.doc",
      "contentType": "application/msword"
    }
  ]
}
```

## Объект MessageSet

Определяет набор сообщений.

СВОЙСТВО	ТИП	ОПИСАНИЕ
messages	Message[]	Массив объектов Message.

СВОЙСТВО	ТИП	ОПИСАНИЕ
watermark	строка	Максимальный водяной знак сообщений в наборе. Клиент может использовать значение <code>watermark</code> , чтобы указать последнее сообщение, которое он видел при получении сообщений от бота.

### Объект Attachment

Определяет вложение без изображения.

СВОЙСТВО	ТИП	ОПИСАНИЕ
contentType	строка	Тип мультимедиа содержимого во вложении.
url	строка	URL-адрес для содержимого вложения.

### Объект Conversation

Определяет диалог Direct Line.

СВОЙСТВО	ТИП	ОПИСАНИЕ
conversationId	строка	Идентификатор, который уникально идентифицирует диалог, для которого действует указанный токен.
token	строка	Токен, действующий для указанного диалога.
expires_in	number	Число секунд до истечения срока действия токена.

### Объект ошибки

Определяет ошибку.

СВОЙСТВО	ТИП	ОПИСАНИЕ
code	строка	Код ошибки. Одно из следующих значений: <code>MissingProperty</code> , <code>MalformedData</code> , <code>NotFound</code> , <code>ServiceError</code> , <code>Internal</code> , <code>InvalidRange</code> , <code>NotSupported</code> , <code>NotAllowed</code> , <code>BadCertificate</code> .
message	строка	Текстовое описание ошибки.
statusCode	number	Код состояния.

### Объект ErrorMessage

Полезные данные стандартизированного сообщения об ошибке.

СВОЙСТВО	ТИП	ОПИСАНИЕ
error	Error	Объект Error, содержащий сведения об ошибке.

# Программа командной строки Bot Framework

27.10.2020 • 2 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Программа командной строки Bot Framework заменяет набор изолированных инструментов, предназначенных для управления ботами Bot Framework и связанными с ними службами. Большинство инструментов уже перенесено, а остальные будут перенесены в будущих выпусках. CLI объединяет набор кроссплатформенных инструментов, обеспечивая единый и согласованный интерфейс.

Устаревшие инструменты в последующих выпусках будут считаться нерекомендуемыми. Все новые разработки, исправления ошибок и новые функции будут реализовываться только в новом объединенном Bot Framework CLI.

## Дополнительные сведения

- [Репозиторий инструментов Bot Framework](#)
- [Справочные материалы по Bot Framework CLI](#)

# События и триггеры в адаптивных диалоговых окнах — справочное руководство

27.03.2021 • 13 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Общие сведения об этом разделе см. в статье концепция [событий и триггеров в адаптивных диалоговых окнах](#).

## Базовый триггер

Триггер `OnCondition` является базовым триггером, от которого наследуют все остальные триггеры.

Триггеры в адаптивном диалоге определяются как список объектов `OnCondition`, как показано в следующем примере.

```
Triggers = new List<OnCondition>()
{
    ...
}
```

## Триггеры событий распознавателя

ПРИЧИНА СОБЫТИЯ	ИМЯ ТРИГГЕРА	БАЗОВОЕ СОБЫТИЕ	ОПИСАНИЕ
Выбор намерения	<code>OnChooseIntent</code>	<code>ChooseIntent</code>	Этот триггер выполняется при обнаружении неоднозначности результатов из нескольких распознавателей в <code>CrossTrainedRecognizerSet</code> .
Распознано намерение	<code>OnIntent</code>	<code>RecognizedIntent</code>	Действия, выполняемые при распознавании указанного намерения.
Намерение QnAMatch	<code>OnQnAMatch</code>	<code>RecognizedIntent</code>	Этот триггер выполняется, когда <code>QnAMakerRecognizer</code> возвращает намерение <code>QnAMatch</code> . Сущность <code>@answer</code> будет содержать ответ <code>QnAMaker</code> .

ПРИЧИНА СОБЫТИЯ	ИМЯ ТРИГГЕРА	БАЗОВОЕ СОБЫТИЕ	ОПИСАНИЕ
Распознано неизвестное намерение	OnUnknownIntent	UnknownIntent	<p>Действия, выполняемые, когда входные данные пользователя не распознаны или не соответствуют ни одному из триггеров OnIntent . Его также можно использовать в качестве первого триггера в корневом диалоге вместо OnBeginDialog , чтобы выполнить все необходимые задачи при первом запуске диалога.</p>

#### TIP

Используйте OnUnknownIntent триггер для перехвата и реагирования при возникновении цели "нет".

Использование OnIntent триггера для управления намерением «нет» может привести к непредвиденным результатам.

#### Пример события распознавателя

Ниже приведены примеры триггеров OnIntent И OnUnknownIntent , чтобы продемонстрировать использование триггеров распознавателя.

#### NOTE

- Триггер OnIntent позволяет обработать событие recognizedIntent . Событие вызывается распознавателем . За исключением распознавателя QnA Maker , все встроенные РАСПОЗНАВАТЕЛИ пакета SDK для Bot создают это событие, когда они успешно обнаруживают входные данные пользователя, чтобы программа-робот могла правильно реагировать на запросы.
- Используйте триггер OnUnknownIntent , чтобы перехватывать ввод и реагировать, если событие recognizedIntent не было перехвачено и обработано каким-либо другим триггером . Это означает, что любое необработанное назначение (включая "нет") может вызвать срабатывание триггера, но только в том случае, если в диалоговом окне не выполняются какие-либо действия.

```

// Create the root dialog as an Adaptive dialog.
var rootDialog = new AdaptiveDialog(nameof(AdaptiveDialog));

// Add a regex recognizer
rootDialog.Recognizer = new RegexRecognizer()
{
    Intents = new List<IntentPattern>()
    {
        new IntentPattern()
        {
            Intent = "HelpIntent",
            Pattern = "(?i)help"
        },
        new IntentPattern()
        {
            Intent = "CancelIntent",
            Pattern = "(?i)cancel|never mind"
        }
    }
};

// Create an OnIntent trigger named "helpTrigger" that handles the intent named "HelpIntent".
var helpTrigger = new OnIntent("HelpIntent");

// Create a list of actions to execute when the trigger named "helpTrigger" fires.
var helpActions = new List<Dialog>();
helpActions.Add(new SendActivity("Hello, I'm the samples bot. At the moment, I respond to only help!"));
helpTrigger.Actions = helpActions;

// Add the OnIntent trigger "helpTrigger" to the root dialog
rootDialog.Triggers.Add(helpTrigger);

// Create a trigger to handle the unhandled intent events. The unknown intent trigger fires when a
// recognizedIntent event raised by the recognizer is not handled by any OnIntent trigger in the dialog.
// Given the RegEx recognizer added to this dialog, this trigger will fire when the user says 'cancel'.
// The RegexRecognizer returned the 'cancel' intent, however, we have no trigger attached to handled it.
// The OnUnknownIntent trigger will also fire when user says 'yo' (or any other word that does not map
// to any intent in the recognizer). When the recognizer parses the user input and does not detect an
// intent, it will return a 'none' intent and since there is no OnIntent trigger to handle a 'none'
// intent, the unknown intent trigger fires.
var unhandledIntentTrigger = new OnUnknownIntent();
var unhandledIntentActions = new List<Dialog>();
unhandledIntentActions.Add(new SendActivity("Sorry, I did not recognize that"));
unhandledIntentTrigger.Actions = unhandledIntentActions;

// Add the OnUnknownIntent trigger "unhandledIntentTrigger" to the root dialog
rootDialog.Triggers.Add(unhandledIntentTrigger);

```

## Триггеры событий диалога

Триггеры диалогового окна обрабатывают события, относящиеся к диалоговому окну, которые связаны с **жизненным циклом** диалогового окна. В настоящее время в пакете SDK для Bot Framework доступны 6 триггеров диалога, и все они являются производными от класса `OnDialogEvent`.

### TIP

Они не подобны обычным обработчикам событий прерывания, где действия дочернего элемента будут продолжать выполняться после завершения действий обработчика. Для всех событий, указанных ниже, Bot запустит новый набор действий и завершит свою работу после завершения этих действий.

Имя триггера	Базовое событие	Описание
OnBeginDialog	BeginDialog	Действия, выполняемые при запуске этого диалога. Используется только для дочерних диалогов, но не для корневого диалога. В корневых диалогах используйте <code>OnUnknownIntent</code> для выполнения действий инициализации диалога.
OnCancelDialog	CancelDialog	Это событие позволяет предотвратить отмену текущего диалога из-за дочернего диалога, выполняющего действие <code>CancelAllDialogs</code> .
OnEndOfActions	EndOfActions	Это событие порождается после обработки всех действий и событий неоднозначности.
OnError	Error	Действия, выполняемые при возникновении события диалога. Это событие аналогично тому <code>OnCancelDialog</code> , что вы не мешаете адаптивному диалогу, содержащему этот триггер, в данном случае из-за ошибки в дочернем диалоговом окне.
OnRepromptDialog	RepromptDialog	Действия, выполняемые при возникновении события <code>RepromptDialog</code> .
OnDialog	DialogEvents.VersionChanged	

#### TIP

При использовании [декларативного](#) подхода к адаптивным диалоговым окнам диалоговые окна определяются как `.dialog` файлы, которые используются для создания диалоговых окон во время выполнения. Эти диалоговые окна можно также изменить во время выполнения, обновив `.dialog` файл непосредственно, затем обрабатывая `resourceExplorer.Changed` событие, чтобы перезагрузить диалоговое окно. Можно также записать `DialogEvents.VersionChanged` событие в [триггере](#), чтобы выполнить все необходимые действия, которые могут возникнуть в результате изменения диалогового окна в середине диалога с пользователем. Дополнительные сведения см. в разделе [Автоматическая перезагрузка диалоговых окон при изменении файлов](#) в разделе [Использование декларативных ресурсов в адаптивных диалоговых окнах](#).

#### Пример события диалогового окна

В этом примере показано, как отправить пользователю приветствие с помощью триггера `OnBeginDialog`.

```

var welcomeDialog = new AdaptiveDialog(nameof(AdaptiveDialog))
{
    Triggers = new List<OnCondition>()
    {
        new OnBeginDialog()
        {
            Actions = new List<Dialog>()
            {
                new SendActivity("Hello world!")
            }
        }
    }
};

```

## Триггеры событий действия

Триггеры действия позволяют связать действия с любым входящим действием клиента. Например, когда новый пользователь присоединяется и бот начинает новую беседу. Дополнительные сведения о действиях можно найти в разделе [Схема действия в Bot Framework](#).

Все события действия имеют базовое событие `ActivityReceived`, которое дополнительно уточняется с помощью *типа действия*. Базовый класс, от которого наследуются все триггеры действия, — `OnActivity`.

ПРИЧИНА СОБЫТИЯ	ACTIVITYTYPE	ИМЯ ТРИГГЕРА	ОПИСАНИЕ
Greeting	<code>ConversationUpdate</code>	<code>OnConversationUpdateActivity</code>	Действия, выполняемые <code>conversationUpdate</code> при получении действия, когда Bot или пользователь присоединяется или оставляет диалог.
Беседа завершена	<code>EndOfConversation</code>	<code>OnEndOfConversationActivity</code>	Действия, выполняемые при получении <code>endOfConversation</code> действия.
Событие получено	<code>Event</code>	<code>OnEventActivity</code>	Действия, выполняемые при получении <code>event</code> действия.
Передача человеку	<code>Handoff</code>	<code>OnHandoffActivity</code>	Действия, выполняемые при получении <code>handoff</code> действия.
Инициирована беседа	<code>Invoke</code>	<code>OnInvokeActivity</code>	Действия, выполняемые при получении <code>invoke</code> действия.
Пользователь вводит сообщение	<code>Typing</code>	<code>OnTypingActivity</code>	Действия, выполняемые при получении <code>typing</code> действия.

### Пример события действия

#### `OnConversationUpdateActivity`

`OnConversationUpdateActivity` Триггер позволяет обрабатывать событие, *полученное действием*.

`OnConversationUpdateActivity` Триггер срабатывает, только если полученное действие является `conversationUpdate` действием.

В следующем фрагменте кода показано, как создать триггер `onConversationUpdateActivity`.

```
var myDialog = new AdaptiveDialog(nameof(AdaptiveDialog))
{
    Generator = new TemplateEngineLanguageGenerator(new TemplateEngine().AddFile(Path.Combine(".", "myDialog.lg"))),
    Triggers = new List<OnCondition>() {
        new OnConversationUpdateActivity() {
            Actions = new List<Dialog>() {
                new SendActivity("${Welcome-user()}")
            }
        }
    }
};
```

## Триггеры событий сообщения

Триггеры событий сообщения позволяют реагировать на любое событие сообщения, например, когда сообщение изменено (`MessageUpdate`) либо удалено (`MessageDeletion`), или когда кто-либо реагирует (`MessageReaction`) на сообщение (например, к распространенным реакциям на сообщение относятся такие действия, как отметка "Нравится", "Сердечко", "Смех", "Удивление" и "Злость").

События сообщения относятся к типу события действия, поэтому все события сообщения имеют базовое событие `ActivityReceived`, которое дополнительно уточняется с помощью *типа действия*. Базовый класс, от которого наследуются все триггеры сообщения, — `OnActivity`.

ПРИЧИНА СОБЫТИЯ	ACTIVITYTYPE	ИМЯ ТРИГГЕРА	ОПИСАНИЕ
Полученное сообщение	<code>Message</code>	<code>OnMessageActivity</code>	Действия, выполняемые при получении действия типа <code>MessageReceived</code> .
Сообщение удалено	<code>MessageDeletion</code>	<code>OnMessageDeleteActivity</code>	Действия, выполняемые при получении действия типа <code>MessageDelete</code> .
Реагирование на сообщение	<code>MessageReaction</code>	<code>OnMessageReactionActivity</code>	Действия, выполняемые при получении действия типа <code>MessageReaction</code> .
Сообщение изменено	<code>MessageUpdate</code>	<code>OnMessageUpdateActivity</code>	Действия, выполняемые при получении действия типа <code>MessageUpdate</code> .

## Триггер настраиваемого события

Можно порождать собственные события, добавив действие `EmitEvent` в любой триггер. Затем можно будет обработать это пользовательское событие в любом триггере любого диалога бота, определив триггер *настраиваемого события*. Триггер настраиваемого события — это триггер `OnDialogEvent`, который фактически становится настраиваемым триггером, если для свойства `Event` задано то же значение, что для свойства `EventName` действия `EmitEvent`.

**TIP**

Можно разрешить другим диалогам бота работать с настраиваемым событием, задав для свойства `BubbleEvent` действия `EmitEvent` значение `true`.

ПРИЧИНА СОБЫТИЯ	ИМЯ ТРИГГЕРА	БАЗОВЫЙ КЛАСС	ОПИСАНИЕ
Настраиваемое событие	<code>OnDialogEvent</code>	<code>OnCondition</code>	Действия, выполняемые при обнаружении настраиваемого события. Используйте действие <a href="#">выдать настраиваемое событие</a> для вызова пользовательского события.

## Пример настраиваемого события

```
var rootDialog = new AdaptiveDialog(nameof(AdaptiveDialog))
{
    Generator = new TemplateEngineLanguageGenerator(),
    Triggers = new List<OnCondition>()
    {
        new OnUnknownIntent()
        {
            Actions = new List<Dialog>()
            {
                new TextInput()
                {
                    Prompt = new ActivityTemplate("What's your name?"),
                    Property = "user.name",
                    AlwaysPrompt = true,
                    OutputFormat = "toLowerCase(this.value)"
                },
                new EmitEvent()
                {
                    EventName = "contoso.custom",
                    EventValue = "=user.name",
                    BubbleEvent = true,
                },
                new SendActivity("Your name is ${user.name}"),
                new SendActivity("And you are ${userType}")
            }
        },
        new OnDialogEvent()
        {
            Event = "contoso.custom",

            // You can use conditions (expression) to examine value of the event as part of the trigger
            selection process.
            Condition = "turn.dialogEvent.value && (substring(turn.dialogEvent.value, 0, 1) == 'v')",
            Actions = new List<Dialog>()
            {
                new SendActivity("In custom event: '${turn.dialogEvent.name}' with the following value
'${turn.dialogEvent.value}'"),
                new SetProperty()
                {
                    Property = "$userType",
                    Value = "VIP"
                }
            }
        },
        new OnDialogEvent()
        {
    }
```

```

        Event = "contoso.custom",

        // You can use conditions (expression) to examine value of the event as part of the trigger
selection process.
        Condition = "turn.dialogEvent.value && (substring(turn.dialogEvent.value, 0, 1) == 's')",
        Actions = new List<Dialog>()
        {
            new SendActivity("In custom event: '${turn.dialogEvent.name}' with the following value
'${turn.dialogEvent.value}'''),
            new SetProperty()
            {
                Property = "$userType",
                Value = "Special"
            }
        },
        new OnDialogEvent()
        {
            Event = "contoso.custom",
            Actions = new List<Dialog>()
            {
                new SendActivity("In custom event: '${turn.dialogEvent.name}' with the following value
'${turn.dialogEvent.value}'''),
                new SetProperty()
                {
                    Property = "$userType",
                    Value = "regular customer"
                }
            }
        }
    );

```

## Дополнительные сведения

- [Общие сведения об адаптивных диалогах](#)
- [Библиотека диалогов](#)
- [Действия в адаптивных диалогах](#)

# Действия в адаптивных диалоговых окнах — справочное руководство

27.10.2020 • 24 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье перечислены действия, определенные в пакете SDK для Bot Framework, сгруппированные по их общему назначению.

- Общие сведения об этом разделе см. в статье концепция [действий в адаптивных диалоговых окнах](#).
- Сведения о том, как запрашивать ввод пользователя, — важный и очень полезный тип действия, см. в разделе [запрос ввода данных пользователем с помощью адаптивных диалоговых окон](#).

## Действия

ДЕЙСТВИЕ ДЛЯ ВЫПОЛНЕНИЯ	НАЗВАНИЕ ДЕЙСТВИЯ	НАЗНАЧЕНИЕ ЭТОГО ДЕЙСТВИЯ
Отправка любых действий, например ответов пользователю.	<a href="#">SendActivity</a>	Отправляет действие, например ответ пользователю.
Обновление действия	<a href="#">UpdateActivity</a>	Обновляет отправленное действие.
Удаление действия	<a href="#">DeleteActivity</a>	Удаляет отправленное действие.
Получение элементов действия	<a href="#">GetActivityMembers</a>	Возвращает список членов действия и сохраняет их в свойство в <a href="#">памяти</a> .

Пример кода см. в статье [примеры действий](#).

## Условные операторы

Первые два действия — это условные операторы, которые позволяют боту принимать решения по заданным вами условиям. Эти условия задаются набором условных операторов с логическими выражениями, которые возвращают логическое значение true или false.

Остальные действия связаны с операторами цикла, которые позволяют повторить выполнение блока кода для каждого элемента в коллекции.

ДЕЙСТВИЕ ДЛЯ ВЫПОЛНЕНИЯ	НАЗВАНИЕ ДЕЙСТВИЯ	НАЗНАЧЕНИЕ ЭТОГО ДЕЙСТВИЯ
Ветвление: if/else	<a href="#">IfCondition</a>	Выполняет набор действий на основе логического выражения.
Ветвление: Switch (несколько вариантов)	<a href="#">SwitchCondition</a>	Выполняет набор действий на основе соответствия шаблону.
Цикл: для каждого элемента	<a href="#">ForEach</a>	Выполняет цикл по набору значений, сохраненных в массиве.

ДЕЙСТВИЕ ДЛЯ ВЫПОЛНЕНИЯ	НАЗВАНИЕ ДЕЙСТВИЯ	НАЗНАЧЕНИЕ ЭТОГО ДЕЙСТВИЯ
Цикл: для каждой страницы (несколько элементов)	ForEachPage	Циклический перебор большого набора значений, хранящихся в массиве, по одной странице за раз.
Выход из цикла	BreakLoop	Завершает внешний цикл.
Продолжение цикла	ContinueLoop	Запускает следующую итерацию включающего цикла.
Переход к другому действию	GotoAction	Передает управление указанному действию, определяемому ИДЕНТИФИКАТОРом действия.

См. также [примеры создания условий](#).

## Управление диалогами

ДЕЙСТВИЕ ДЛЯ ВЫПОЛНЕНИЯ	НАЗВАНИЕ ДЕЙСТВИЯ	НАЗНАЧЕНИЕ ЭТОГО ДЕЙСТВИЯ
Начало нового диалога	BeginDialog	Начинает выполнять другой диалог. Когда диалог завершится, будет возобновлено выполнение текущего триггера.
Отмена диалога	CancelDialog	Отменяет активный диалог. Используется, когда нужно немедленно закрыть диалог, даже не завершая текущий процесс.
Отмена всех диалогов	CancelAllDialogs	Отменяет все активные диалоги, включая родительские. Используйте это действие, если нужно извлечь все диалоги из стека, а очистить стек диалогов можно с помощью метода отмены всех диалогов в контексте диалога. Создает событие <code>CancelAllDialogs</code> .
Завершение диалога	EndDialog	Завершает активный диалог. Используйте это действие, если перед выходом нужно завершить работу диалога и возвратить результаты. Создает событие <code>EndDialog</code> .
Завершение шага диалога	EndTurn	Завершает текущий шаг диалога, не закрывая сам диалог.
Повторение текущего диалога	RepeatDialog	Используется для перезапуска родительского диалога.
Замена текущего диалога	ReplaceDialog	Замена текущего диалога новым диалогом.

ДЕЙСТВИЕ ДЛЯ ВЫПОЛНЕНИЯ	НАЗВАНИЕ ДЕЙСТВИЯ	НАЗНАЧЕНИЕ ЭТОГО ДЕЙСТВИЯ
Получение участников беседы	GetConversationMembers	Позволяет получить список участников беседы и сохранить их в свойство в <a href="#">памяти</a> .
Изменение действий	EditActions	Позволяет изменять в режиме реального времени текущую последовательность действий на основе введенных пользователем данных. Особенно полезно при обработке <a href="#">прерываний</a> .

См. [примеры кода для управления диалогами](#).

## Управление свойствами

ДЕЙСТВИЕ ДЛЯ ВЫПОЛНЕНИЯ	НАЗВАНИЕ ДЕЙСТВИЯ	НАЗНАЧЕНИЕ ЭТОГО ДЕЙСТВИЯ
Изменение массива	EditArray	Выполняет операцию с массивом.
Удаление свойства	DeleteProperty	Удаляет свойство из <a href="#">памяти</a> .
Удаление свойств	DeleteProperties	Удаляет несколько свойств одновременно.
Создание или обновление свойства	SetProperty	Задает значение свойства в <a href="#">памяти</a> .
Создание или обновление свойств	SetProperties	Задает значение нескольких свойств одновременно.

См. [примеры кода для управления свойствами](#).

## Доступ к внешним ресурсам

ДЕЙСТВИЕ ДЛЯ ВЫПОЛНЕНИЯ	НАЗВАНИЕ ДЕЙСТВИЯ	НАЗНАЧЕНИЕ ЭТОГО ДЕЙСТВИЯ
Запуск диалога навыка	BeginSkill	Начинает навык и пересыпает действия навыкам до окончания навыка.
Отправка HTTP-запроса	HttpRequest	Выполняет HTTP-запрос к конечной точке.
Запуск пользовательского события	EmitEvent	Вызывает пользовательское событие. Добавьте <a href="#">пользовательский триггер</a> в Аддитивное диалоговое окно, чтобы отреагировать на событие.
Выход пользователя	SignOutUser	Подписывает текущего пользователя, выполнившего вход.

ДЕЙСТВИЕ ДЛЯ ВЫПОЛНЕНИЯ	НАЗВАНИЕ ДЕЙСТВИЯ	НАЗНАЧЕНИЕ ЭТОГО ДЕЙСТВИЯ
Вызов пользовательского кода	<a href="#">CodeAction</a>	Вызывает пользовательский код. Пользовательский код должен быть асинхронным, принимать контекст диалогового окна и объект в качестве параметров и возвращать результат диалогового окна.

См. [примеры кода для доступа к внешним ресурсам](#).

## Варианты для отладки

ДЕЙСТВИЕ ДЛЯ ВЫПОЛНЕНИЯ	НАЗВАНИЕ ДЕЙСТВИЯ	НАЗНАЧЕНИЕ ЭТОГО ДЕЙСТВИЯ
Вход в консоль	<a href="#">LogAction</a>	Записывает сообщение в консоль и (необязательно) отправляет его как действие трассировки.
Создание события трассировки	<a href="#">TraceActivity</a>	Отправляет действие трассировки с произвольными полезными данными.

См. [примеры инструментов для отладки](#).

## Примеры исходного кода

### Примеры действий

#### Отправка действия

Отправляет действие.

```
// Example of a simple SendActivity step
var greetUserDialog = new AdaptiveDialog("greetUserDialog");
greetUserDialog.Triggers.Add(new OnIntent()
{
    Intent = "greetUser",
    Actions = new List<Dialog>() {
        new SendActivity("Hello")
    }
});

// Example that includes reference to property on bot state.
var greetUserDialog = new AdaptiveDialog("greetUserDialog");
greetUserDialog.Triggers.Add(new OnIntent()
{
    Intent = "greetUser",
    Actions = new List<Dialog>()
    {
        new TextInput()
        {
            Property = "user.name",
            Prompt = new ActivityTemplate("What is your name?")
        },
        new SendActivity("Hello, ${user.name}")
    }
});
```

#### Обновление действия

Обновляет ранее отправленное действие. Идентификатор действия возвращается из вызова

`SendActivity`.

```
new UpdateActivity ()  
{  
    ActivityId = "id",  
    Activity = new ActivityTemplate("updated value")  
}
```

#### **Удаление действия**

Удаляет ранее отправленное действие. Требует указать идентификатор предыдущего действия.

```
new DeleteActivity ()  
{  
    ActivityId = "id"  
}
```

#### **Получение элементов действия**

Возвращает элементы действия, связанного с текущим шагом, и сохраняет этот список в свойство.

```
new GetActivityMembers()  
{  
    Property = "turn.activityMembers"  
}
```

Дополнительные сведения о том, как в адаптивных диалогах применить в *действии отправки* создание текста вместо заранее сохраненных текстов ответа см. [здесь](#).

### **Примеры создания условий**

#### **IfCondition**

Выполняет ветвление потока беседы на основе определенного условия. Условия выражаются с помощью [адаптивных выражений](#).

```

var addToDoDialog = new AdaptiveDialog("addToDoDialog");
addToDoDialog.Triggers.Add(new OnIntent()
{
    Intent = "addToDo",
    Actions = new List<Dialog>()
    {
        // Save the userName entity from a recognizer.
        new SaveEntity("dialog.addTodo.title", "@todoTitle"),
        new TextInput()
        {
            Prompt = new ActivityTemplate("What is the title of your todo?"),
            Property = "dialog.addTodo.title"
        },
        // Add the current todo to the todo's list for this user.
        new EditArray()
        {
            ItemsProperty = "user.todos",
            Value = "=dialog.addTodo.title"
            ChangeType = EditArray.ArrayChangeType.Push
        },
        new SendActivity("Ok, I have added ${dialog.addTodo.title} to your todos."),
        new IfCondition()
        {
            Condition = "toLower(dialog.addTodo.title) == 'call santa'",
            Actions = new List<Dialog>()
            {
                new SendActivity("Yes master. On it right now \\[You have unlocked an easter egg] :)")
            }
        },
        new SendActivity("You now have ${count(user.todos)} items in your todo.")
    }
});

```

### **SwitchCondition**

Выполняет ветвление потока беседы на основе результата оценки выражения. Дополнительные сведения об адаптивных выражениях см. [здесь](#).

```

// Create an adaptive dialog.
var cardDialog = new AdaptiveDialog("cardDialog");
cardDialog.Triggers.Add(new OnIntent()
{
    Intent = "ShowCards",
    Actions = new List<Dialog>()
    {
        // Add choice input.
        new ChoiceInput()
        {
            // Output from the user is automatically set to this property
            Property = "turn.cardDialog.cardChoice",

            // List of possible styles supported by choice prompt.
            Style = ListStyle.Auto,
            Prompt = new ActivityTemplate("What card would you like to see?"),
            Choices = new ChoiceSet(new List<Choice>() {
                new Choice("Adaptive card"),
                new Choice("Hero card"),
                new Choice("Video card")
            })
        },
        // Use SwitchCondition step to dispatch to right dialog based on choice input.
        new SwitchCondition()
        {
            Condition = "turn.cardDialog.cardChoice",
            Cases = new List<Case>()
            {
                new Case("Adaptive card", new List<Dialog>() { new SendActivity("${AdaptiveCardRef()}") }),
                new Case("Hero card", new List<Dialog>() { new SendActivity("${HeroCard()}") } ),
                new Case("Video card", new List<Dialog>() { new SendActivity("${VideoCard()}") } ),
            },
            Default = new List<Dialog>()
            {
                new SendActivity("[AllCards]")
            }
        }
    });
});
```

### ForEach

Цикл foreach будет удобным для извлечения элементов массива или коллекции. Он часто используется для выполнения действия с каждым элементом коллекции.

```

var rootDialog = new AdaptiveDialog(nameof(AdaptiveDialog))
{
    Generator = new TemplateEngineLanguageGenerator(),
    Triggers = new List<OnCondition>()
    {
        new OnUnknownIntent()
        {
            Actions = new List<Dialog>()
            {
                new SetProperty()
                {
                    Property = "turn.colors",
                    Value = "=createArray('red', 'blue', 'green', 'yellow', 'orange', 'indigo')"
                },
                new Foreach()
                {
                    ItemsProperty = "turn.colors",
                    Actions = new List<Dialog>()
                    {
                        // By default, dialog.foreach.value holds the value of the item
                        // dialog.foreach.index holds the index of the item.
                        // You can use short hands to refer to these via
                        //     $foreach.value
                        //     $foreach.index
                        new SendActivity("${$foreach.index}: Found '${$foreach.value}' in the collection!")
                    }
                }
            }
        }
    }
};

```

#### **ForEachPage**

Позволяет применить некоторые шаги к элементам коллекции. Размер страницы определяет, сколько элементов извлекается из коллекции за один цикл.

```
var rootDialog = new AdaptiveDialog(nameof(AdaptiveDialog))
{
    Generator = new TemplateEngineLanguageGenerator(),
    Triggers = new List<OnCondition>()
    {
        new OnUnknownIntent()
        {
            Actions = new List<Dialog>()
            {
                new SetProperty()
                {
                    Property = "turn.colors",
                    Value = "=createArray('red', 'blue', 'green', 'yellow', 'orange', 'indigo')"
                },
                new ForeachPage()
                {
                    ItemsProperty = "turn.colors",
                    PageSize = 2,
                    Actions = new List<Dialog>()
                    {
                        // By default, dialog.foreach.page holds the value of the page
                        //      $foreach.page
                        new SendActivity("Page content: ${join($foreach.page, ', ')}")
                    }
                }
            }
        }
    };
}
```

#### Прерывание цикла

Прерывает выполнение цикла или текущей последовательности действий.

```

var rootDialog = new AdaptiveDialog(nameof(AdaptiveDialog))
{
    Generator = new TemplateEngineLanguageGenerator(),
    Triggers = new List<OnCondition>()
    {
        new OnUnknownIntent()
        {
            Actions = new List<Dialog>()
            {
                new SetProperty()
                {
                    Property = "turn.colors",
                    Value = "=createArray('red', 'blue', 'green', 'yellow', 'orange', 'indigo')"
                },
                new Foreach()
                {
                    ItemsProperty = "turn.colors",
                    Actions = new List<Dialog>()
                    {
                        new IfCondition()
                        {
                            Condition = "$foreach.value == 'green'",
                            Actions = new List<Dialog>()
                            {
                                new BreakLoop()
                            },
                            ElseActions = new List<Dialog>()
                            {
                                // By default, dialog.foreach.value holds the value of the item
                                // dialog.foreach.index holds the index of the item.
                                // You can use short hands to refer to these via
                                //     $foreach.value
                                //     $foreach.index
                                new SendActivity("${$foreach.index}: Found '${$foreach.value}' in the
collection!")
                            }
                        }
                    }
                }
            }
        }
    };
}

```

#### **Продолжение цикла**

Продолжает выполнять текущий цикл или последовательность действий без обработки оставшихся инструкций в текущей итерации этого цикла.

```

var rootDialog = new AdaptiveDialog(nameof(AdaptiveDialog))
{
    Generator = new TemplateEngineLanguageGenerator(),
    Triggers = new List<OnCondition>()
    {
        new OnUnknownIntent()
        {
            Actions = new List<Dialog>()
            {
                new SetProperty()
                {
                    Property = "turn.colors",
                    Value = "=createArray('red', 'blue', 'green', 'yellow', 'orange', 'indigo')"
                },
                new Foreach()
                {
                    ItemsProperty = "turn.colors",
                    Actions = new List<Dialog>()
                    {
                        new IfCondition()
                        {
                            // Skip items at even position in the collection.
                            Condition = "$foreach.index % 2 == 0",
                            Actions = new List<Dialog>()
                            {
                                new ContinueLoop()
                            },
                            ElseActions = new List<Dialog>()
                            {
                                // By default, dialog.foreach.value holds the value of the item
                                // dialog.foreach.index holds the index of the item.
                                // You can use short hands to refer to these via
                                //     $foreach.value
                                //     $foreach.index
                                new SendActivity("${$foreach.index}: Found '${$foreach.value}' in the
collection!")
                            }
                        }
                    }
                }
            }
        }
    };
};

```

#### **Переход к действию**

Переход к действиям по меткам в пределах текущей области действия.

```

var adaptiveDialog = new AdaptiveDialog()
{
    Triggers = new List<OnCondition>()
    {
        new OnBeginDialog()
        {
            Actions = new List<Dialog>()
            {
                new GotoAction()
                {
                    ActionId = "end"
                },
                new SendActivity("this will be skipped."),
                new SendActivity()
                {
                    Id = "end",
                    Activity = new ActivityTemplate("The End.")
                }
            }
        }
    }
};

```

## Примеры для управления диалогами

### BeginDialog:

Запускает новый диалог и помещает его в стек диалогов. `BeginDialog` требует указать имя целевого диалога, которое может иметь любой тип, включая адаптивный, каскадный и т. д.

Действие `BeginDialog` определяет свойство с именем `ResultProperty`, которое позволяет указать место для сохранения результатов при завершении диалога.

```

new BeginDialog("BookFlightDialog")
{
    // Any value returned by BookFlightDialog will be captured in the property specified here.
    ResultProperty = "$bookFlightResult"
}

```

### TIP

Точно так же, как при вызове любого диалога в пакете SDK для Bot Framework, вызов `BeginDialog` для запуска адаптивного диалога позволяет указать параметр `options` для передачи в диалог входных данных.

### EndDialog

Завершает диалог, извлекая его из стека диалогов, и возвращает в родительский диалог результат выполнения (необязательно).

По умолчанию в адаптивных диалогах для параметра `defaultResultProperty` задано значение `dialog.results`, а значит, все содержимое [области памяти](#) автоматически возвращается вызывающему объекту во всех сценариях автоматического завершения диалога. Если вы завершите диалог с помощью действия `EndDialog`, придется явным образом указать данные, возвращаемые вызывающему объекту, в свойстве `value`.

```

new EndDialog()
{
    // Value property indicates value to return to the caller.
    Value = "=$userName"
}

```

**TIP**

Адаптивные диалоги по умолчанию завершаются автоматически, когда закончат выполнение всех действий.

Чтобы переопределить это поведение, задайте значение `false` для свойства `AutoEndDialog` адаптивного диалога.

**CancelAllDialogs**

Удаляет все диалоги в стеке, включая родительские и дочерние.

```
new CancelAllDialogs()
```

**EndTurn**

Завершает текущий шаг диалога, не закрывая сам диалог.

```
new EndTurn()
```

**RepeatDialog**

Перезапускает родительский диалог. Это особенно полезно, если вам нужно создать диалог с постраничной выдачей результатов от бота пользователю и поддержкой навигации по этим результатам.

**IMPORTANT**

Обязательно используйте для получения сведений от пользователя `EndTurn()` или один из входов, чтобы случайно не создать бесконечный цикл.

```

var rootDialog = new AdaptiveDialog(nameof(AdaptiveDialog))
{
    Triggers = new List<OnCondition>()
    {
        new OnUnknownIntent()
        {
            Actions = new List<Dialog>()
            {
                new TextInput()
                {
                    Prompt = new ActivityTemplate("Give me your favorite color. You can always say cancel to stop this."),
                    Property = "turn.favColor",
                },
                new EditArray()
                {
                    ArrayProperty = "user.favColors",
                    Value = "=turn.favColor",
                    ChangeType = EditArray.ArrayChangeType.Push
                },
                // This is required because TextInput will skip prompt if the property exists - which it will from the previous turn.
                // Alternately you can also set `AlwaysPrompt = true` on the TextInput.
                new DeleteProperty() {
                    Property = "turn.favColor"
                },
                // Repeat dialog step will restart this dialog.
                new RepeatDialog()
            }
        },
        new OnIntent("CancelIntent")
        {
            Actions = new List<Dialog>()
            {
                new SendActivity("You have ${count(user.favColors)} favorite colors - ${join(user.favColors, ', ', 'and')}" ),
                new EndDialog()
            }
        }
    },
    Recognizer = new RegexRecognizer()
    {
        Intents = new List<IntentPattern>()
        {
            new IntentPattern()
            {
                Intent = "HelpIntent",
                Pattern = "(?i)help"
            },
            new IntentPattern()
            {
                Intent = "CancelIntent",
                Pattern = "(?i)cancel|never mind"
            }
        }
    }
};

```

### ReplaceDialog

Запускает новый диалог и помещает его в стек диалогов вместо текущего.

```

// This sample illustrates the use of ReplaceDialog tied to explicit user confirmation
// to switch to a different dialog.

// Create an adaptive dialog.
var getUserName = new AdaptiveDialog("getUserName");
getUserName.Triggers.Add(new OnIntent()
{
    Intent = "getUserName",
    Actions = new List<Dialog>()
    {
        new TextInput()
        {
            Property = "user.name",
            Prompt = new ActivityTemplate("What is your name?")
        },
        new SendActivity("Hello ${user.name}, nice to meet you!")
    }
});

getUserName.Triggers.Add(new OnIntent()
{
    Intent = "GetWeather",
    Actions = new List<Dialog>()
    {
        // confirm with user that they do want to switch to another dialog
        new ChoiceInput()
        {
            Prompt = new ActivityTemplate("Are you sure you want to switch to talk about the weather?"),
            Property = "turn.contoso.getWeather.confirmChoice",
            Choices = new ChoiceSet(new List<Choice>())
            {
                new Choice("Yes"),
                new Choice("No")
            }
        },
        new SwitchCondition()
        {
            Condition = "turn.contoso.getWeather.confirmChoice",
            Cases = new List<Case>()
            {
                // Call ReplaceDialog to switch to a different dialog.
                // BeginDialog will keep current dialog in the stack to be resumed after child dialog ends.
                // ReplaceDialog will remove current dialog from the stack and add the new dialog.
                {
                    Value = "Yes",
                    Actions = new List<Dialog>()
                    {
                        new ReplaceDialog("getWeatherDialog")
                    }
                },
                {
                    Value = "No",
                    Actions = new List<Dialog>()
                    {
                        new EndDialog()
                    }
                }
            }
        }
    }
});

```

#### **Получение участников беседы**

Возвращает список участников активной беседы и сохраняет этот список в свойство.

```
new GetConversationMembers()
{
    Property = "turn.convMembers"
}
```

## EditActions

Изменяет текущую последовательность действий. Особенно полезно при обработке [прерываний](#). EditActions можно использовать для вставки или удаления действий в любом месте последовательности, в том числе добавлять действия в начало или конец последовательности.

```
var rootDialog = new AdaptiveDialog(nameof(AdaptiveDialog))
{
    Generator = new TemplateEngineLanguageGenerator(),
    Recognizer = new RegexRecognizer()
    {
        Intents = new List<IntentPattern>()
        {
            new IntentPattern()
            {
                Intent = "appendSteps",
                Pattern = "(?i)append"
            },
            new IntentPattern()
            {
                Intent = "insertSteps",
                Pattern = "(?i)insert"
            },
            new IntentPattern()
            {
                Intent = "endSteps",
                Pattern = "(?i)end"
            }
        }
    },
    Triggers = new List<OnCondition>()
    {
        new OnUnknownIntent()
        {
            Actions = new List<Dialog>()
            {
                new ChoiceInput()
                {
                    Prompt = new ActivityTemplate("What type of EditAction would you like to see?"),
                    Property = "$userChoice",
                    AlwaysPrompt = true,
                    Choices = new ChoiceSet(new List<Choice>()
                    {
                        new Choice("Append actions"),
                        new Choice("Insert actions"),
                        new Choice("End actions"),
                    })
                },
                new SendActivity("This message is after your EditActions choice..")
            }
        },
        new OnIntent()
        {
            Intent = "appendSteps",
            Actions = new List<Dialog>() {
                new SendActivity("In append steps .. Steps specified via EditSteps will be added to the
current plan."),
                new EditActions()
                {
                    Actions = new List<Dialog>() {
                        // These steps will be appended to the current set of steps being executed.

```

```

        new SendActivity("I was appended!")
    },
    ChangeType = ActionChangeType.AppendActions
}
},
new OnIntent() {
    Intent = "insertSteps",
    Actions = new List<Dialog>() {
        new SendActivity("In insert steps .. "),
        new EditActions()
    {
        Actions = new List<Dialog>() {
            // These steps will be inserted before the current steps being executed.
            new SendActivity("I was inserted")
        },
        ChangeType = ActionChangeType.InsertActions
    }
},
new OnIntent()
{
    Intent = "endSteps",
    Actions = new List<Dialog>()
{
    new SendActivity("In end steps .. "),
    new EditActions()
    {
        // The current sequence will be ended. This is especially useful if you are looking to
        end an active interruption.
        ChangeType = ActionChangeType.EndSequence
    }
}
},
};


```

## Примеры для управления свойствами

### **SetProperty**

Используется, чтобы задать значение для свойства в [памяти](#). Значением может быть явно заданная строка либо выражение. Дополнительные сведения об аддитивных выражениях см. [здесь](#).

```

new SetProperty()
{
    Property = "user.firstName",
    // If the value of user.name is 'Mahatma Gandhi', this sets first name to 'Mahatma'
    Value = "=split(user.name, ' ')[0]"
},

```

### **SetProperties**

Инициализирует одно или несколько свойств одним действием.

```
new SetProperties()
{
    Assignments = new List<PropertyAssignment>()
    {
        new PropertyAssignment()
        {
            Property = "user.name",
            Value = "Vishwac"
        },
        new PropertyAssignment()
        {
            Property = "user.age",
            Value = "=coalesce($age, 30)"
        }
    }
}
```

### DeleteProperty

Удаляет свойство из [памяти](#).

```
new DeleteProperty
{
    Property = "user.firstName"
}
```

### DeleteProperties

Удаляет сразу несколько свойств одним действием.

```
new DeleteProperties()
{
    Properties = new List<StringExpression>()
    {
        new StringExpression("user.name"),
        new StringExpression("user.age")
    }
}
```

### EditArray

Используется, чтобы редактировать операции над свойством массива.

```

var addToDoDialog = new AdaptiveDialog("addToDoDialog");
addToDoDialog.Triggers.Add(new OnIntent()
{
    Intent = "addToDo",
    Actions = new List<Dialog>() {
        // Save the userName entity from a recognizer.
        new SaveEntity("dialog.addTodo.title", "@todoTitle"),
        new TextInput()
        {
            Prompt = new ActivityTemplate("What is the title of your todo?"),
            Property = "dialog.addTodo.title"
        },
        // Add the current todo to the todo's list for this user.
        new EditArray()
        {
            ItemsProperty = "user.todos",
            Value = "=dialog.addTodo.title"
            ChangeType = EditArray.ArrayChangeType.Push
        },
        new SendActivity("Ok, I have added ${dialog.addTodo.title} to your todos."),
        new SendActivity("You now have ${count(user.todos)} items in your todo.")
    });
});

```

## Примеры для доступа к внешним ресурсам

### BeginSkill

Действие `BeginSkill` запускает указанный навык, управляет переадресацией действий в этот навык и получением действий от навыка, а в случае завершения работы навыка обрабатывает его результаты.

### HttpRequest

Используется, чтобы отправить HTTP-запросы к любой конечной точке.

```

new HttpRequest()
{
    // Set response from the http request to turn.httpResponse property in memory.
    ResultProperty = "turn.httpResponse",
    Method = HttpRequest.HttpMethod.POST,
    Header = new Dictionary<string, string>(), /* request header */
    Body = { } /* request body */
};

```

### EmitEvent

Используется, чтобы создать пользовательское событие, на которое может реагировать бот. Вы можете управлять восходящей маршрутизацией для создаваемого события: сохранить его только в пределах текущего диалога или передать по восходящей цепочке родительских элементов.

```

var rootDialog = new AdaptiveDialog(nameof(AdaptiveDialog))
{
    Generator = new TemplateEngineLanguageGenerator(),
    Triggers = new List<OnCondition>()
    {
        new OnUnknownIntent()
        {
            Actions = new List<Dialog>()
            {
                new TextInput()
                {
                    Prompt = new ActivityTemplate("What's your name?"),
                    Property = "user.name",
                    AlwaysPrompt = true,
                    OutputFormat = "toLowerCase(this.value)"
                },
            }
        }
    }
};

```

```

        new EmitEvent()
        {
            EventName = "contoso.custom",
            EventValue = "=user.name",
            BubbleEvent = true,
        },
        new SendActivity("Your name is ${user.name}"),
        new SendActivity("And you are ${$userType}")
    }
},
new OnDialogEvent()
{
    Event = "contoso.custom",

    // You can use conditions (expression) to examine value of the event as part of the trigger
selection process.
    Condition = "turn.dialogEvent.value && (substring(turn.dialogEvent.value, 0, 1) == 'v')",
    Actions = new List<Dialog>()
    {
        new SendActivity("In custom event: '${turn.dialogEvent.name}' with the following value
`${turn.dialogEvent.value}`"),
        new SetProperty()
        {
            Property = "$userType",
            Value = "VIP"
        }
    }
},
new OnDialogEvent()
{
    Event = "contoso.custom",

    // You can use conditions (expression) to examine value of the event as part of the trigger
selection process.
    Condition = "turn.dialogEvent.value && (substring(turn.dialogEvent.value, 0, 1) == 's')",
    Actions = new List<Dialog>()
    {
        new SendActivity("In custom event: '${turn.dialogEvent.name}' with the following value
`${turn.dialogEvent.value}`"),
        new SetProperty()
        {
            Property = "$userType",
            Value = "Special"
        }
    }
},
new OnDialogEvent()
{
    Event = "contoso.custom",
    Actions = new List<Dialog>()
    {
        new SendActivity("In custom event: '${turn.dialogEvent.name}' with the following value
`${turn.dialogEvent.value}`"),
        new SetProperty()
        {
            Property = "$userType",
            Value = "regular customer"
        }
    }
}
};


```

#### Выход пользователя

Выполняет выход текущего пользователя из системы с помощью [входа через OAuth](#).

```
new SignOutUser()
{
    UserId = "userid",
    ConnectionName = "connection-name"
}
```

#### CodeAction

Как можно предположить по названию, это действие позволяет выполнить пользовательский фрагмент кода.

```
// Example customCodeStep method
private async Task<DialogTurnResult> CodeActionSampleFn(DialogContext dc, System.Object options)
{
    await dc.Context.SendActivityAsync(MessageFactory.Text("In custom code step"));
    // This code step decided to just return the input payload as the result.
    return dc.EndDialogAsync(options)
}

// Adaptive dialog that calls a code step.
var rootDialog = new AdaptiveDialog(rootDialogName) {
    Triggers = new List<OnCondition>()
    {
        new OnUnknownIntent()
        {
            Actions = new List<Dialog>()
            {
                new CodeAction(CodeActionSampleFn),
                new SendActivity("After code step")
            }
        }
    }
};
```

#### Примеры для механизмов отладки

##### TraceActivity

Отправляет действие трассировки с указанными полезными данными.

##### NOTE

Действия трассировки можно сохранять в виде расшифровок и отправлять их в эмулятор для целей отладки.

```
new TraceActivity()
{
    // Name of the trace event.
    Name = "contoso.TraceActivity",
    ValueType = "Object",
    // Property from memory to include in the trace
    ValueProperty = "user"
}
```

##### Сохранение действия

Записывает сообщение в консоль и (необязательно) отправляет его как действие трассировки.

```
new LogAction()
{
    Text = new TextTemplate("Hello"),
    // Automatically sends the provided text as a trace activity
    TraceActivity = true
}
```

## Дополнительные сведения

- Сведения о действиях, связанных со сбором данных от пользователя, см. в статье [Запрос ввода данных пользователем с помощью адаптивных диалогов](#).
- Дополнительные сведения об адаптивных выражениях см. [в этой статье](#).

# Входные данные в адаптивных диалоговых окнах

## — справочное руководство

27.03.2021 • 25 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В пакете SDK для Bot Framework определены разнообразные диалоги ввода для сбора и проверки входных данных, вводимых пользователем.

ТИП ВХОДНЫХ ДАННЫХ	КЛАСС ВХОДНЫХ ДАННЫХ	ОПИСАНИЕ	РЕЗУЛЬТАТЫ
Базовый класс	<a href="#">InputDialog</a>	Это базовый класс, от которого наследуются все классы входных данных. Он определяет все общие свойства.	Объект.
текст	<a href="#">TextInput</a>	Используется, чтобы попросить пользователей указать <b>слово</b> или <b>предложение</b> .	Строка.
Number	<a href="#">NumberInput</a>	Используется, чтобы предложить пользователю ввести <b>число</b> .	Числовое значение.
Подтверждение	<a href="#">ConfirmInput</a>	Используется для запроса <b>подтверждения</b> у пользователя.	Значение типа Boolean.
Несколько вариантов	<a href="#">ChoiceInput</a>	Используется для запроса выбора из <b>набора параметров</b> .	Значение или индекс выбранных данных.
Файл или вложение	<a href="#">AttachmentInput</a>	Используется, чтобы предложить или позволить пользователю <b>передать файл</b> .	Коллекция объектов вложения.
Дата или время	<a href="#">DateTimeInput</a>	Используется, чтобы предложить пользователю ввести <b>дату или время</b> .	Коллекция объектов даты и времени.
Имя для входа OAuth	<a href="#">OAuthInput</a>	Используется для того, чтобы пользователи могли <b>входить на защищенный сайт</b> .	Ответ маркера.

## InputDialog

Классы входных данных, предоставляемые пакетом SDK для Bot Framework, являются производными от

базового *диалога ввода*, который является производным от класса *dialog*. Все диалоги ввода имеют следующие общие свойства.

### AllowInterruptions

Логическое выражение. Значение `true` разрешает родительскому диалогу прервать диалог ввода; в противном случае используется значение `false`.

Сведения о прерываниях см. в статье понятие [обработка прерываний в адаптивных диалоговых окнах](#).

#### NOTE

Родительский диалог ввода также может быть прерван. Это означает, что если `AllowInterruptions` имеет значение `true`, то будет запущен распознаватель в родительском адаптивном диалоге ввода и будут вычислены его триггеры.

### AlwaysPrompt

Логическое выражение. Если имеет значение `true`, всегда запрашиваются входные данные. Если имеет значение `false`, то запрос выводится только в том случае, если привязанное [свойство](#) равно NULL или пустое.

### DefaultValue

Адаптивное выражение, представляющее результат по умолчанию для диалога ввода. Если ввод данных пользователем завершается неудачей [максимальное число раз](#), диалог ввода завершается и для этого свойства устанавливается значение по умолчанию.

```
DefaultValue = "9"
```

### DefaultValueResponse

Ответ, отправляемый, когда все [проверки](#) (`MaxTurnCount`) при входных данных пользователя завершаются неудачей и задается значение [DefaultValue](#).

```
DefaultValueResponse = new ActivityTemplate("Sorry, we have reach the maximum number of attempts of  
'${%MaxTurnCount}' to get your input, so for now, we will go with a default value of: '${%DefaultValue}'")
```

### InvalidPrompt

Шаблон действия, с помощью которого выполняется повторный запрос на ввод, если введенные пользователем данные распознаны, но не прошли проверку. (Если входные данные не прошли проверку [максимальное число раз](#), то используется [значение по умолчанию](#) и отправляется [ответ со значением по умолчанию](#).)

#### NOTE

Свойство `InvalidPrompt` работает только в сочетании со свойством [Validations](#).

```
InvalidPrompt = new ActivityTemplate("Sorry, {this.value} does not work. Please enter a number between one  
and ten (1-10).")
```

### MaxTurnCount

Целочисленное выражение. Максимальное число попыток ввода. Если это ограничение превышено, используется [значение по умолчанию](#) и отправляется [ответ со значением по умолчанию](#).

```
MaxTurnCount = 2
```

## **prompt**

Шаблон действия, с помощью которого изначально запрашиваются входные данные пользователя.

```
Prompt = new ActivityTemplate("Hi, What is your name?")
```

## **Свойство**

Путь в памяти или выражение, результатом вычисления которого является путь в памяти к свойству, к которому привязывается диалог ввода. Путь в памяти будет использоваться для получения начального значения диалога ввода. Он также будет использоваться для сохранения результатов этого диалога. Свойства `Prompt` и `Value` участвуют в распознавании и проверке, поэтому недопустимое начальное значение приведет к появлению запроса.

Используйте это, чтобы определить, к какому свойству привязан диалог ввода. Пример:

```
Property = "user.name"
```

## **UnrecognizedPrompt**

Шаблон действия, с помощью которого выполняется повторный запрос на ввод, если введенные пользователем данные не распознаны. (Если входные данные не прошли проверку [максимальное число раз](#), то используется [значение по умолчанию](#) и отправляется [ответ со значением по умолчанию](#).)

```
UnrecognizedPrompt = new ActivityTemplate("Sorry, '{turn.activity.text}' did not include a valid number")
```

## **Validations (Проверки)**

Список логических выражений. Распознанные входные данные недопустимы, если результатом вычисления любого из этих выражений является `false`. `this.value` Для проверки вводимых пользователем данных в выражениях проверки можно использовать. Проверки реализуются с помощью [адаптивных выражений](#).

## **Значение**

Строковое выражение. Путь в памяти к свойству для получения входных данных из каждого этапа. Это свойство будет использоваться в качестве начального значения для диалога ввода, если результатом вычисления свойства `Property` диалога является значение NULL или пустое значение. Если результатом вычисления свойств `Property` и `Value` диалога является значение NULL или пустое значение, то этот диалог запрашивает входные данные.

Вот то следует учитывать при работе со свойством `Value`.

- Свойство `Value` является [адаптивным выражением](#).
- Если выражение возвращает значение NULL, то диалог ввода может попытаться извлечь данные непосредственно из входных данных.
- Если выражение возвращает значение, то в качестве входных данных будет использоваться это значение.
- Свойство `Value` позволяет определить, как данные, например результаты [распознавателя](#), привязываются к диалогу ввода.

Примеры:

- Привязка входных данных к любой сущности age, распознаваемой во входных данных: "=@age"

- Использование @age или @number в качестве входных данных: "=coalesce(@age, @number)"

### TIP

Вы можете ознакомиться с примером, использующим эти свойства `InputDialog`, в примере кода в разделе [NumberInput](#) ниже.

## TextInput

Используйте *текстовые входные данные*, если хотите, чтобы речь пользователя принималась в качестве значения определенного фрагмента информации, которую пытается собирать бот. В качестве примеров можно привести *имя пользователя* и *тему электронного письма*.

Действие `TextInput` наследует все свойства, определенные в `InputDialog`, и определяет еще одно дополнительное свойство.

- `OutputFormat`: с помощью [адаптивных выражений](#) можно изменить строку. Например, в примере кода ниже выражение `OutputFormat` будет заменять первую букву каждого слова в имени пользователя на прописную.

### Пример TextInput

```
// Create root dialog as an adaptive dialog.
var getUserNameDialog = new AdaptiveDialog("GetUserNameDialog");

// Add an intent trigger.
getUserNameDialog.Triggers.Add(new OnIntent()
{
    Intent = "GetName",
    Actions = new List<Dialog>()
    {
        // Add TextInput step. This step will capture user's input and use it to populate the 'user.name' property.
        new TextInput()
        {
            Property = "user.fullName",
            Prompt = new ActivityTemplate("Please enter your full name.")
            OutputFormat = "join(foreach(split(this.value, ' '), item, concat(toUpper(substring(item, 0, 1)), substring(item, 1))))", ' '
        }
    }
});
```

## NumberInput

Предлагает пользователю ввести число.

Действие `NumberInput` наследует все свойства, определенные в `InputDialog`, и определяет еще два дополнительных свойства.

- `DefaultLocale`: задает языковой стандарт по умолчанию для обработки входных данных, который будет использоваться, если он не указан вызывающим объектом. Поддерживаемые языковые стандарты: испанский, голландский, английский, французский, немецкий, японский, португальский и китайский.
- `OutputFormat`: С помощью [адаптивных выражений](#) можно выполнять действия, чтобы оперировать числами. Например, можно написать выражение для преобразования введенного значения температуры по Фаренгейту в эквивалентное значение в градусах Цельсия. Можно выполнить

математические вычисления, например, чтобы добавить к введенному значению сумму налога и стоимость доставки, или просто выполнить преобразование типа, чтобы указать, что значение является числом с плавающей запятой либо целым числом, как показано в примере кода ниже.

## Пример NumberInput

```
var rootDialog = new AdaptiveDialog(nameof(AdaptiveDialog))
{
    Generator = new TemplateEngineLanguageGenerator(),
    Triggers = new List<OnCondition> ()
    {
        new OnUnknownIntent()
        {
            Actions = new List<Dialog>()
            {
                new NumberInput() {
                    Property = "user.favoriteNumber",
                    Prompt = new ActivityTemplate("Give me your favorite number (1-10)"),
                    // You can refer to incoming user message via turn.activity.text
                    UnrecognizedPrompt = new ActivityTemplate("Sorry, '{turn.activity.text}' did not include
a valid number"),
                    // You can provide a list of validation expressions. Use turn.value to refer to any
                    value extracted by the recognizer.
                    Validations = new List<BoolExpression> () {
                        "int(this.value) >= 1",
                        "int(this.value) <= 10"
                    },
                    InvalidPrompt = new ActivityTemplate("Sorry, {this.value} does not work. Can you give me
a different number that is between 1-10?"),
                    MaxTurnCount = 2,
                    DefaultValue = "9",
                    DefaultValueResponse = new ActivityTemplate("Sorry, we have tried for '${%MaxTurnCount}'"
number of times and I'm still not getting it. For now, I'm setting '${%property}' to '${%DefaultValue}'"),
                    AllowInterruptions = "false",
                    AlwaysPrompt = true,
                    OutputFormat = "float(this.value)"
                },
                new SendActivity("Your favorite number is ${user.favoriteNumber}")
            }
        }
    };
}
```

## ConfirmInput

**Входные данные подтверждения** удобно использовать после того, как вы уже задали пользователю вопрос и хотите подтвердить его ответ. В отличие от действия с **несколькоими вариантами**, позволяющего боту предоставить пользователю список для выбора, появится запрос подтверждения, предлагающий пользователю принять бинарное решение ("Да" или "Нет").

Действие `ConfirmInput` наследует все свойства, определенные в `InputDialog`, и определяет приведенные ниже дополнительные свойства.

- `choiceOptions` : используется для форматирования представления параметров подтверждения, предлагаемых пользователю. Это **адаптивное выражение**, результатом вычисления которого является объект `ChoiceSet`. Этот объект `ChoiceSet` будет использоваться в качестве резервного, только если первоначальная попытка распознавания `ConfirmInput` завершится ошибкой. Когда выполняется действие `confirmInput`, оно сначала пытается вычислить входные данные как логическое значение. В случае сбоя происходит вторая попытка, на этот раз с помощью распознавателя выбора, выполняющего вычисление на основе `ChoiceSet`.
- `confirmChoices` : варианты выбора или **адаптивное выражение**, результатом вычисления которого

являются варианты, представляемые пользователю.

3. `DefaultLocale` : задает языковой стандарт по умолчанию для обработки входных данных, который будет использоваться, если он не указан вызывающим объектом. Поддерживаемые языковые стандарты: испанский, голландский, английский, французский, немецкий, японский, португальский и китайский.
4. `outputFormat` : форматом выходных данных по умолчанию для действия `ConfirmInput` является логическое значение. Это можно переопределить с помощью свойства `OutputFormat` для **адаптивных выражений**, которое при необходимости можно использовать для изменения возвращаемых результатов. Например, вот как можно сделать так, чтобы действие `ConfirmInput` возвращало число: `OutputFormat = "if(this.value == true, 1, 0)"`. Если это свойство задано, то выходными данными выражения является значение, возвращаемое диалогом.
5. `style` : определяет тип списка, который будет представлен пользователю при подтверждении его входных данных. При этом используется перечисление `ListStyle`, которое содержит следующие элементы.
  - a. `None` : не включать параметры запроса.
  - b. `Auto` : автоматически выбрать соответствующий стиль для текущего канала.
  - c. `Inline` : добавить варианты выбора в запрос в виде встроенного списка.
  - d. `List` : добавить варианты выбора в запрос в виде нумерованного списка.
  - e. `SuggestedAction` : добавить варианты выбора в запрос в качестве предлагаемых действий.
  - f. `HeroCard` : добавить варианты выбора в запрос в качестве элемента HeroCard с кнопками.

## Пример `ConfirmInput`

```
// Create adaptive dialog.
var ConfirmationDialog = new AdaptiveDialog("ConfirmationDialog") {
    Triggers = new List<OnCondition>()
    {
        new OnUnknownIntent()
        {
            Actions = new List<Dialog>()
            {
                // Add confirmation input.
                new ConfirmInput()
                {
                    Property = "turn.contoso.travelBot.confirmOutcome",
                    // Since this prompt is built as a generic confirmation wrapper, the actual prompt
                    // text is read from a specific memory location. The caller of this dialog needs to
                    // set the prompt string to that location before calling the "ConfirmationDialog".
                    // All prompts support rich language generation based resolution for output generation.
                    // See https://docs.microsoft.com/azure/bot-service/file-format/bot-builder-lg-file-
format to learn more about the LG
                    // template format used in the ActivityTemplate object.
                    Prompt = new ActivityTemplate("${turn.contoso.travelBot.confirmPromptMessage}")
                }
            }
        }
    }
};
```

## ChoiceInput

**Входные данные выбора** — это набор параметров, предлагаемых пользователю в качестве **нескольких вариантов выбора**. Это дает возможность предоставлять пользователям список вариантов для выбора.

Действие `ChoiceInput` наследует все свойства, определенные в `InputDialog`, и определяет приведенные

ниже дополнительные свойства.

1. `choiceOptions` : это свойство используется для форматирования представления параметров подтверждения, предлагаемых пользователю.
2. `choices` : адаптивное выражение, результатом вычисления которого является элемент ChoiceSet, содержащий упорядоченный список вариантов для выбора пользователем.
3. `DefaultLocale` : задает языковой стандарт по умолчанию для обработки входных данных, который будет использоваться, если он не указан вызывающим объектом. Поддерживаемые языковые стандарты: испанский, голландский, английский, французский, немецкий, японский, португальский и китайский.
4. `outputFormat` : адаптивное выражение, результатом вычисления которого является одно из значений перечисления `ChoiceOutputFormat`.

```
switch (this.OutputFormat.GetValue(dc.State))  
{  
    case ChoiceOutputFormat.Value:  
    default:  
        dc.State.SetValue(VALUE_PROPERTY, foundChoice.Value);  
        break;  
    case ChoiceOutputFormat.Index:  
        dc.State.SetValue(VALUE_PROPERTY, foundChoice.Index);  
        break;  
}
```

5. `style` : определяет тип списка, который будет представлен пользователю при подтверждении его входных данных. При этом используется перечисление `ListStyle`, которое содержит следующие элементы.
  - a. `None` : не включать параметры запроса.
  - b. `Auto` : автоматически выбрать соответствующий стиль для текущего канала.
  - c. `Inline` : добавить варианты выбора в запрос в виде встроенного списка.
  - d. `List` : добавить варианты выбора в запрос в виде нумерованного списка.
  - e. `SuggestedAction` : добавить варианты выбора в запрос в качестве предлагаемых действий.
  - f. `HeroCard` : добавить варианты выбора в запрос в качестве элемента HeroCard с кнопками.
6. `RecognizerOptions` : `FindChoicesOptions` или выражение, результатом которого является `FindChoicesOptions`. `FindChoicesOptions` имеет следующие свойства:
  - a. `NoValue` : Значение типа Boolean. Значение `true` задает поиск свойства *Value* каждого варианта. В противном случае используется значение `false`. Значение по умолчанию — `false`.
  - b. `NoAction` : Значение типа Boolean. Значение `true` задает поиск по заголовку в свойстве *Action* каждого варианта. В противном случае используется значение `false`. Значение по умолчанию — `false`.
  - c. `RecognizeNumbers` : Значение типа Boolean. Значение `true` разрешает альтернативный ввод данных с помощью распознавателя чисел для соответствия вариантам ввода. В противном случае используется значение `false`. Значение по умолчанию — `true`.
  - d. `RecognizeOrdinals` : Значение типа Boolean. Значение `true` разрешает альтернативный ввод данных с помощью распознавателя порядковых чисел для соответствия вариантам ввода. В противном случае используется значение `false`. Значение по умолчанию — `true`.

## Пример ChoiceInput

```

// Create an adaptive dialog.
var getUserFavoriteColor = new AdaptiveDialog(" GetUserColorDialog");
getUserFavoriteColor.Triggers.Add(new OnIntent()
{
    Intent = "GetColor",
    Actions = new List<Dialog>()
    {
        // Add choice input.
        new ChoiceInput()
        {
            // Output from the user is automatically set to this property
            Property = "user.favColor",

            // List of possible styles supported by choice prompt.
            Style = Bot.Builder.Dialogs.Choices.ListStyle.Auto,
            Prompt = new ActivityTemplate("What is your favorite color?"),
            Choices = new ChoiceSet(new List<Choice>())
            {
                new Choice("Red"),
                new Choice("Blue"),
                new Choice("Green")
            })
        }
    }
});
```

## DateTextInput

Предложение ввести дату и (или) время.

Действие `DateTimeInput` наследует все свойства, определенные в `InputDialog`, и определяет приведенные ниже дополнительные свойства.

1. `DefaultLocale` : задает языковой стандарт по умолчанию для обработки входных данных, который будет использоваться, если он не указан вызывающим объектом. Поддерживаемые языковые стандарты: испанский, голландский, английский, французский, немецкий, японский, португальский и китайский.
2. `outputFormat` : по умолчанию выходными данными для `DateTimeInput` является массив `DateTimeResolutions`. Это свойство позволяет определить адаптивное выражение. Любое возвращаемое им значение становится окончательным значением свойства `property` диалога независимо от того, получено оно в результате вычисления даты и времени или нет.

### Пример DateTextInput

```

var rootDialog = new AdaptiveDialog(nameof(AdaptiveDialog))
{
    Generator = new TemplateEngineLanguageGenerator(_templateEngine),
    Triggers = new List<OnCondition>()
    {
        new OnUnknownIntent()
        {
            Actions = new List<Dialog>()
            {
                new DateTimeInput()
                {
                    Property = "$userData",
                    Prompt = new ActivityTemplate("Give me a date"),
                },
                new SendActivity("You gave me ${$userData}")
            }
        }
    }
};


```

## AttachmentInput

Используется, чтобы предложить пользователю вложить файл.

Действие `AttachmentInput` наследует все свойства, определенные в `InputDialog`, и определяет приведенное ниже дополнительное свойство.

- `OutputFormat` : `AttachmentOutputFormat` ИЛИ выражение, результатом которого является `AttachmentOutputFormat`. Допустимые значения `AttachmentOutputFormat` :
  1. `All` : возвращение всех вложений в виде списка;
  2. `First` : возвращение только первого вложения.

### Пример AttachmentInput

```

var rootDialog = new AdaptiveDialog(nameof(AdaptiveDialog))
{
    Generator = new TemplateEngineLanguageGenerator(_templateEngine),
    Triggers = new List<OnCondition>()
    {
        new OnUnknownIntent()
        {
            Actions = new List<Dialog>()
            {
                new AttachmentInput()
                {
                    Property = "$userAttachmentCarImage",
                    Prompt = new ActivityTemplate("Please give me an image of your car. Drag drop the image to the chat canvas."),
                    OutputFormat = AttachmentOutputFormat.All
                },
                new SendActivity("You gave me ${$userAttachmentCarImage}")
            }
        }
    }
};


```

## OAuthInput

Используется, чтобы предложить пользователю выполнить вход.

Действие `OAuthInput` наследует все свойства, определенные в `InputDialog`, и определяет приведенные

ниже дополнительные свойства.

1. `connectionName` : Имя подключения OAuth, настроенного для бота на странице параметров Службы Azure Bot.
2. `text` : дополнительный текст, отображаемый на карте входа.
3. `title` : текст заголовка, отображаемого на карте входа.
4. `Timeout` : число миллисекунд, в течение которых `OAuthInput` ожидает завершения аутентификации пользователя. Значение по умолчанию — 900 000 миллисекунд, то есть 15 минут.

Действие `OAuthInput` также определяет два новых метода:

1. `GetUserTokenAsync` : этот метод пытается получить маркер пользователя.
2. `SignOutUserAsync` : этот метод выполняет выход пользователя.

`OAuthInput` Действие возвращает объект, `TokenResponse` который содержит значения для `ChannelId`, `ConnectionName`, `Token`, `Expiration`. В приведенном ниже примере возвращаемое значение помещается в `turn` область памяти: `turn.oauth`. Доступ к значениям можно получить, как показано в `LoginSteps()` МЕТОДЕ: `new SendActivity("Here is your token '${turn.oauth.token}'").`.

### Пример Оаусинпут

В этом примере `TokenResponse` объект, возвращаемый `OAuthInput` действием, сохраняется в `MyOAuthInput` переменной. Это позволит вам:

- Вызовите диалоговое окно Оаусинпут при любом включении, в котором Bot потребует маркер.
- Настройте Оаусинпут, чтобы записать ответ маркера в область действия "включить память".
- Прочитайте ответ маркера из области "Включение памяти" и используйте его в соответствии с API, с которым он используется. Например, вы добавляете его в качестве токена носителя для API Graph.

```
public class RootDialog : AdaptiveDialog
{
    this.configuration = configuration;
    _templates = Templates.ParseFile(Path.Combine(".", "Dialogs", "RootDialog", "RootDialog.lg"));
    private OAuthInput MyOAuthInput { get; }

    public RootDialog(IConfiguration configuration) : base(nameof(RootDialog))
    {
        Recognizer = CreateLuisRecognizer(this.configuration),
        Generator = new TemplateEngineLanguageGenerator(_templates);

        MyOAuthInput = new OAuthInput
        {
            // The name of the connection configured on Azure Bot Service for the OAuth connection.
            ConnectionName = configuration["ConnectionName"],

            // The title of the sign in card.
            Title = "Please log in",

            // The text displayed in sign in card.
            Text = "This will give you access!",

            // Title of the sign in card.
            InvalidPrompt = new ActivityTemplate("Login was not successful please try again."),

            // The number of milliseconds the prompt waits for the user to authenticate.
            // Tip: For an easy way to set the timeout to a specific number of minutes,
            // you can multiple the number of minutes by 60,000. 5 * 60000 = 5 minutes.
            Timeout = 5 * 60000,

            // The maximum number of times to ask the user for this value before the dialog gives up.
            MaxTurnCount = 3,
        };
    }
}
```

```

        // Property path to store the value (a TokenResponse object) that is returned by the OAuthInput
action.
        // Since the token can be short-lived, you should call the OAuthInput on any turn in which your
bot
        // needs to access associated resources on behalf of the user. If the token is still valid, the
sign-in
        // card will not be displayed, if it is not still active the user will be prompted to sign in
again.
        Property = "turn.oauth",
    };
    // Save the MyOAuthInput dialog instance in the adaptive dialog's dialog set.
    // This will enable consultation, logging telemetry data etc.
    Dialogs.Add(MyOAuthInput);

    // These steps are executed when this Adaptive Dialog begins
    Triggers = new List<OnCondition>
    {
        // Add a rule to welcome user
        new OnConversationUpdateActivity
        {
            Actions = WelcomeUserSteps(),
        },

        // Respond to user on message activity
        new OnUnknownIntent
        {
            Actions = LoginSteps(),
        },

        // Allow the user to sign out.
        new OnIntent("logout")
        {
            Actions =
            {
                new CodeAction(async (dc, opt) =>
                {
                    await MyOAuthInput.SignOutUserAsync(dc);
                    return new DialogTurnResult(DialogTurnStatus.Complete);
                }),
            }
        },
    };
}

private static List<Dialog> WelcomeUserSteps()
{
    return new List<Dialog>
    {
        // Iterate through membersAdded list and greet user added to the conversation.
        new Foreach()
        {
            ItemsProperty = "turn.activity.membersAdded",
            Actions =
            {
                // Note: Some channels send two conversation update events - one for the Bot added to
the conversation and another for user.
                // Filter cases where the bot itself is the recipient of the message.
                new IfCondition()
                {
                    Condition = "$foreach.value.name != turn.activity.recipient.name",
                    Actions =
                    {
                        new SendActivity("Hello, I'm the multi-turn prompt bot. Please send a message to
get started!")
                    }
                }
            }
        }
    };
}

```

```

        }

    private List<Dialog> LoginSteps()
    {
        return new List<Dialog>
        {
            MyOAuthInput,
            new IfCondition
            {
                Condition = "turn.oauth.token && length(turn.oauth.token)",
                Actions = LoginSuccessSteps(),
                ElseActions =
                {
                    new SendActivity("Sorry, we were unable to log you in."),
                },
            },
            new EndDialog(),
        };
    }

    private List<Dialog> LoginSuccessSteps()
    {
        return new List<Dialog>
        {
            new SendActivity("You are now logged in."),
            new ConfirmInput
            {
                Prompt = new ActivityTemplate("Would you like to view your token?"),
                InvalidPrompt = new ActivityTemplate("Oops, I didn't understand. Would you like to view your token?"),
                MaxTurnCount = 3,
            },
            new IfCondition
            {
                Condition = "turn.lastResult == true",
                ElseActions =
                {
                    new SendActivity("Great. Type anything to continue."),
                },
                Actions =
                {
                    MyOAuthInput,
                    new SendActivity("Here is your token `${turn.oauth.token}`."),
                },
            },
        };
    }
}

```

## **Дополнительные сведения, связанные с OAuth**

Ниже приведены ссылки на обобщенные сведения об аутентификации в пакете SDK для Microsoft Bot Framework. Эти сведения относятся к адаптивным диалогам в целом.

- [Аутентификация бота](#)
- [Добавление аутентификации в веб-приложение](#)

# Распознаватели в адаптивных диалоговых окнах

## — справочное руководство

27.03.2021 • 12 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Распознаватели позволяют вашему роботу понять вводимые пользователем данные, и в адаптивном диалоговом окне может быть настроен один или несколько распознавателей. Дополнительные сведения о распознавателях см. в статье [распознаватели в адаптивных диалоговых окнах](#).

## RegexRecognizer

*RegexRecognizer* дает возможность извлекать данные намерений и сущностей из речевого фрагмента на основе шаблонов регулярных выражений.

`RegexRecognizer` состоит в основном из:

- `Intents`. Объект `Intents` содержит список объектов `IntentPattern`, а эти объекты `IntentPattern` содержат свойство `Intent`, которое является именем намерения, и свойство `Pattern`, содержащее регулярное выражение, используемое для анализа речевого фрагмента для распознания намерения.
- `Entities`. Каждый объект `Entities` содержит список объектов `EntityRecognizer`. В пакете SDK для Bot Framework определено несколько классов `EntityRecognizer`, которые помогут определять сущности, содержащиеся в речевых фрагментах пользователей.
  - `AgeEntityRecognizer`
  - `ConfirmationEntityRecognizer`
  - `CurrencyEntityRecognizer`
  - `DateTimeEntityRecognizer`
  - `DimensionEntityRecognizer`
  - `EmailEntityRecognizer`
  - `EntityRecognizer`
  - `EntityRecognizerSet`
  - `GuidEntityRecognizer`
  - `HashtagEntityRecognizer`
  - `IpEntityRecognizer`
  - `MentionEntityRecognizer`
  - `NumberEntityRecognizer`
  - `NumberRangeEntityRecognizer`
  - `OrdinalEntityRecognizer`
  - `PercentageEntityRecognizer`
  - `PhoneNumberEntityRecognizer`
  - `RegExEntityRecognizer`
  - `TemperatureEntityRecognizer`
  - `TextEntity`
  - `TextEntityRecognizer`
  - `UrlEntityRecognizer`

## Пример кода RegexRecognizer

```
var rootDialog = new AdaptiveDialog("rootDialog")
{
    Recognizer = new RegexRecognizer()
    {
        Intents = new List<IntentPattern>()
        {
            new IntentPattern()
            {
                Intent = "AddIntent",
                Pattern = "(?i)(?:add|create) .*(?:to-do|todo|task)(?: )?(?:named (?<title>.*))?"
            },
            new IntentPattern()
            {
                Intent = "HelpIntent",
                Pattern = "(?i)help"
            },
            new IntentPattern()
            {
                Intent = "CancelIntent",
                Pattern = "(?i)cancel|never mind"
            }
        },
        Entities = new List<EntityRecognizer>()
        {
            new ConfirmationEntityRecognizer(),
            new DateTimeEntityRecognizer(),
            new NumberEntityRecognizer()
        }
    }
}
```

### TIP

- `RegexRecognizer` выдаст намерение `None`, если входной речевой фрагмент не соответствует какому-либо определенному намерению. Для реализации этого сценария можно создать триггер `OnIntent` с `Intent = "None"`.
- `RegexRecognizer` удобно использовать для тестирования и быстрого создания прототипов. Для более сложных ботов рекомендуется использовать распознаватель LUIS.
- Вы можете найти полезным [краткий справочник](#) по языку регулярных выражений (RegEx).

## Распознаватель LUIS

Интеллектуальная служба распознавания речи (LUIS) — это облачная служба API, которая применяет пользовательскую аналитику машинного обучения к тексту пользователя в разговорном стиле и на естественном языке, чтобы предсказать общий смысл и извлечь соответствующую подробную информацию. Распознаватель LUIS позволяет извлекать намерения и сущности из речевых фрагментов пользователей с помощью определенного приложения LUIS, которое было предварительно обучено.

Вот как можно создать распознаватель LUIS.

```
var rootDialog = new AdaptiveDialog("rootDialog")
{
    Recognizer = new LuisAdaptiveRecognizer()
    {
        ApplicationId = "<LUIS-APP-ID>",
        EndpointKey = "<ENDPOINT-KEY>",
        Endpoint = "<ENDPOINT-URI>"
    }
}
```

#### TIP

Приведенные ниже сведения помогут узнать больше о том, как внедрить распознавание речи в бот с помощью LUIS.

- [LUIS.ai](#) — это служба на основе машинного обучения, которая позволяет внедрить функции естественного языка в бот.
- [Что такое LUIS?](#)
- [Создание приложения LUIS на портале LUIS](#)
- [Распознавание речи](#)
- [.lu File Format \(Формат файлов LU\)](#)
- [Адаптивные выражения](#)

## Распознаватель QnA Maker

[QnAMaker.ai](#) — это одна из служб [Microsoft Cognitive Services](#), которая позволяет создавать пары "вопрос-ответ" на основе существующего содержимого — документов, URL-адресов, документов в формате PDF и т. д. Для интеграции с этой службой можно использовать распознаватель QnA Maker.

#### NOTE

Распознаватель QnA Maker распознаватель выдает событие `QnAMatch`, которое можно обработать с помощью триггера `OnQnAMatch`. Весь ответ QnA Maker будет доступен в свойстве `answer`.

```

var adaptiveDialog = new AdaptiveDialog()
{
    var recognizer = new QnAMakerRecognizer()
    {
        HostName = configuration["qna:hostname"],
        EndpointKey = configuration["qna:endpointKey"],
        KnowledgeBaseId = configuration["qna:KnowledgeBaseId"],
    }

    Triggers = new List<OnCondition>()
    {
        new OnConversationUpdateActivity()
        {
            Actions = WelcomeUserAction()
        },
    }

    // With QnA Maker set as a recognizer on a dialog, you can use the OnQnAMatch trigger to render the
    answer.
    new OnQnAMatch()
    {
        Actions = new List<Dialog>()
        {
            new SendActivity()
            {
                Activity = new ActivityTemplate("Here's what I have from QnA Maker - ${@answer}"),
            }
        }
    }
}

// Add adaptiveDialog to the DialogSet.
AddDialog(adaptiveDialog);
};

```

## Многоязычный распознаватель

При создании сложного многоязычного бота, как правило, один распознаватель привязывается к конкретному языку и языковому стандарту. Многоязычный распознаватель позволяет легко указать распознаватель для использования на основе свойства [locale](#) входящего действия пользователя.

```

var rootDialog = new AdaptiveDialog("rootDialog")
{
    Recognizer = new MultiLanguageRecognizer()
    {
        Recognizers = new Dictionary<string, Recognizer>()
        {
            {
                "en",
                new RegexRecognizer()
                {
                    Intents = new List<IntentPattern>()
                    {
                        new IntentPattern()
                        {
                            Intent = "AddIntent",
                            Pattern = "(?i)(?:add|create) .*(?:to-do|todo|task)(?: )?(?:named (?<title>.*))?"
                        },
                        new IntentPattern()
                        {
                            Intent = "HelpIntent",
                            Pattern = "(?i)help"
                        },
                        new IntentPattern()
                        {
                            Intent = "CancelIntent",
                            Pattern = "(?i)cancel|never mind"
                        }
                    },
                    Entities = new List<EntityRecognizer>()
                    {
                        new ConfirmationEntityRecognizer(),
                        new DateTimeEntityRecognizer(),
                        new NumberEntityRecognizer()
                    }
                }
            },
            {
                "fr",
                new LuisAdaptiveRecognizer()
                {
                    ApplicationId = "<LUIS-APP-ID>",
                    EndpointKey = "<ENDPOINT-KEY>",
                    Endpoint = "<ENDPOINT-URI>"
                }
            }
        };
    };
};

```

## Набор распознавателей

Иногда может потребоваться запускать несколько распознавателей на каждом этапе беседы. Именно для этого предназначен набор распознавателей. Все распознаватели выполняются на каждом этапе диалога, а результат представляет собой объединение всех результатов распознавания.

```

var adaptiveDialog = new AdaptiveDialog()
{
    Recognizer = new RecognizerSet()
    {
        Recognizers = new List<Recognizer>()
        {
            new ValueRecognizer(),
            new QnAMakerRecognizer()
            {
                KnowledgeBaseId = "<KBID>",
                HostName = "<HostName>",
                EndpointKey = "<Key>"
            }
        }
    }
};

```

## Набор взаимно-обучаемых распознавателей

Набор взаимно-обучаемых распознавателей сравнивает результаты распознавания от нескольких распознавателей, чтобы выбрать наилучший из них. Для заданной коллекции распознавателей взаимно-обучаемый распознаватель сделает следующее.

- Повысит приоритет результата распознавания одного из распознавателей, если все другие распознаватели переносят распознавание на отдельный распознаватель. Чтобы перенести распознавание, распознаватель может вернуть намерение `None` или явное намерение `DeferToRecognizer_recognizerId`.
- Создайте `OnChooseIntent` событие, чтобы разрешить коду выбирать, какой результат распознавания следует использовать. Результаты каждого распознавателя возвращаются с помощью свойства `turn.recognized.candidates`. Это позволяет выбрать наиболее подходящий результат.

```

var adaptiveDialog = new AdaptiveDialog()
{
    Recognizer = new CrossTrainedRecognizerSet()
    {
        Recognizers = new List<Recognizer>()
        {
            new LuisAdaptiveRecognizer()
            {
                Id = "Luis-main-dialog",
                ApplicationId = "<LUIS-APP-ID>",
                EndpointKey = "<ENDPOINT-KEY>",
                Endpoint = "<ENDPOINT-URI>"
            },
            new QnAMakerRecognizer()
            {
                Id = "qna-main-dialog",
                KnowledgeBaseId = "<KBID>",
                HostName = "<HostName>",
                EndpointKey = "<Key>"
            }
        }
    }
};

```

## Перекрестное обучение моделей LUIS и QnA

Чтобы получить все преимущества набора распознавателей, выполните [перекрестное обучение](#) файлов. Для этого можно использовать командную строку Bot Framework (BF CLI) для автоматизации этого процесса, команды `Luis: Cross-Training` и `qnamaker: Cross-poезд`. При выполнении

команды перекрестного обучения будут созданы копии файлов. Iu и. QnA, внесены необходимые обновления, а затем сохранен в указанном каталоге.

#### TIP

Чтобы создать файлы для перекрестной обучении, и LU, и QnA, можно использовать интерфейс командной строки BF `luis:cross-train` или `qnamaker:cross-train` команду. Не нужно выполнять обе команды, так как они выполняют одно и то же действие. Ниже показано использование `luis:cross-train` команды.

```
bf luis:cross-train -i <input-folder-name> -o <output-file-name> --config <cross-train-configuration-file>
```

Полный пример перекрестного обучения для Bot см. в статье [Создание программы-робота для использования Luis и QnA Maker распознаватели](#).

#### Luis: обязательные параметры перекрестного обучения

- `--in` — Каталог, включая подкаталоги, в котором будет выполняться поиск файлов. Iu и. QnA.
- `--out` — Каталог, в который будут сохранены новые файлы выходных данных перекрестной обучены. Iu и. QnA. Это каталог, в который будет указываться `luis:build` `--in` параметр команды.
- `--config` — Указывает файл конфигурации перекрестной подготовки — JSON-файл, необходимый для работы команды.

#### Файл конфигурации перекрестного обучения

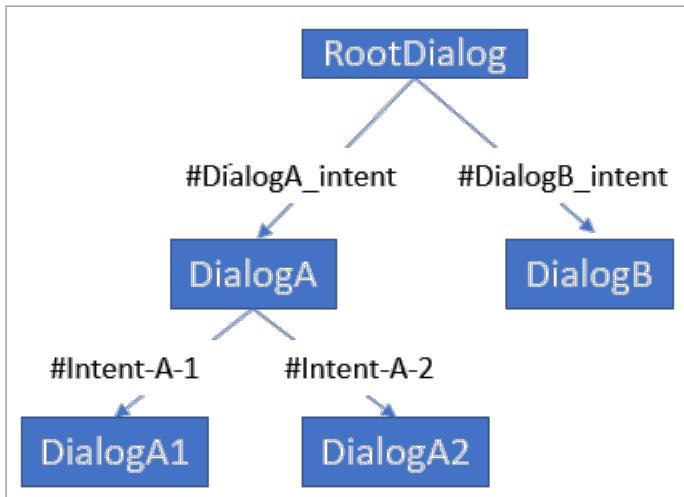
Ниже приведена общая структура файла конфигурации для перекрестного обучения.

```
{
    // list each .lu file including variations per lang x locale.
    // Lang x locale is denoted using 4 letter code. e.g. it-it, fr-fr
    // Paths can either be absolute (full) paths or paths relative to this config file.

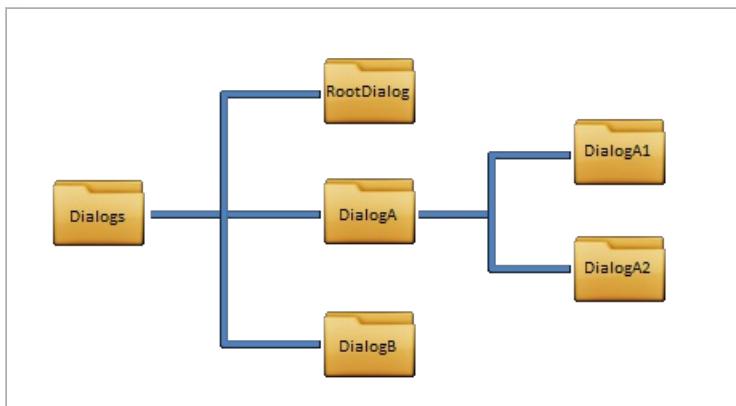
    "<path-of-language-file-to-train>": {
        // indicate if this is an .lu file for the root dialog.
        "rootDialog": <true-or-false>,
        // list of triggers within that dialog
        "triggers": {
            // Key is name of intent within the .lu file (in this case RootDialog.lu)
            // Value is the path to the child dialog's .lu file.
            "<intent-name-1>": "<file-name-with-language-of-associated-child-dialog>",
            "<intent-name-2>": "<file-name-with-language-of-associated-child-dialog>"
            // And so on.
        },
        "<path-of-additional-language-file-to-train>": {
            // indicate if this is an .lu file for the root dialog.
            "rootDialog": <true-or-false>,
            // list of triggers within that dialog
            "triggers": {
                "<intent-name-1>": "<file-name-with-language-of-associated-child-dialog>",
                "<intent-name-2>": "<file-name-with-language-of-associated-child-dialog>"
            }
            // And so on.
        }
    }
}
```

В разделе Triggers файла конфигурации перекрестного обучения перечислите цели в корневом диалоговом окне вместе с Lu-файлом, на который указывает. Необходимо только вывести файлы с расширением. Iu, файлы. QnA будут пересекаться, если они находятся в одном каталоге с тем же именем файла, например *адтододиалог. QnA*.

Например, программа-робот имеет следующую структуру диалогового окна:



Со следующей структурой каталогов:



В каталоге **диалоговых окон** будет создан файл конфигурации, который может выглядеть примерно так:

```

{
    "./rootDialog/rootDialog.lu": {
        "rootDialog": true,
        "triggers": {
            "DialogA_intent": ".DialogA.lu",
            "DialogB_intent": "DialogB.lu"
        }
    },
    "./DialogA/DialogA.lu": {
        "triggers": {
            "DialogA1_intent": "DialogA1.lu",
            "DialogA2_intent": "DialogA2.lu",
            "Intent-A-1": "",
            "Intent-A-2": ""
        }
    },
    "./DialogA/DialogA1/DialogA1.lu": {
        "triggers": {
            "DialogA1_1_intent": "DialogA1.1.lu",
            "DialogA1_2_intent": "DialogA1.2.lu",
        }
    },
    "./DialogB/DialogB.lu": {
        "triggers": {
            "DialogB1_intent": "./DialogB/DialogB1/DialogB1.lu",
        }
    }
}

```

В приведенном выше JSON-файле, если часть значения пары "ключ — значение" пуста, она указывает на намерение, которое не приводит к созданию диалогового окна адаптивного контейнера, а запускает действие, связанное с указанным `OnIntent` триггером.

Если язык включен, пример конфигурации должен выглядеть следующим образом:

```
{  
    "rootDialog.en-us": {  
        "rootDialog": true,  
        "triggers": {  
            "DialogA_intent": "DialogA.en-us",  
            "DialogB_intent": "DialogB.en-us"  
        }  
    },  
    "DialogA.en-us": {  
        "triggers": {  
            "DialogA1_intent": "DialogA1.en-us",  
            "DialogA2_intent": "DialogA2.en-us",  
            "Intent-A-1": "",  
            "Intent-A-2": ""  
        }  
    },  
    "DialogA1.en-us": {  
        "triggers": {  
            "DialogA1.1_intent": "DialogA1.1.en-us",  
            "DialogA1.2_intent": "DialogA1.2.en-us",  
        }  
    },  
    "DialogB.en-us": {  
        "triggers": {  
            "DialogB1_intent": "DialogB1.en-us",  
        }  
    }  
}
```

#### TIP

Если ваш Bot содержит только модели LUIS и не имеет QnA Maker моделей, можно переучить только модели LUIS. Дополнительные сведения о перекрестном обучении моделей LUIS см. в разделе [Luis to Luis Cross Training](#).

## Дополнительные сведения

- [Что такое LUIS?](#)
- [Распознавание речи](#)
- [.lu File Format \(Формат файлов LU\)](#)
- [Адаптивные выражения](#)
- [Извлечение данных из текста речевого фрагмента с помощью намерений и сущностей](#)
- [Добавление возможности распознавания естественного языка в бот](#)
- [Добавление возможности создания естественного языка в бот](#)

# Управление состоянием в адаптивных диалоговых окнах — справочное руководство

27.03.2021 • 8 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Эта статья содержит технические сведения, которые помогут вам работать с областями памяти в адаптивных диалоговых окнах. Общие сведения об областях памяти и управлении состоянием в адаптивных диалоговых окнах см. в статье принцип [управления состоянием в адаптивных диалоговых окнах](#).

## TIP

В путях к свойствам не учитывается регистр. Например, `user.name` — это тоже самое, что и `user.Name`. Кроме того, если у вас нет свойства `user.name` и вы создадите свойство с именем `user.name.first`, то объект `user.name` будет создан автоматически.

## Область пользователей

Область пользователей предназначена для сохраняемых данных, соответствующих идентификатору пользователя, с которым выполняется общение.

Примеры:

- `user.name`
- `user.address.city`

## Область беседы

Область беседы предназначена для сохраняемых данных, соответствующих идентификатору осуществляющей беседы.

Примеры:

- `conversation.hasAccepted`
- `conversation.dateStarted`
- `conversation.lastMaleReference`
- `conversation.lastFemaleReference`
- `conversation.lastLocationReference`

В следующем примере показано, как можно использовать область диалога для получения входных данных от пользователя, создавая новый `PropertyAssignment` объект для использования в `SetProperties` [действии] [SetProperties-Action], получая значение из области диалога.

```
new PropertyAssignment()
{
    Property = "conversation.flightBooking.departureCity",
    Value = "=turn.recognized.entities.fromCity.location"
},
```

## Область диалога

Область диалога предназначена для хранения данных в течение жизненного цикла связанного диалога. Она предоставляет область памяти для внутреннего учета каждого диалога. Область диалога очищается по завершении связанного диалога.

Краткие примеры области диалога.

- Сокращение для `dialog.orderStarted` — `$orderStarted`.
- Сокращение для `dialog.shoppingCart` — `$shoppingCart`.

Все параметры, передаваемые в `beginDialog` при создании адаптивного диалога, становятся свойствами этого диалога и доступны при условии, что они находятся в области. Обращаться к этим свойствам можно по имени: `dialog.<propertyName>`. Например, если вызывающий объект передал `{a: '1', b: '2'}`, то эти параметры будут заданы как свойства `dialog.a` и `dialog.b`.

### Подобласти диалога

Все действия триггера в адаптивном диалоге имеют собственные подобласти, к которым можно обращаться по имени. Например, к действию `Foreach` можно обратиться так: `dialog.Foreach`. По умолчанию индекс и значение задаются в области `dialog.foreach`, доступ к которой можно получить так: `dialog.Foreach.index` И `dialog.Foreach.value`. Дополнительные сведения о действии можно узнать [Foreach](#) в разделе **действия в адаптивных диалоговых окнах**.

## Область этапа

Область этапа содержит *временные данные*, предназначенные только для текущего этапа. Область этапа обеспечивает место для совместного использования данных в течение времени существования текущего этапа.

Примеры:

- `turn.bookingConfirmation`
- `turn.activityProcessed`

### Подобласти этапа

#### `turn.activity`

Каждое входящее действие в боте доступно посредством области `turn.activity`.

Например, ниже приведен пример того, что можно определить в LG-файле, чтобы бот реагировал на некорректный ответ на вопрос о возрасте пользователя.

```
Sorry, I do not understand '${turn.activity.text}'. ${GetAge()}
```

Можно также установить значения свойств в исходном коде диалогов.

```
ItemsProperty = "turn.activity.membersAdded"
```

#### `turn.recognized`

Все намерения и сущности, возвращаемые от [распознавателем](#) на заданном этапе, автоматически передаются в область `turn.recognized` и остаются доступными до следующего этапа. У области `turn.recognized` имеются три свойства:

- `turn.recognized.intents.xxx` : список основных намерений, классифицируемых распознавателем для этого этапа.

- `turn.recognized.entities.xxx` : список сущностей, распознанных на этом этапе;
- `turn.recognized.score` : оценка достоверности наиболее вероятного намерения для этого этапа.

Например, в приложении для бронирования авиабилетов может использоваться намерение `flight` (авиарейс) с сущностями для пунктов отправления и назначения. В приведенном ниже примере показано, как получить значение пункта назначения перед завершением этапа.

```
Value = "=turn.recognized.entities.fromCity.location"
```

Существует другой способ сделать это с помощью [краткой нотации памяти](#).

```
// Value is a property containing an expression. @entityName is shorthand to refer to the value of
// the entity recognized. @fromCity.location is same as turn.recognized.entities.fromCity.location
Value = "@fromCity.location"
```

#### **turn.dialogEvent**

`turn.dialogEvent` содержит полезные данные события, порождаемого системой или кодом. Доступ к сведениям, содержащимся в полезных данных, можно получить, обратившись к области `turn.dialogEvent.<eventName>.value`.

#### **turn.lastResult**

Можно получить доступ к результатам последнего диалога, вызванным из области `turn.lastResult`.

#### **turn.activityProcessed**

`turn.activityProcessed` — логическое свойство; если оно задано, это означает, что `turnContext.activity` используется каким-либо компонентом в системе.

#### **turn.interrupted**

`turn.interrupted` — логическое свойство; значение `true` указывает, что произошло прерывание.

## Область параметров

Это представляет все параметры, доступные для программы-робота через систему настройки конкретных параметров платформы. Например, если вы разрабатываете Бот с помощью C#, эти параметры будут отображаться в `appsettings.jsb` файле..

#### **Пример области параметров.**

Ниже приведен пример файла `appsettings.json`, в котором хранятся параметры конфигурации для бота.

```
{
  "MicrosoftAppId": "<yourMicrosoftAppId>",
  "MicrosoftAppPassword": "<yourMicrosoftAppPassword>",
  "QnAMaker": {
    "knowledgebaseId": "<yourQnAKnowledgebaseId>",
    "hostname": "https://<YourHostName>.azurewebsites.net/qnamaker",
    "endpointKey": "yourEndpointKey"
  }
}
```

Ниже приведен пример того, как можно ссылаться на параметры конфигурации в файле `appsettings.json` с помощью области памяти параметров.

```
var recognizer = new QnAMakerRecognizer()
{
    HostName = settings.QnAMaker.hostname,
    EndpointKey = settings.QnAMaker:endpointKey,
    KnowledgeBaseId = settings.QnAMaker:knowledgebaseId,
};
```

## Эта область

Область `this` относится к контейнеру свойств активного действия. Ее удобно использовать для действий ввода, так как их жизненный цикл обычно превышает один этап беседы.

- `this.value` содержит текущее распознанное значение из входных данных.
- `this.turnCount` содержит количество запросов на ввод отсутствующей информации для данного действия ввода.

В этом примере показано распространенное использование в классе запуска бота.

### Пример такой области:

```
new TextInput()
{
    property = "user.name",
    prompt = new ActivityTemplate("what is your name?"),
    defaultValue = "Human",
    defaultValueResponse = new ActivityTemplate("Sorry, I still am not getting it after ${this.turnCount}
attempts. For now, I'm setting your name to '${class.defaultValue}' for now. You can always say 'My name is
<your name>' to re-introduce yourself to me.")
}
```

## Область класса

Она содержит свойства экземпляра активного диалога. Указать ссылку на эту область можно следующим образом:  `${class.<propertyName>}` .

### Пример области класса.

```
new TextInput()
{
    Property = "user.age"
    Prompt = new ActivityTemplate("What is your age?"),
    DefaultValue = "30",
    DefaultValueResponse = new ActivityTemplate("Sorry, I'm not getting it in spite of you trying
${this.turnCount} number of times. \n I'm going with ${class.defaultValue} for now.")
}
```

## Дополнительные сведения

- Общие сведения об управлении состоянием в адаптивных диалоговых окнах см. в статье [принцип управления состоянием в адаптивных диалоговых окнах](#).
- [Короткая нотация памяти](#).

# Стандартные функции адаптивных выражений

27.03.2021 • 160 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье перечислены доступные стандартные функции с упорядочением по их назначению. Дополнительные сведения об операторах, используемых в предварительно созданных функциях и синтаксисе выражений, см. в разделе [Операторы](#).

Предварительно созданные выражения разделены на приведенные ниже типы.

- [String](#)
- [Коллекция](#)
- [Логическое сравнение](#)
- [Преобразование](#)
- [Math](#)
- [Дата](#)
- [Timex](#)
- [Синтаксический анализ URI](#)
- [Обработка и создание объектов](#)
- [Регулярное выражение](#)
- [Проверка типа](#)

Этот же список можно просмотреть [в алфавитном порядке](#).

## Строковые функции

КОМПОНЕНТ	ОБЪЯСНЕНИЕ
<a href="#">length</a>	Возвращает длину строки.
<a href="#">replace</a>	Заменяет подстроку указанной строкой и возвращает обновленную строку. Эта функция учитывает регистр.
<a href="#">replaceIgnoreCase</a>	Заменяет подстроку указанной строкой и возвращает обновленную строку. Эта функция не учитывает регистр.
<a href="#">split</a>	Возвращает массив подстрок с разбиением по указанному разделителю.
<a href="#">substring</a>	Возвращает символы из строки.
<a href="#">toLowerCase</a>	Возврат строки в нижнем регистре в необязательном формате языкового стандарта.
<a href="#">toUpperCase</a>	Возврат строки в верхнем регистре в необязательном формате языкового стандарта.
<a href="#">trim</a>	Удаляет пробелы в начале и конце строки.

КОМПОНЕНТ	ОБЪЯСНЕНИЕ
addOrdinal	Возвращает порядковый номер полученного числа.
endsWith	Проверяет, заканчивается ли строка определенной подстрокой. Возвращает <code>true</code> , если подстрока найдена, или <code>false</code> в противном случае. Эта функция не учитывает регистр.
startsWith	Проверяет, начинается ли строка с определенной подстроки. Возвращает <code>true</code> , если подстрока найдена, или <code>false</code> в противном случае. Эта функция не учитывает регистр.
countWord	Возвращает количество слов в полученной строке.
concat	Объединяет две или более строк и возвращает объединенную строку.
newGuid	Возвращает строку с новым GUID.
indexOf	Возвращает начальную позицию или значение индекса указанной подстроки <b>или</b> находит указанный объект и возвращает индекс его первого вхождения в коллекцию (отсчет начинается с нуля). Эта функция не учитывает регистр, а значения индексов начинаются с 0.
lastIndexOf	Возвращает начальную позицию или значение индекса последнего вхождения подстроки <b>или</b> находит указанный объект и возвращает индекс его последнего вхождения в диапазоне элементов в списке (отсчет начинается с нуля). Эта функция не учитывает регистр, а значения индексов начинаются с 0.
sentenceCase	Сделать прописной первую букву первого слова в строке в необязательном локальном формате.
: заглавный регистр	Сделать прописной первую букву каждого слова в строке в необязательном формате языкового стандарта.

## ФУНКЦИИ ДЛЯ КОЛЛЕКЦИЙ

КОМПОНЕНТ	ОБЪЯСНЕНИЕ
contains	<p>Пытается найти элемент в строке, элемент в массиве или параметр в составном объекте.</p> <p><b>Примеры:</b></p> <pre>contains('hello world', 'hello') contains(createArray('1','2'), '1') contains(json("{'foo':'bar'}"), 'foo')</pre>
first	Возвращает первый элемент коллекции.

КОМПОНЕНТ	ОБЪЯСНЕНИЕ
join	Возвращает строку, содержащую все элементы из массива, в которой каждый символ отделен разделителем. <b>Пример.</b> join(createArray('a','b'), '.') = "a.b"
last	Возвращает последний элемент коллекции.
count	Возвращает количество элементов в коллекции.
foreach	Обрабатывает каждый элемент и возвращают новую коллекцию.
union	Возвращает коллекцию, которая содержит все элементы из указанных коллекций.
skip	Удаляет элементы из начала коллекции и возвращает оставшиеся элементы.
take	Возвращает элементы, расположенные в начале коллекции.
intersection	Возвращает коллекцию, которая содержит только общие элементы в указанных коллекциях.
subArray	Возвращает подмассив, полученный по указанным начальной и конечной позициям. Значения индекса начинаются с числа 0.
select	Обрабатывает каждый элемент и возвращают новую коллекцию преобразованных элементов.
where	Применяет фильтр к каждому элементу и возвращает новую коллекцию тех элементов, которые соответствуют указанному условию.
sortBy	Сортирует элементы коллекции по возрастанию и возвращает упорядоченную коллекцию.
sortByDescending	Сортирует элементы коллекции по убыванию и возвращает упорядоченную коллекцию.
indicesAndValues	Преобразует массив или объект в массив объектов, состоящих из индексов и значений.
flatten	Преобразует массивы в плоские структуры, у которых значения не являются массивами.
unique	Удаляет из массива все дубликаты.

## ФУНКЦИИ ЛОГИЧЕСКОГО СРАВНЕНИЯ

КОМПОНЕНТ	ОБЪЯСНЕНИЕ
<code>and (и);</code>	Логическое "И". Возвращает значение <code>true</code> , если все указанные выражения имеют значение <code>true</code> .
<code>equals</code>	Сравнение по строгому равенству. Возвращает значение <code>true</code> , если указанные значения равны.
<code>empty</code>	Проверяет, пуст ли целевой объект.
<code>greater</code>	Сравнение "больше чем". Возвращает значение <code>true</code> , если первое значение больше второго, или <code>false</code> в противном случае.
<code>greaterOrEquals</code>	Сравнение "больше или равно". Возвращает <code>true</code> , если первое значение больше второго или равно ему, или <code>false</code> в противном случае.
<code>if (если);</code>	Проверьте, какое значение имеет выражение: <code>true</code> или <code>false</code> . Возвращает указанное значение на основе результата.
<code>less</code>	Оператор сравнения "меньше чем". Возвращает <code>true</code> , если первое значение меньше второго, или <code>false</code> в противном случае.
<code>lessOrEquals</code>	Оператор сравнения "меньше или равно". Возвращает <code>true</code> , если первое значение меньше второго или равно ему, или <code>false</code> в противном случае.
<code>not (не);</code>	Оператор логического отрицания. Возвращает <code>true</code> , если выражение ложно, или <code>false</code> в противном случае.
<code>или диспетчер конфигурации служб</code>	Оператор логического "ИЛИ". Возвращает значение <code>true</code> , если хотя бы одно выражение истинно, или <code>false</code> , если все являются ложными.
<code>exists</code>	Проверяет истинность выражения.

## ФУНКЦИИ ПРЕОБРАЗОВАНИЯ

КОМПОНЕНТ	ОБЪЯСНЕНИЕ
<code>float</code>	Возврат представления с плавающей запятой указанной строки.
<code>int</code>	Возвращает целочисленное представление указанной строки.
<code>строка</code>	Возврат строковой версии указанного значения в необязательном формате языкового стандарта.
<code>bool;</code>	Возвращает указанное значение в логическом формате.

КОМПОНЕНТ	ОБЪЯСНЕНИЕ
createArray	Возвращает массив, полученный на основе нескольких входных данных.
json	Возвращает значение с типом JSON (нотация объектов JavaScript), объект с типом строки или XML.
base64	Возвращает строку или массив байтов в кодировке base64.
base64ToBinary	Возвращает двоичную версию строки с кодировкой base64.
base64ToString	Преобразует строку в кодировке base64 в строковый формат.
binary	Возвращает двоичную версию входного значения.
dataUri	Возвращает URI для входного значения.
dataUriToBinary	Возвращает URI данных в двоичном формате.
dataUriToString	Возвращает URI данных в строковом формате.
uriComponent	Возвращает кодированную версию URI для входного значения, заменив символы, опасные для URL-адреса, на escape-символы.
uriComponentToString	Преобразует строку в формате URI в строковый формат.
xml	Только для C#. Возвращает строку в формате XML.
formatNumber	Форматирование значения до ближайшего числа до указанного числа цифр дробной части и необязательного заданного языкового стандарта.

## Математические функции

КОМПОНЕНТ	ОБЪЯСНЕНИЕ
добавление	Математическое сложение. Возвращает результат сложения двух чисел (вариант только с числами) или объединения двух или более строк.
div	Математическое деление. Возвращает целочисленный результат деления двух чисел.
max	Возвращает наибольшее значение из коллекции.
min	Возвращает наименьший элемент коллекции.
mod (модуль)	Возвращает остаток результата деления двух чисел.

КОМПОНЕНТ	ОБЪЯСНЕНИЕ
mul	Математическое умножение. Возвращает результат умножения двух чисел.
rand	Возвращает случайное число в диапазоне между указанными минимальным и максимальным значениями.
sub	Математическое вычитание. Вычитает второе число из первого числа и возвращает результат.
sum	Возвращает сумму чисел в массиве.
range	Возвращает массив целых чисел, который начинается с заданного целого числа.
exp	Возвращает результат возведения указанного числа в указанную степень.
average	Возвращает среднее значение чисел в числовом массиве.
фабрич	Возврат самого крупного целочисленного значения, которое меньше или равно указанному числу.
толок	Возврат наименьшего целого значения, которое больше или равно указанному числу.
округло	Округляет значение до ближайшего целого или до указанного числа дробных разрядов.

## ФУНКЦИИ ДАТЫ И ВРЕМЕНИ

КОМПОНЕНТ	ОБЪЯСНЕНИЕ
addDays	Добавление числа указанных дней к заданной метке времени в необязательном формате языкового стандарта.
addHours	Добавляет указанное число часов к заданной метке времени в необязательном формате языкового стандарта.
addMinutes	Добавляет указанное число минут к заданной метке времени в необязательном формате языкового стандарта.
addSeconds	Добавляет указанное количество секунд к указанной метке времени.
dayOfMonth	Возвращает день месяца для указанной метки времени или выражения Timex.
dayOfWeek	Возвращает день недели для указанной метки времени.

КОМПОНЕНТ	ОБЪЯСНЕНИЕ
<code>dayOfYear</code>	Возвращает порядковый номер дня в году для указанной метки времени.
<code>formatDateTime</code>	Возврат метки времени в необязательном формате языкового стандарта.
<code>formatEpoch</code>	Возврат метки времени в необязательном формате языкового стандарта от времени для среды UNIX (время UNIX, время POSIX).
<code>formatTicks</code>	Возврат метки времени в необязательном формате языкового стандарта от тактов.
<code>subtractFromTime</code>	Вычтите количество единиц времени из метки времени в необязательном формате языкового стандарта.
<code>utcNow</code>	Возврат текущей метки времени в необязательном формате языкового стандарта в виде строки.
<code>dateReadBack</code>	Обратное чтение даты с помощью библиотеки <code>date-time</code> .
<code>month</code>	Возвращает месяц для указанной метки времени.
<code>date</code>	Возвращает дату для указанной метки времени.
<code>year</code>	Возвращает год для указанной метки времени.
<code>getTimeOfDay</code>	Возвращает время дня для указанной метки времени.
<code>getFutureTime</code>	Возврат текущей метки времени в необязательном формате языкового стандарта и заданных единиц времени.
<code>getPastTime</code>	Возврат текущей метки времени в необязательном формате языкового стандарта за вычетом заданных единиц времени.
<code>addToTime</code>	Добавьте несколько единиц времени в отметку времени в необязательном формате языкового стандарта.
<code>convertFromUTC</code>	Преобразование метки времени в необязательном формате языкового стандарта из универсального скоординированного времени (UTC).
<code>convertToUTC</code>	Преобразование метки времени в необязательном формате языкового стандарта в универсальное скоординированное время (UTC).
<code>startOfDay</code>	Возврат начала дня для метки времени в необязательном формате языкового стандарта.
<code>startOfHour</code>	Возврат начала часа для отметки времени в необязательном формате языкового стандарта.

КОМПОНЕНТ	ОБЪЯСНЕНИЕ
startOfMonth	Возвращает начало месяца для отметки времени в необязательном формате языкового стандарта.
ticks	Возвращает значение свойства количества тактов для указанной метки времени.
тиккстодайс	Преобразует значение свойства тактов в число дней.
тиккстохаурс	Преобразуйте значение свойства тактов в число часов.
тиккстоминутес	Преобразуйте значение свойства тактов в число минут.
датетимедифф	Возвращает разность в тактах между двумя метками времени.
жет превиаусвиабледате	Возвращает предыдущей допустимой даты выражения Timex на основе текущей даты и дополнительно заданного часового пояса.
жет некствиабледате	Возвращает следующей допустимой даты выражения Timex на основе текущей даты и дополнительно заданного часового пояса.
жет превиаусвиаблетиме	Возвращает предыдущее допустимое время выражения Timex на основе текущего времени и дополнительно заданного часового пояса.
жет некствиаблетиме	Возвращает следующее допустимое время выражения Timex на основе текущего времени и дополнительно заданного часового пояса.

## ФУНКЦИИ Timex

КОМПОНЕНТ	ОБЪЯСНЕНИЕ
isPresent	Возвращает значение true, если выражение TimexProperty или Timex указывает текущее время.
isDuration	Возвращает значение true, если полученный объект TimexProperty или выражение Timex указывает на длительность времени.
isTime	Возвращает значение true, если объект TimexProperty или выражение Timex указывает на конкретное время.
isDate	Возвращает значение true, если объект TimexProperty или выражение Timex указывает на конкретную дату.
isTimeRange	Возвращает значение true, если объект TimexProperty или выражение Timex указывает на диапазон времени.
isDateRange	Возвращает значение true, если объект TimexProperty или выражение Timex указывает на диапазон дат.

КОМПОНЕНТ	ОБЪЯСНЕНИЕ
<a href="#">isDefinite</a>	Возвращает значение true, если объект TimexProperty или выражение Timex указывает на определенный день.

## Функции анализа URI

КОМПОНЕНТ	ОБЪЯСНЕНИЕ
<a href="#">uriHost</a>	Возвращает значение узла из исходного значения универсального кода ресурса (URI).
<a href="#">uriPath</a>	Возвращает значение пути из исходного значения URI.
<a href="#">uriPathAndQuery</a>	Возвращает значения пути и запроса из исходного значения URI.
<a href="#">uriPort</a>	Возвращает значение порта из исходного значения URI.
<a href="#">uriQuery</a>	Возвращает значение запроса из исходного значения URI.
<a href="#">uriScheme</a>	Возвращает значение схемы из исходного значения URI.

## Функции обработки и создания объектов

КОМПОНЕНТ	ОБЪЯСНЕНИЕ
<a href="#">addProperty</a>	Добавляет свойство и значение или пару "имя-значение" в объект JSON и возвращает обновленный объект.
<a href="#">removeProperty</a>	Удаляет свойство из объекта JSON и возвращает обновленный объект.
<a href="#">setProperty</a>	Задает новое значение для свойства в объекте JSON и возвращает обновленный объект.
<a href="#">getProperty</a>	Возрат значения указанного свойства или корневого свойства из объекта JSON.
<a href="#">coalesce</a>	Возвращает первое ненулевое значение из одного или нескольких параметров.
<a href="#">xPath</a>	Только для C#. Проверяет XML на наличие узлов или значений, которые соответствуют выражению XPath, и возвращает соответствующие узлы или значения.
<a href="#">jPath</a>	В указанном объекте или строке JSON проверяет наличие узлов или значений, которые соответствуют выражению пути, и возвращает найденные узлы.
<a href="#">setPathToValue</a>	Задает значение конкретного пути и возвращает это значение.

## Функции регулярных выражений

КОМПОНЕНТ	ОБЪЯСНЕНИЕ
isMatch	Возвращает значение true, если строка соответствует общему шаблону регулярного выражения.

## Функции проверки типа

КОМПОНЕНТ	ОБЪЯСНЕНИЕ
КОНЦА строки	Только для C#. Возврат текста последовательности конца строки (конца строки).
isInteger	Возвращает значение true, если входное значение является целым числом.
isFloat	Возвращает значение true, если входное значение является числом с плавающей запятой.
isBoolean	Возвращает значение true, если входное значение является логическим значением.
isArray	Возвращает значение true, если входное значение является массивом.
isObject	Возвращает значение true, если входное значение является объектом.
isDateTime	Возвращает значение true, если входное значение является меткой времени в формате UTC ISO.
isString	Возвращает значение true, если входное значение является строкой.

Предварительно созданные функции сортируются в алфавитном порядке

### add

Возвращает результат сложения двух или более чисел (вариант только с числами) или объединения двух или более строк (другой вариант).

```
add(<item1>, <item2>, ...)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<элемент1>, <элемент2>, ...	Да	any	items

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<итоговая_сумма>	число или строка	Результат сложения указанных чисел или результат объединения.

### Пример

В этом примере добавляются указанные числа:

```
add(1, 1.5)
```

И возвращается результат 2,5.

Этот пример объединяет указанные элементы:

```
add('hello',null)
add('hello','world')
```

И возвращает результаты.

- hello
- helloworld

### addDays

Добавьте к метке времени в необязательном формате языкового стандарта число дней.

```
addDays('<timestamp>', <days>, '<format>'?, '<locale>'?)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<timestamp>	Да	строка	Строка с меткой времени в стандартном формате UTC ISO: YYYY-MM-DDTHH:mm:ss.fffZ
<days>	Да	Целое число	Положительное или отрицательное число дней для добавления
<format>	Нет	строка	<a href="#">Пользовательский шаблон формата</a> . По умолчанию для метки времени используется формат UTC ISO (YYYY-MM-DDTHH:mm:ss.fffZ), который соответствует стандарту <a href="#">ISO 8601</a> .
<языковой стандарт>	нет	строка	Дополнительный языковой стандарт сведений о культуре

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<обновленная_метка_времени>	строка	Метка времени и указанное число дней

### Пример 1

В этом примере к указанной метке времени добавляется 10 дней:

```
addDays('2018-03-15T13:00:00.000Z', 10)
```

И возвращается результат 2018-03-25T00:00:00.000Z.

### Пример 2

В этом примере от указанной метки времени отнимается пять дней:

```
addDays('2018-03-15T00:00:00.000Z', -5)
```

И возвращается результат 2018-03-10T00:00:00.000Z.

### Пример 3

В этом примере добавляется 1 день к заданной метке времени в настройках de-de :

```
addDays('2018-03-15T13:00:00.000Z', 1, '', 'de-dE')
```

И возвращает результат 16.03.18 13:00:00.

## addhours

Добавьте число часов в отметку времени в необязательном формате языкового стандарта.

```
addHours('<timestamp>', <hours>, '<format>?', '<locale>'?)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<timestamp>	Да	строка	Строка, содержащая метку времени
<hours>	Да	Целое число	Положительное или отрицательное число часов для добавления
<format>	Нет	строка	Пользовательский шаблон формата. По умолчанию для метки времени используется формат UTC ISO (YYYY-MM-DDTHH:mm:ss.fffZ), который соответствует стандарту ISO 8601.

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<языковой стандарт>	нет	строка	Дополнительный языковой стандарт сведений о культуре

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<обновленная_метка_времени>	строка	Метка времени и указанное число часов

### Пример 1

В этом примере в указанную метку времени добавляется 10 часов:

```
addHours('2018-03-15T00:00:00.000Z', 10)
```

И возвращается результат 2018-03-15T10:00:00.000Z.

### Пример 2

В этом примере от указанной метки времени отнимается пять часов:

```
addHours('2018-03-15T15:00:00.000Z', -5)
```

И возвращается результат 2018-03-15T10:00:00.000Z.

### Пример 3

В этом примере добавляется 2 часа в указанную отметку времени в языковом стандарте de-de :

```
addHours('2018-03-15T13:00:00.000Z', 2, '', 'de-DE')
```

И возвращает результат 15.03.18 15:00:00.

## addminutes

Добавьте число минут в отметку времени в необязательном формате языкового стандарта.

```
addMinutes('<timestamp>', <minutes>, '<format>'?, '<locale>'?)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<timestamp>	Да	строка	Строка, содержащая метку времени
<minutes>	Да	Целое число	Положительное или отрицательное число минут для добавления

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<format>	Нет	строка	Пользовательский шаблон формата. По умолчанию для метки времени используется формат UTC ISO (YYYY-MM-DDTHH:mm:ss.fffZ), который соответствует стандарту <a href="#">ISO 8601</a> .
<языковой стандарт>	нет	строка	Дополнительный языковой стандарт сведений о культуре

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<обновленная_метка_времени>	строка	Метка времени и указанное число минут

### Пример 1

В этом примере в указанную метку времени добавляется 10 минут:

```
addMinutes('2018-03-15T00:10:00.000Z', 10)
```

И возвращается результат 2018-03-15T00:20:00.000Z.

### Пример 2

В этом примере от указанной метки времени отнимается пять минут:

```
addMinutes('2018-03-15T00:20:00.000Z', -5)
```

И возвращается результат 2018-03-15T00:15:00.000Z.

### Пример 3

В этом примере к заданной метке времени в настройках de-de добавляется 30 минут:

```
addMinutes('2018-03-15T00:00:00.000Z', 30, '', 'de-DE')
```

И возвращает результат 15.03.18 13:30:00.

### addOrdinal

Возвращает порядковый номер полученного числа.

```
addOrdinal(<number>)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<number>	Да	Целое число	Числовые значения для преобразования в порядковый номер.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<результат>	строка	Порядковый номер, полученный из исходного числа.

### Пример

```
addOrdinal(11)
addOrdinal(12)
addOrdinal(13)
addOrdinal(21)
addOrdinal(22)
addOrdinal(23)
```

И возвращаются следующие результаты.

- 11th
- 12th
- 13th
- 21st
- 22nd
- 23rd

### addProperty

Добавляет свойство и его значения или пару "имя — значение" в объект JSON и возвращает обновленный объект. Если объект уже существует во время выполнения, функция выдает ошибку.

```
addProperty('<object>', '<property>', value)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<object>	Да	объект	Объект JSON, в который вы хотите добавить свойство
<property>	Да	строка	Имя добавляемого свойства.
<value>	Да	any	Значение свойства

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<обновленный_объект>	объект	Обновленный объект JSON, полученный после добавления свойства.

### Пример

В этом примере свойство `accountNumber` добавляется в объект `customerProfile`, который затем преобразуется в JSON с помощью функции `json()`. Эта функция присваивает значение, созданное функцией `newGuid()`, и возвращает обновленный объект.

```
addProperty(json('customerProfile'), 'accountNumber', newGuid())
```

### addseconds

Добавляет количество секунд к метке времени.

```
addSeconds('<timestamp>', <seconds>, '<format>'?)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<timestamp>	Да	строка	Строка, содержащая метку времени
<seconds>	Да	Целое число	Положительное или отрицательное число секунд для добавления
<format>	Нет	строка	<a href="#">Пользовательский шаблон формата</a> . По умолчанию для метки времени используется формат UTC ISO (YYYY-MM-DDTHH:mm:ss.fffZ), который соответствует стандарту <a href="#">ISO 8601</a> .

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<обновленная_метка_времени>	строка	Метка времени и указанное число секунд

### Пример 1

В этом примере к указанной метке времени добавляется 10 секунд:

```
addSeconds('2018-03-15T00:00:00.000Z', 10)
```

И возвращается результат `2018-03-15T00:00:10.000Z`.

### Пример 2

В этом примере от указанной метки времени отнимается пять секунд:

```
addSeconds('2018-03-15T00:00:30.000Z', -5)
```

И возвращается результат 2018-03-15T00:00:25.000Z.

### addToTime

Добавьте несколько единиц времени в отметку времени в необязательном формате языкового стандарта. См. раздел [getFutureTime\(\)](#).

```
addToTime('<timestamp>', '<interval>', <timeUnit>, '<format>?', '<locale>?')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<timestamp>	Да	строка	Строка, содержащая метку времени
<interval>	Да	Целое число	Число единиц времени для добавления
<timeUnit>	Да	строка	Единица измерения времени, которая будет применяться к значению <i>interval</i> . Поддерживаются следующие единицы: Second (секунда), Minute (минута), Hour (час), Day (день), Week (неделя), Month (месяц) и Year (год).
<format>	Нет	строка	<a href="#">Пользовательский шаблон формата</a> . По умолчанию для метки времени используется формат UTC ISO (YYYY-MM-DDTHH:mm:ss.fffZ), который соответствует стандарту <a href="#">ISO 8601</a> .
<языковой стандарт>	нет	строка	Дополнительный языковой стандарт сведений о культуре

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<обновленная_метка_времени>	строка	Метка времени с добавлением числа указанных единиц времени в определенном формате.

### Пример 1

В этом примере к указанной метке времени добавляется один день.

```
addToTime('2018-01-01T00:00:00.000Z', 1, 'Day')
```

И возвращается результат 2018-01-02T00:00:00.000Z.

### Пример 2

В этом примере к указанной метке времени добавляются две недели.

```
addToTime('2018-01-01T00:00:00.000Z', 2, 'Week', 'MM-DD-YY')
```

И возвращается результат в формате MM-DD-YY: 01-15-18.

### и

Проверяет, истинны ли все выражения. Возвращает значение `true`, если все выражения истинны, или `false`, если хотя бы одно выражение ложно.

```
and(<expression1>, <expression2>, ...)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<code>&lt;expression1&gt;</code> , <code>&lt;expression2&gt;</code> , ...	Да	Логическое	Выражения, которые следует проверить

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<code>true</code> или <code>false</code>	Логическое	Возвращает значение <code>true</code> , если все выражения истинны. Возвращает значение <code>false</code> , если хотя бы одно выражение ложно.

### Пример 1

В этих примерах проверяется, все ли указанные логические значения верны:

```
and(true, true)  
and(false, true)  
and(false, false)
```

И возвращаются следующие результаты.

- оба выражения имеют значения `true`, поэтому функция возвращает `true`;
- одно выражение имеет значение `false`, поэтому функция возвращает `false`;
- оба выражения имеют значения `false`, поэтому функция возвращает `false`.

### Пример 2

В этих примерах проверяется, все ли указанные выражения верны:

```
and>equals(1, 1), equals(2, 2))  
and>equals(1, 1), equals(1, 2))  
and>equals(1, 2), equals(1, 3))
```

И возвращаются следующие результаты.

- оба выражения имеют значения true, поэтому функция возвращает `true`;
- одно выражение имеет значение false, поэтому функция возвращает `false`;
- оба выражения имеют значения false, поэтому функция возвращает `false`.

### average

Возвращает среднее числовое значение по числовому массиву.

```
average(<numericArray>)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<code>&lt;numericArray&gt;</code>	Да	массив чисел	Входной массив для вычисления среднего арифметического.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<code>&lt;average-of-array&gt;</code>	number	Среднее арифметическое значение по указанному массиву.

### Пример

В этом примере вычисляется среднее по массиву `createArray()`.

```
average(createArray(1,2,3))
```

И возвращается результат 2.

### base64

Возвращает строку или массив байтов в кодировке base64.

```
base64('<value>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<code>&lt;value&gt;</code>	Да	Строка или байтовый массив	Строка входных данных

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<строка_base64>	строка	Исходная строка в формате строки с кодировкой base64.

### Пример 1

В этом примере строка `hello` преобразуется в строку с кодировкой base64.

```
base64('hello')
```

И возвращается результат "aGVsbG8=". .

### Пример 2

В этом примере принимается массив `byteArr` с набором значений `new byte[] { 3, 5, 1, 12 }`.

```
base64('byteArr')
```

И возвращается результат "AwUBDA==". .

### base64ToBinary

Возвращает строку с кодировкой base64 в формате двоичного массива.

```
base64ToBinary('<value>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<value>	Да	строка	Преобразуемая строка с кодировкой base64

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<двоичная версия для строки base64>	массив байтов;	Версия строки с кодировкой base64 в двоичном формате.

### Пример

В этом примере строка `AwUBDA==` с кодировкой base64 преобразуется в двоичную строку:

```
base64ToBinary('AwUBDA==')
```

И возвращается результат `new byte[] { 3, 5, 1, 12 }`.

### base64ToString

Возвращает строковое значение, полученное из строки с кодировкой base64, что по сути означает расшифровку строки base64.

```
base64ToString('<value>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<value>	Да	строка	Декодируемая строка с кодировкой base64.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<декодированная строка base64>	строка	Строковая версия, полученная из строки с кодировкой base64.

### Пример

В этом примере строка aGVsbG8= в кодировке base64 преобразуется в расшифрованную строку.

```
base64ToString('aGVsbG8=')
```

И возвращается результат hello.

### binary

Возвращает строку в двоичном формате.

```
binary('<value>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<value>	Да	строка	Преобразуемая строка

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<двоичная форма входного значения>	массив байтов;	Двоичная строка, полученная из указанной строки.

### Пример

В этом примере строка hello преобразуется в двоичную строку.

```
binary('hello')
```

И возвращается результат new byte[] { 104, 101, 108, 108, 111 } .

### bool

Возвращает значение в логическом формате.

```
bool(<value>)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<value>	Да	any	Значение, которое необходимо преобразовать

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
true или false	Логическое	Указанное значение в логическом формате.

### Пример

В этих примерах указанные значения преобразуются в логические:

```
bool(1)  
bool(0)
```

И возвращаются следующие результаты.

- `true`
- `false`

### ceiling

Возврат самого крупного целочисленного значения, которое меньше или равно указанному числу.

```
ceiling('<number>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<number>	Да	number	Входной номер

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<Целочисленное значение>	целочисленный	Самое большое целое значение, большее или равное входному номеру

### Пример

В этом примере возвращается самое большое целочисленное значение, меньшее или равное числу 10,333:

```
ceiling(10.333)
```

И возвращает целое число 11.

### coalesce

Возвращает первое ненулевое значение из одного или нескольких параметров. Пустые строки, пустые массивы и пустые объекты не имеют значение null.

```
coalesce(<object**1>, <object**2>, ...)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<object**1>, <object**2>, ...	Да	Любой (допустимы смешанные типы)	Один или несколько элементов для проверки на наличие значения NULL

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<первый не нулевой элемент>	any	Первый элемент или значение, которое не равно NULL. Если все параметры равны NULL, эта функция возвращает значение NULL.

#### Пример

Эти примеры возвращают первое значение, не равное NULL, из указанных значений или равное NULL, когда все значения равны NULL:

```
coalesce(null, true, false)
coalesce(null, 'hello', 'world')
coalesce(null, null, null)
```

И соответственно возвращают:

- `true`
- `hello`
- `null`

### concat

Объединить два или более объектов и вернуть объединенные объекты в списке или строке.

```
concat('<text1>', '<text2>', ...)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<object1>, <object2>, ...	Да	any	По крайней мере два объекта для сцепления.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<object1object2...>	строка или список	Объединенная строка или список. Значения NULL пропускаются.

Ожидаемые возвращаемые значения:

- Если все элементы являются списками, будет возвращен список.
- Если существует элемент, который не является списком, будет возвращена строка.
- Если значение равно null, оно пропускается, а не конкатаинатед.

#### Пример

В этом примере объединяются строки Hello и World.

```
concat('Hello', 'World')
```

И возвращается результат HelloWorld.

#### Пример 2

В этом примере объединяются списки [1, 2] и [3, 4]:

```
concat([1,2],[3,4])
```

И возвращает результат [1, 2, 3, 4].

#### Пример 3

В этих примерах объединяются объекты различных типов:

```
concat('a', 'b', 1, 2)
concat('a', [1,2])
```

И возвращаются следующие результаты соответственно:

- Стока AB12.
- Объект System.Collections.Generic.List`1[System.Object]. Это не читается и лучше избегать.

#### Пример 4

В этих примерах объединение объектов будет null следующим:

```
concat([1,2], null)
concat('a', 1, null)
```

И возвращаются следующие результаты соответственно:

- Список [1, 2].
- Стока a1.

#### contains

Проверяет наличие определенного элемента в коллекции. Возвращает true, если элемент найден, или false в противном случае. Эта функция учитывает регистр.

```
contains('<collection>', '<value>')
contains([<collection>], '<value>')
```

Эта функция работает с такими типами коллекций:

- строка, в которой ищется подстрока;
- массив, в котором ищется значение;
- словарь, в котором ищется ключ.

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<collection>	Да	Строка, массив или словарь.	Коллекция для проверки
<value>	Да	Строка, массив или словарь соответственно.	Искомый элемент

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
true или false	Логическое	Возвращает <code>true</code> , если элемент найден. Возвращает <code>false</code> , если не найден.

### Пример 1

В этом примере строка `hello world` проверяется на наличие подстроки `world`.

```
contains('hello world', 'world')
```

И возвращается результат `true`.

### Пример 2

В этом примере строка `hello world` проверяется на наличие подстроки `universe`.

```
contains('hello world', 'universe')
```

И возвращается результат `false`.

### count

Возвращает число элементов в коллекции.

```
count('<collection>')
count([<collection>])
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<collection>	Да	строка или массив	Коллекция с элементами для подсчета

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<длина или число>	Целое число	Число элементов в данной коллекции

*Примеры.*

В этих примерах подсчитывается количество элементов в этих коллекциях:

```
count('abcd')
count(createArray(0, 1, 2, 3))
```

И в обоих возвращается результат 4.

### **countWord**

Возвращает количество слов в исходной строке.

```
countWord('<text>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<text>	Да	строка	Строка для подсчета слов.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<count>	Целое число	Количество слов в исходной строке.

*Пример*

В этом примере подсчитывается количество слов в строке `hello world`.

```
countWord("hello word")
```

И возвращается результат 2.

### **convertFromUTC**

Преобразование метки времени в необязательном формате языкового стандарта из универсального времени (UTC) в целевой часовой пояс.

```
convertFromUTC('<timestamp>', '<destinationTimeZone>', '<format>?', '<locale>?')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<timestamp>	Да	строка	Строка, содержащая метку времени

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<destinationTimeZone>	Да	строка	Имя целевого часового пояса. Поддерживаются часовые пояса Windows и IANA.
<format>	Нет	строка	Пользовательский шаблон формата. Формат отметки времени по умолчанию — <a href="#">Формат "o"</a> , гггг-мм-ддТНН: мм: SS. ffffffK, который соответствует <a href="#">стандарту ISO 8601</a> .
<языковой стандарт>	нет	строка	Дополнительный языковой стандарт сведений о культуре

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<преобразованная метка времени>	строка	Метка времени, преобразованная в целевой часовой пояс

*Примеры:*

В этом примере время в формате UTC преобразовывается в Тихоокеанское время США.

```
convertFromUTC('2018-02-02T02:00:00.000Z', 'Pacific Standard Time', 'MM-DD-YY')
convertFromUTC('2018-02-02T02:00:00.000Z', 'Pacific Standard Time')
```

И возвращаются следующие результаты:

- 02-01-18
- 2018-01-01T18:00:00.0000000

*Пример 2*

В этом примере выполняется преобразование метки времени в языковом стандарте en-US из UTC в тихоокеанское стандартное время:

```
convertFromUTC('2018-01-02T02:00:00.000Z', 'Pacific Standard Time', 'D', 'en-US')
```

И возвращает результат в **понедельник, 1 января 2018**.

### convertToUTC

Преобразуйте метку времени в необязательном формате языкового стандарта в универсальное скоординированное время (UTC) из исходного часового пояса.

```
convertToUTC('<timestamp>', '<sourceTimeZone>', '<format>?', '<locale>?')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<timestamp>	Да	строка	Строка, содержащая метку времени
<sourceTimeZone>	Да	строка	Имя целевого часового пояса. Поддерживаются часовые пояса Windows и IANA.
<format>	Нет	строка	<a href="#">Пользовательский шаблон формата</a> . По умолчанию для метки времени используется формат UTC ISO (YYYY-MM-DDTHH:mm:ss.fffZ), который соответствует стандарту <a href="#">ISO 8601</a> .
<языковой стандарт>	нет	строка	Дополнительный языковой стандарт сведений о культуре

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<преобразованная метка времени>	строка	Метка времени, преобразованная в целевой часовой пояс

### Пример

В этом примере метка времени преобразуется в формат UTC по тихоокеанскому времени (зима).

```
convertToUTC('01/01/2018 00:00:00', 'Pacific Standard Time')
```

И возвращает результат 2018-01-01T08:00:00.000 z.

### Пример 2

В этом примере отметка времени в локале de-de преобразуется в формат UTC по тихоокеанскому времени:

```
convertToUTC('01/01/2018 00:00:00', 'Pacific Standard Time', '', 'de-DE')
```

И возвращает результат 01.01.18 08:00:00.

### createArray

Возвращает массив из нескольких экземпляров входных данных.

```
createArray('<object1>', '<object2>', ...)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<object1>, <object2>, ...	Да	Любой, но не смешанный.	По крайней мере два элемента для создания массива

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
[<объект1>, <объект2>, ...]	массив	Массив, созданный из всех входных элементов

### Пример

В этом примере создается массив из следующих входных элементов:

```
createArray('h', 'e', 'l', 'l', 'o')
```

И возвращается результат [h ,e, l, l, o] .

### dataUri

Возвращает URI данных для строки.

```
dataUri('<value>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<value>	Да	строка	Преобразуемая строка

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
[<date-uri>]	строка	URI данных для входной строки

### Пример

```
dataUri('hello')
```

Возвращается результат data:text/plain;charset=utf-8;base64,aGVsbG8= .

### dataUriToBinary

Возвращает URI данных в двоичном формате.

```
dataUriToBinary('<value>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<value>	Да	строка	URI данных для преобразования

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
[<binary-for-data-uri>]	массив байтов;	URI данных в двоичном формате.

### Пример

В этом примере создается двоичная версия следующего URI данных.

```
dataUriToBinary('aGVsbG8=')
```

И возвращается результат new byte[] { 97, 71, 86, 115, 98, 71, 56, 61 } .

### dataUriToString

Возвращает URI данных в строковом формате.

```
dataUriToString('<value>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<value>	Да	строка	URI данных для преобразования

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
[<string-for-data-uri>]	строка	Строковая версия URI данных.

### Пример

В этом примере создается строковое значение на основе следующего URI данных.

```
dataUriToString('data:text/plain;charset=utf-8;base64,aGVsbG8=')
```

И возвращается результат hello.

### Дата

Возвращает значение указанной метки времени в формате m/dd/yyyy.

```
date('<timestramp>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<timestamp>	Да	строка	Строка, содержащая метку времени

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<date>	строка	Дата из указанной метки времени.

```
date('2018-03-15T13:00:00.000Z')
```

И возвращается результат 3-15-2018.

### dateReadBack

Обратное чтение даты с помощью библиотеки date-time.

```
dateReadBack('<currentDate>', '<targetDate>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<currentDate>	Да	строка	Строка, содержащая текущую дату.
<targetDate>	Да	строка	Строка, содержащая целевую дату.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<date-readback>	строка	Обратное чтение значений от текущей даты до целевой даты.

### Пример 1

```
dateReadBack('2018-03-15T13:00:00.000Z', '2018-03-16T13:00:00.000Z')
```

Возвращается значение tomorrow (завтра).

### датетимедифф

Возвращает разность в тактах между двумя метками времени.

```
dateTimeDiff('<timestamp1>', '<timestamp2>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
----------	-------------	-----	----------

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<timestamp1>	Да	строка	Первая строка метки времени для сравнения
<timestamp2>	Да	строка	Вторая строка метки времени для сравнения

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<тактов>	number	Разница между двумя метками времени в тактах

### Пример 1

В этом примере возвращается разность между двумя метками времени в тактах:

```
dateTimeDiff('2019-01-01T08:00:00.000Z', '2018-01-01T08:00:00.000Z')
```

И возвращает число 315360000000000.

### Пример 2

В этом примере возвращается разность между двумя метками времени в тактах:

```
dateTimeDiff('2018-01-01T08:00:00.000Z', '2019-01-01T08:00:00.000Z')
```

Возвращает результат -315360000000000. Обратите внимание, что значение является отрицательным числом.

### dayOfMonth

Возвращает день месяца из метки времени.

```
dayOfMonth('<timestamp>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<timestamp>	Да	строка	Строка, содержащая метку времени

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<день месяца>	Целое число	День месяца из указанной метки времени

### Пример

В этом примере возвращается число дня месяца из следующей метки времени.

```
dayOfMonth('2018-03-15T13:27:36Z')
```

И возвращается результат 15.

### dayOfWeek

Возвращает день недели из метки времени.

```
dayOfWeek('<timestamp>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<timestamp>	Да	строка	Строка, содержащая метку времени

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<день недели>	Целое число	День недели из указанной метки времени. Здесь 0 обозначает воскресенье, 1 — понедельник и т. д.

### Пример

В этом примере возвращается номер дня недели из следующей метки времени.

```
dayOfWeek('2018-03-15T13:27:36Z')
```

И возвращается результат 3.

### dayOfYear

Возвращает день года из метки времени.

```
dayOfYear('<timestamp>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<timestamp>	Да	строка	Строка, содержащая метку времени

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<день года>	Целое число	День года из указанной метки времени

### Пример

В этом примере возвращается порядковый номер дня в году из следующей метки времени.

```
dayOfYear('2018-03-15T13:27:36Z')
```

И возвращается результат **74**.

## div

Возвращает целочисленный результат деления двух чисел. Чтобы получить остаток, см. инструкции в разделе [mod\(\)](#).

```
div(<dividend>, <divisor>)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<dividend>	Да	number	Число, которое нужно поделить на <i>делитель</i>
<divisor>	Да	number	Число, на которое делится <i>делимое</i> . Не может иметь значение 0.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<результат деления>	number	Результат от деления первого числа на второе.

### Пример

В обоих примерах первое число делится на второе:

```
div(10, 5)
div(11, 5)
```

И возвращается результат 2.

Если любой из параметров имеет формат числа с плавающей запятой, результат также возвращается в этом формате.

### Пример

```
div(11.2, 3)
```

Возвращается результат 5,5.

## empty

Проверяет, пуст ли экземпляр. Возвращает `true`, если входные данные пусты. Пустые входные данные означают:

- входные данные имеют значение NULL или не определены;
- входные данные имеют значение NULL или являются пустой строкой;
- входные данные являются коллекцией нулевого размера;
- входные данные являются объектом без свойства.

```
empty('<instance>')
empty([<instance>])
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<instance>	Да	any	Проверяемый экземпляр

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
true или false	Логическое	Возвращается значение <code>true</code> , если экземпляр является пустым.

### Пример

В этих примерах проверяется, является ли указанный экземпляр пустым.

```
empty('')
empty('abc')
empty([1])
empty(null)
```

И возвращаются такие результаты соответственно:

- передается пустая строка, поэтому функция возвращает `true`.
- Передается строка `abc`, поэтому функция возвращает `false`.
- Передает коллекцию с одним элементом, поэтому функция возвращает `false`.
- Передает объект `NULL`, поэтому функция возвращает `true`.

### endsWith

Проверяет, заканчивается ли строка определенной подстрокой. Возвращает `true`, если подстрока найдена, или `false` в противном случае. Эта функция не учитывает регистр.

```
endsWith('<text>', '<searchText>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<text>	Да	строка	Проверяемая строка
<searchText>	Да	строка	Конечная подстрока для поиска

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
true или false	Логическое	Возвращает значение <code>true</code> , если конечная подстрока обнаружена. Возвращает <code>false</code> , если не обнаружена.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
-----------------------	-----	----------

### Пример 1

В этом примере проверяется, заканчивается ли строка `hello world` строкой `world`.

```
endsWith('hello world', 'world')
```

И возвращается результат `true`.

### Пример 2

В этом примере проверяется, заканчивается ли строка `hello world` строкой `universe`.

```
endsWith('hello world', 'universe')
```

И возвращается результат `false`.

## КОНЦА строки

Только для C#. Возврат текста последовательности конца строки (конца строки).

```
EOL()
```

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<code>&lt;исосплатформ&gt;</code>	строка	Возвратите <code>\r\n</code> в Windows и <code>\n</code> в Mac и Linux.

### Пример

В этом примере проверяется конец текста последовательностей строки:

```
EOL()
```

И возвращает следующие строки:

- Windows: `\r\n`
- Mac или Linux: `\n`

## equals (равно)

Проверяет, эквивалентны ли оба значения, выражения или объекта. Возвращается значение `true`, если они эквивалентны, или `false`, если нет.

```
equals('<object1>', '<object2>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<code>&lt;object1&gt;, &lt;object2&gt;</code>	Да	any	Значения, выражения или объекты для сравнения

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
true или false	Логическое	Возвращает значение <code>true</code> , если оба значения эквивалентны. Возвращает значение <code>false</code> , если это не так.

### Пример

В этих примерах проверяется, являются ли указанные входные данные эквивалентными.

```
equals(true, 1)
equals('abc', 'abcd')
```

И возвращаются такие результаты:

- оба значения эквивалентны, поэтому функция возвращает `true`.
- оба значения не эквивалентны, поэтому функция возвращает `false`.

### exists (существует)

Проверяет истинность выражения.

```
exists(expression)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
expression	Да	expression	Выражение, которое будет проверяться на истинность.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<true или false>	Логическое	Результат проверки выражения.

### Пример

В следующих примерах проверяется истинность `foo = {"bar": "value"}`.

```
exists(foo.bar)
exists(foo.bar2)
```

И возвращаются такие результаты соответственно:

- `true`
- `false`

### exp

Возвращает результат возведения указанного числа в указанную степень.

```
exp(realNumber, exponentNumber)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
realNumber	Да	number	Вещественное число для вычисления степени.
exponentNumber	Да	number	Показатель степени.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<result-exp>	number	Результат возведения <code>realNumber</code> в указанную степень.

### Пример

В этом примере вычисляется степень числа.

```
exp(2, 2)
```

И возвращается результат 4.

### first

Возвращает первый элемент из строки или массива.

```
first('<collection>')
first([<collection>])
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<collection>	Да	строка или массив	Коллекция, в которой нужно найти первый элемент.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<первый элемент коллекции>	any	Первый элемент в коллекции

### Пример

В этих примерах выполняется поиск первого элемента в следующих коллекциях.

```
first('hello')
first(createArray(0, 1, 2))
```

И возвращаются такие результаты соответственно:

- h
- 0

### преобразовать в плоский формат

Преобразует массив в плоскую структуру, чтобы его значения не являлись массивами. Вы можете указать необязательный параметр максимальной глубины преобразования.

```
flatten([<collection>], '<depth>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<collection>	Да	массив	Коллекция для преобразования в плоскую структуру.
<depth>	Нет	number	Максимальная глубина для преобразования в плоскую структуру. По умолчанию не имеет ограничений.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<new-collection>	массив	Новая коллекция, элементы которой преобразованы в плоскую структуру до указанной глубины.

#### Пример 1

В этом примере следующий массив преобразуется в плоскую структуру.

```
flatten(createArray(1, createArray(2), createArray(createArray(3, 4), createArray(5, 6))))
```

И возвращается результат [1, 2, 3, 4, 5, 6] .

#### Пример 2

В этом примере массив преобразуется в плоскую структуру до глубины 1.

```
flatten(createArray(1, createArray(2), createArray(createArray(3, 4), createArray(5, 6))), 1)
```

И возвращается результат [1, 2, [3, 4], [5, 6]] .

## FLOAT

Преобразует строковую версию числа с плавающей запятой в числовое представление. Эта функция используется только при передаче пользовательских параметров в приложение, такое как приложение логики. Если строка не может быть преобразована в тип float, будет создано исключение.

```
float('<value>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<value>	Да	строка	Строка для преобразования, содержащая допустимое представление числа с плавающей запятой.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<значение с плавающей запятой>	FLOAT	Число с плавающей запятой, полученное из указанной строки.

### Пример

В этом примере преобразуется версия с плавающей запятой строки:

```
float('10.333')
```

И возвращает значение float 10,333.

### floor

Возврат самого крупного целочисленного значения, которое меньше или равно указанному числу.

```
floor('<number>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<number>	Да	number	Входной номер

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<Целочисленное значение>	целочисленный	Самое большое целое значение, меньшее или равное входному номеру

### Пример

В этом примере вычисляется значение Floor для числа 10,333:

```
floor(10.333)
```

И возвращает целое число 10.

### foreach

Обрабатывает каждый элемент и возвращают новую коллекцию.

```
foreach([<collection-instance>], <iteratorName>, <function>)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<collection-instance>	Да	массив или объект	Коллекция с элементами.
<iteratorName>	Да	Имя итератора.	Ключевой элемент стрелочной функции.
<function>	Да	expression	Функция, которая содержит <code>iteratorName</code> .

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<new-collection>	массиве	Новая коллекция, в которой к каждому элементу применена оценка указанной функцией.

### Пример 1

В этом примере создается новая коллекция.

```
foreach(createArray(0, 1, 2, 3), x, x + 1)
```

И возвращается результат [1, 2, 3, 4].

### Пример 2

В этих примерах создается новая коллекция:

```
foreach(json("{ 'name': 'jack', 'age': '15' }"), x, concat(x.key, ':', x.value))
foreach(json("{ 'name': 'jack', 'age': '15' }"), x=> concat(x.key, ':', x.value))
```

И возвращают результат ["имя: гнездо", "Age: 15"]. Обратите внимание, что второе выражение является **лямбда-выражением**, которое может оказаться более удобочитаемым.

## formatDateTime

Возврат метки времени в необязательном формате языкового стандарта.

```
formatDateTime('<timestamp>', '<format>?', '<locale>?')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<timestamp>	Да	строка	Строка, содержащая метку времени

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<format>	Нет	строка	Пользовательский шаблон формата. По умолчанию для метки времени используется формат UTC ISO (YYYY-MM-DDTHH:mm:ss.fffZ), который соответствует стандарту <a href="#">ISO 8601</a> .
<языковой стандарт>	нет	строка	Дополнительный языковой стандарт сведений о культуре

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<метка времени в другом формате>	строка	Обновленная метка времени в указанном формате

### Пример 1

В этом примере метка времени преобразуется в указанный формат:

```
formatDateTime('03/15/2018 12:00:00', 'yyyy-MM-ddTHH:mm:ss')
```

И возвращает результат 2018-03-15T12:00:00.

### Пример 2

В этом примере выполняется преобразование метки времени в языковой стандарт de-de :

```
formatDateTime('2018-03-15', '', 'de-DE')
```

И возвращает результат 15.03.18 00:00:00.

## formatEpoch

Возврат метки времени в необязательном формате языкового стандарта в указанном формате из времени UNIX (также известно как время эпохи, время POSIX, время эпохи UNIX).

```
formatEpoch('<epoch>', '<format>?', '<locale>'?)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<epoch>	Да	number	Номер эпохи

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<format>	Нет	строка	Пользовательский шаблон формата. По умолчанию для метки времени используется формат UTC ISO (YYYY-MM-DDTHH:mm:ss.fffZ), который соответствует стандарту <a href="#">ISO 8601</a> .
<языковой стандарт>	нет	строка	Дополнительный языковой стандарт сведений о культуре

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<метка времени в другом формате>	строка	Обновленная метка времени в указанном формате

### Пример

В этом примере отметка времени Unix преобразуется в заданный формат:

```
formatEpoch(1521118800, 'yyyy-MM-ddTHH:mm:ss.fffZ')
```

И возвращается результат 2018-03-15T12:00:00.000Z.

### Пример

В этом примере выполняется преобразование метки времени Unix в языковой стандарт de-de :

```
formatEpoch(1521118800, '', 'de-DE')
```

И возвращает результат 15.03.18 13:00:00.

### formatNumber

Форматирование значения до указанного числа цифр дробной части и необязательного заданного языкового стандарта.

```
formatNumber('<number>', '<precision-digits>', '<locale>'?)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<number>	Да	number	Входной номер
<точность-цифры>	Да	Целое число	Указанное число цифр дробной части

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<языковой стандарт>	нет	строка	Дополнительный языковой стандарт сведений о культуре

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<Возвращаемое значение>	number	Возвращаемое значение входного формата с указанным числом дробных разрядов и указанным языковым стандартом

### Пример 1

Этот пример форматирует данные об число 10,333 до 2 цифр дробной части:

```
formatNumber(10.333, 2)
```

И возвращает строку 10,33.

### Пример 2

В этих примерах нумерация форматируется в указанное число цифр в языковом стандарте en-US :

```
formatNumber(12.123, 4, 'en-US')
formatNumber(1.551, 2, 'en-US')
formatNumber(12.123, 4, 'en-US')
```

И возвращаются следующие результаты соответственно:

- 12,12
- 1,55
- 12,1230

### formatTicks

Возврат метки времени в необязательном формате языкового стандарта в указанном формате из тактов.

```
formatTicks('<ticks>', '<format>?', '<locale>'?)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<epoch>	Да	number (или bigint в JavaScript)	Число тактов

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<format>	Нет	строка	Пользовательский шаблон формата. По умолчанию для метки времени используется формат UTC ISO (YYYY-MM-DDTHH:mm:ss.ffffZ), который соответствует стандарту <a href="#">ISO 8601</a> .
<языковой стандарт>	нет	строка	Дополнительный языковой стандарт сведений о культуре

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<метка времени в другом формате>	строка	Обновленная метка времени в указанном формате

### Пример 1

В этом примере такты преобразуются в указанный формат:

```
formatTicks(637243624200000000, 'yyyy-MM-ddTHH:mm:ss.ffffZ')
```

И возвращается результат 2020-05-06T11:47:00.000Z.

### Пример 2

В этом примере выполняется преобразование тактов в указанный формат в языковом стандарте de-de :

```
formatTicks(637243624200000000, '', 'de-DE')
```

И возвращает результат 06.05.20 11:47:00.

### getFutureTime

Возврат текущей метки времени в необязательном формате языкового стандарта и заданных единиц времени.

```
getFutureTime(<interval>, <timeUnit>, '<format>'?, '<locale>'?)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<interval>	Да	Целое число	Число указанных единиц времени для добавления.

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<timeUnit>	Да	строка	Единица измерения времени, которая будет применяться к значению <i>interval</i> . Поддерживаются следующие единицы: Second (секунда), Minute (минута), Hour (час), Day (день), Week (неделя), Month (месяц) и Year (год).
<format>	Нет	строка	Пользовательский шаблон формата. По умолчанию для метки времени используется формат UTC ISO (YYYY-MM-DDTHH:mm:ss.fffZ), который соответствует стандарту <a href="#">ISO 8601</a> .
<языковой стандарт>	нет	строка	Дополнительный языковой стандарт сведений о культуре

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<обновленная_метка_времени>	строка	Текущая метка времени и указанное число единиц времени

### Пример 1

Допустим, текущая метка времени такая: 2019-03-01T00:00:00.000Z. В следующем примере к этой метке времени добавляются пять дней.

```
getFutureTime(2, 'Week')
```

И возвращается результат 2019-03-15T00:00:00.000Z.

### Пример 2

Допустим, текущая метка времени такая: 2018-03-01T00:00:00.000Z. В следующем примере к этой метке времени добавляются пять дней и результат преобразуется в формат MM-DD-YY.

```
getFutureTime(5, 'Day', 'MM-DD-YY')
```

И возвращает результат 03-06-18.

### Пример 3

Предположим, что текущая метка времени — 2020-05-01T00:00:00.000 Z, а языковой стандарт — de-de. В приведенном ниже примере в отметку времени добавляется 1 день.

```
getFutureTime(1, 'Day', '', 'de-DE')
```

И возвращает результат 02.05.20 00:00:00.

### **жетнекствиабледате**

Возврат следующей допустимой даты выражения Timex на основе текущей даты и дополнительно заданного часового пояса.

```
getNextViableDate(<timexString>, <timezone>?)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<тимексстринг>	Да	строка	Строка Timex даты для вычисления.
<стандарт>	Нет	строка	Необязательный часовой пояс.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<некствиаблетиме>	строка	Следующая приемлемая Дата.

### *Примеры*

Предположим, что дата — 2020-06-12 , а текущее время — 15:42:21.

Эти примеры оценивают строку Timex для следующей допустимой даты на основе указанных выше даты и времени:

```
getPreviousViableDate("XXXX-12-20", "America/Los_Angeles")
getPreviousViableDate("XXXX-02-29")
```

И возвращают следующие строки соответственно:

- 2020-12-20
- 2024-02-29

### **жетнекствиаблетиме**

Возвращает следующее допустимое время выражения Timex на основе текущего времени и дополнительно заданного часового пояса.

```
getNextViableTime(<timexString>, <timezone>?)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<тимексстринг>	Да	строка	Строка Timex времени для вычисления.

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<стандарт>	Нет	строка	Необязательный часовой пояс.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<некствиаблетиме>	строка	Следующий приемлемый момент времени.

### Примеры

Предположим, что дата — 2020-06-12 , а текущее время — 15:42:21.

В этих примерах вычисляется Timex строка для следующего допустимого времени на основе указанных выше даты и времени.

```
getNextViableTime("TXX:12:14", "Asia/Tokyo")
getNextViableTime("TXX:52:14")
```

И возвращают следующие строки репсективели:

- T16:12:14
- T15:52:14

### getPastTime

Возвращает текущую метку времени, вычитая указанные единицы времени.

```
getPastTime(<interval>, <timeUnit>, '<format>'?)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<interval>	Да	Целое число	Число указанных единиц времени для вычитания.
<timeUnit>	Да	строка	Единица измерения времени, которая будет применяться к значению <i>interval</i> . Поддерживаются следующие единицы: Second (секунда), Minute (минута), Hour (час), Day (день), Week (неделя), Month (месяц) и Year (год).
<format>	Нет	строка	Пользовательский шаблон формата. По умолчанию для метки времени используется формат UTC ISO (YYYY-MM-DDTHH:mm:ss.fffZ), который соответствует стандарту ISO 8601.

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<обновленная_метка_времени>	строка	Текущая метка времени за вычетом указанного числа единиц времени

### Пример 1

Допустим, текущая метка времени такая: 2018-02-01T00:00:00.000Z. В этом примере от указанной метки времени отнимается пять дней.

```
getPastTime(5, 'Day')
```

И возвращается результат 2019-01-27T00:00:00.000Z.

### Пример 2

Допустим, текущая метка времени такая: 2018-03-01T00:00:00.000Z. В этом примере вычитается пять дней до метки времени в формате **ММ-ДД-ГГ**:

```
getPastTime(5, 'Day', 'MM-DD-YY')
```

И возвращает результат 02-26-18.

### Пример 3

Предположим, что текущая метка времени — 2020-05-01T00:00:00.000 z , а языковой стандарт — de-de. В приведенном ниже примере вычитается 1 день из метки времени:

```
getPastTime(1,'Day', '', 'de-DE')
```

И возвращает результат 31.04.20 00:00:00.

### жетпревиаусвиабледате

Возврат предыдущей допустимой даты выражения Timex на основе текущей даты и дополнительно заданного часового пояса.

```
getPreviousViableDate(<timexString>, <timezone>?)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<тимексстринг>	Да	строка	Строка Timex даты для вычисления.
<стандарт>	Нет	строка	Необязательный часовой пояс.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<превиаусвиабледате>	строка	Предыдущая приемлемая Дата.

### Примеры

Предположим, что дата — 2020-06-12 , а текущее время — 15:42:21.

Эти примеры оценивают Timex строку для предыдущей допустимой даты на основе указанных выше даты и времени:

```
getPreviousViableDate("XXXX-12-20", "Eastern Standard Time")
getPreviousViableDate("XXXX-02-29")
```

И возвращают следующие строки репсективели:

- 2019-12-20
- 2020-02-29

### жетпревиаусвиаблетьиме

Возвращает предыдущее допустимое время выражения Timex на основе текущей даты и дополнительно заданного часового пояса.

```
getPreviousViableTime(<timexString>, <timezone>?)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<тимексстринг>	Да	строка	Строка Timex времени для вычисления.
<стандарт>	Нет	строка	Необязательный часовой пояс.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<превиаусвиаблетьиме>	строка	Предыдущее допустимое время.

### Примеры

Предположим, что дата — 2020-06-12 , а текущее время — 15:42:21.

В этих примерах вычисляется Timex строка для предыдущего периода времени в зависимости от указанных выше даты и времени.

```
getPreviousViableTime("TXX:52:14")
getPreviousViableTime("TXX:12:14", 'Europe/London')
```

И возвращают следующие строки репсективели:

- T14:52:14

- T15:12:14

## getProperty

Возврат значения указанного свойства или корневого свойства из объекта JSON.

### Возврат значения указанного свойства

```
getProperty(<JSONObject>, '<propertyName>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<JSONObject>	Да	объект	Объект JSON, содержащий свойство и значения.
<propertyName>	Нет	строка	Имя необязательного свойства, из которого нужно получить доступ к значениям.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
value	строка	Значение указанного свойства в объекте JSON.

### Пример

Предположим, у вас есть следующий объект JSON:

```
{
    "a:b" : "a:b value",
    "c":
    {
        "d": "d key"
    }
}
```

В этом примере извлекается указанное свойство из указанного выше объекта JSON:

```
getProperty({"a:b": "value"}, 'a:b')
getProperty(c, 'd')
```

И возвращают следующие строки соответственно:

- значение a:b
- ключ d

### Возврат корневого свойства

```
getProperty('<propertyName>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<propertyName>	Да	строка	Имя необязательного свойства для доступа к значениям из области корневой памяти.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
value	строка	Значение корневого свойства в объекте JSON.

### Пример

Предположим, у вас есть следующий объект JSON:

```
{
  "a:b" : "a:b value",
  "c":
  {
    "d": "d key"
  }
}
```

В этом примере извлекается корневое свойство из приведенного выше объекта JSON:

```
getProperty("a:b")
```

И возвращает строковое значение **а:б**.

### getTimeOfDay

Возвращает время дня для указанной метки времени.

```
getTimeOfDay('<timestamp>')
```

Это может быть одна из следующих строк.

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<timestamp>	Да	строка	Строка, содержащая исходную метку времени.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<time-of-day>	строка	Время дня, полученное для указанной метки времени.

Ниже перечислены строки, которые могут обозначать время дня.

ВРЕМЯ ДНЯ	ОТМЕТКА ВРЕМЕНИ
midnight	24:00
morning (утро)	00:01–11:59
afternoon (день)	12:00
evening (вечер)	18:00–22:00
ночь	22:01–23:59

### Пример

```
getTimeOfDay('2018-03-15T08:00:00.000Z')
```

Возвращается значение morning (утро).

### greater

Проверяет, является ли первое значение большим, чем второе. Возвращает значение `true`, если первое значение больше второго, или `false` в противном случае.

```
greater(<value>, <compareTo>)
greater('<value>', '<compareTo>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<code>&lt;value&gt;</code>	Да	Целое число, число с плавающей запятой или строка.	Проверяет, является ли первое значение большим, чем второе
<code>&lt;compareTo&gt;</code>	Да	Целое число, число с плавающей запятой или строка соответственно.	Значение сравнения

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<code>true</code> или <code>false</code>	Логическое	Возвращает <code>true</code> , если первое значение больше второго. Возвращает <code>false</code> , если первое значение меньше второго или равно ему.

### Пример

В этих примерах проверяется, является ли первое значение большим, чем второе.

```
greater(10, 5)
greater('apple', 'banana')
```

И возвращаются следующие результаты соответственно:

- `true`
- `false`

### greaterOrEquals

Проверяет, является ли первое значение большим, чем второе, или равным ему. Возвращает `true`, если первое значение больше второго или равно ему, или `false` в противном случае.

```
greaterOrEquals(<value>, <compareTo>)
greaterOrEquals('<value>', '<compareTo>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<code>&lt;value&gt;</code>	Да	Целое число, число с плавающей запятой или строка.	Проверяет, является ли первое значение большим, чем второе, или равным ему
<code>&lt;compareTo&gt;</code>	Да	Целое число, число с плавающей запятой или строка соответственно.	Значение сравнения

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<code>true</code> или <code>false</code>	Логическое	Возвращает <code>true</code> , если первое значение больше второго или равно ему. Возвращает <code>false</code> , если первое значение меньше второго.

### Пример

В этих примерах проверяется, является ли первое значение большим чем второе, или равным ему.

```
greaterOrEquals(5, 5)
greaterOrEquals('apple', 'banana')
```

И возвращаются следующие результаты соответственно:

- `true`
- `false`

### if

Проверьте, какое значение имеет выражение: `true` или `false`. Возвращает указанное значение на основе результата.

```
if(<expression>, <valueIfTrue>, <valueIfFalse>)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<code>&lt;expression&gt;</code>	Да	Логическое	Выражение для вычисления

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<valuelfTrue>	Да	any	Возвращаемое значение, если выражение истинно.
<valuelfFalse>	Да	any	Возвращаемое значение, если выражение ложно.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<указанное возвращаемое значение>	any	Указанное значение, которое возвращается в зависимости от результата вычисления выражения: <code>true</code> ИЛИ <code>false</code> .

### Пример

В этом примере вычисляется, является ли `equals(1,1)` истинным.

```
if>equals(1, 1), 'yes', 'no')
```

Здесь возвращается значение `yes`, так как указанное выражение возвращает результат `true`. В противном случае этот пример возвращал бы `no`.

### indexOf

Возвращает начальную позицию или значение индекса для подстроки. Эта функция не учитывает регистр, а значения индексов начинаются с 0.

```
indexOf('<text>', '<searchText>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<text>	Да	строка или массив	Строки, которая содержит подстроку для поиска
<searchText>	Да	строка	Подстрока для поиска

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<значение индекса>	Целое число	Начальное положение или значение индекса указанной подстроки.
Если строка не найдена, возвращается -1.		

### Пример 1

В этом примере выполняется поиск начального значения индекса для подстроки `world` в строке `hello`

`world.`

```
indexOf('hello world', 'world')
```

И возвращается результат 6.

### Пример 2

В этом примере выполняется поиск начального значения индекса для подстроки `def` в массиве `['abc', 'def', 'ghi']`:

```
indexOf(createArray('abc', 'def', 'ghi'), 'def')
```

И возвращается результат 1.

### indicesAndValues

Преобразует массив или объект в массив объектов, состоящих из индексов (текущий индекс) и свойств значений. Для массивов индекс обозначает положение в этом массиве. Для объектов это ключ, по которому можно обращаться к значению.

```
indicesAndValues('<collection or object>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<code>&lt;коллекция или объект&gt;</code>	Да	массив или объект	Исходный массив или объект

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<code>&lt;collection&gt;</code>	массиве	Новый массив. У каждого элемента есть два свойства: индекс, который обозначает положение в массиве или ключ в объекте, и соответствующее индексу значение.

### Пример 1

Предположим, у вас есть такой список: `{ items: ["zero", "one", "two"] }`. Следующая функция принимает этот список.

```
indicesAndValues(items)
```

И возвращает новый список.

```
[  
  {  
    index: 0,  
    value: 'zero'  
  },  
  {  
    index: 1,  
    value: 'one'  
  },  
  {  
    index: 2,  
    value: 'two'  
  }  
]
```

### Пример 2

Предположим, у вас есть такой список: { items: ["zero", "one", "two"] } . Следующая функция принимает этот список.

```
where(indicesAndValues(items), elt, elt.index >= 1)
```

И возвращает новый список.

```
[  
  {  
    index: 1,  
    value: 'one'  
  },  
  {  
    index: 2,  
    value: 'two'  
  }  
]
```

### Пример 3

Предположим, у вас есть такой список: { items: ["zero", "one", "two"] } . Следующая функция принимает этот список.

```
join(foreach(indicesAndValues(items), item, item.value), ',')
```

И возвращается результат zero,one,two. Это выражение дает такой же результат, как и [join\(items, ','\)](#).

### Пример 4

Предположим, у вас есть такой объект: { user: {name: 'jack', age: 20} } . Следующая функция принимает этот объект.

```
indicesAndValues(user)
```

И возвращает новый объект.

```
[  
  {  
    index: 'name',  
    value: 'jack'  
  },  
  {  
    index: 'age',  
    value: 20  
  }  
]
```

## INT

Возвращает целочисленную версию строки. Если строка не может быть преобразована в целое число, будет создано исключение.

```
int('<value>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<value>	Да	строка	Преобразуемая строка

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<целочисленный результат>	Целое число	Целочисленная версия указанной строки.

### Пример

В этом примере создается целочисленная версия для строки 10.

```
int('10')
```

И возвращается результат 10 (целое число).

## пересечению

Возвращает коллекцию, которая содержит только общие элементы в указанных коллекциях. Чтобы появиться в результатах, элемент должен находиться во всех коллекциях, переданных этой функции. Если один или несколько элементов имеют одинаковое имя, в результатах появляется последний элемент с таким именем.

```
intersection([<collection1>], [<collection2>], ...)  
intersection('<collection1>', '<collection2>', ...)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<collection1>, <collection2>	Да	Массив или объект, но не оба типа сразу.	Коллекции, из которых нужно получить только общие элементы.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<общий элементы>	Массив или объект соответственно.	Коллекция, которая содержит только общие элементы в указанных коллекциях

### Пример

В этом примере выполняется поиск общих элементов в следующих массивах.

```
intersection(createArray(1, 2, 3), createArray(101, 2, 1, 10), createArray(6, 8, 1, 2))
```

И возвращается массив только с элементами [1, 2].

### isArray

Возвращает `true`, если входное значение является массивом.

```
isArray('<input>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
< <i>input</i> >	Да	any	Входное значение для проверки.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
< <i>Boolean-result</i> >	Логическое	Возвращает <code>true</code> , если входные данные содержат массив, или <code>false</code> в противном случае.

### Примеры

В следующем примере производится проверка, является ли входное значение объектом.

```
isArray('hello')
isArray(createArray('hello', 'world'))
```

И возвращаются следующие результаты соответственно:

- входные данные представляют собой строку, поэтому функция возвращает `false`;
- входные данные представляют собой массив, поэтому функция возвращает `true`;

### isBoolean

Возвращает `true`, если входное значение является логическим.

```
isBoolean('<input>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<input>	Да	any	Входное значение для проверки.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<Boolean-result>	Логическое	Возвращает <code>true</code> , если получено логическое входное значение, или <code>false</code> в противном случае.

### Примеры

В следующем примере производится проверка, является ли входное значение логическим.

```
isBoolean('hello')
isBoolean(32 > 16)
```

И возвращаются следующие результаты соответственно:

- входные данные представляют собой строку, поэтому функция возвращает `false`;
- входные данные представляют собой логическое значение, поэтому функция возвращает `true`.

### isDate

Возвращает значение `true`, если полученный объект TimexProperty или выражение Timex указывает на допустимую дату. Допустимыми считаются даты, для которых определены месяц и день месяца или день недели.

```
isDate('<input>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<input>	Да	Объект или строка	Входные данные в формате объекта TimexProperty или строки выражения Timex.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<boolean-result>	Логическое	Возвращает <code>true</code> , если входные данные обозначают допустимую дату, или <code>false</code> в противном случае.

### Примеры

В этих примерах производится проверка, являются ли входные значения допустимыми датами.

```
isDate('2020-12')
isDate('xxxx-12-21')
```

И возвращаются следующие результаты:

- false
- true

### isDateRange

Возвращает значение  true , если полученный объект TimexProperty или выражение Timex указывает на допустимый диапазон дат.

```
isDateRange('<input>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<input>	Да	Объект или строка	Входные данные в формате объекта TimexProperty или строки выражения Timex.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<boolean-result>	Логическое	Возвращает <input type="checkbox"/> true , если полученные входные данные определяют допустимый диапазон дат, или <input checked="" type="checkbox"/> false в противном случае.

### Примеры

В этих примерах производится проверка, являются ли входные значения допустимыми диапазонами дат.

```
isDateRange('PT30M')
isDateRange('2012-02')
```

И возвращаются следующие результаты:

- false
- true

### isDateTime

Возвращает  true , если полученные входные данные являются строкой с меткой времени в формате UTC ISO (YYYY-MM-DDTHH:mm:ss.fffZ).

```
isDateTime('<input>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<input>	Да	any	Входное значение для проверки.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<Boolean-result>	Логическое	Возвращает <code>true</code> , если полученные входные данные являются строкой с меткой времени в формате UTC ISO, или <code>false</code> в противном случае.

### Примеры

В следующих примерах производится проверка, является ли входное значение строкой в формате UTC ISO.

```
isDateTime('hello world!')
isDateTime('2019-03-01T00:00:00.000Z')
```

И возвращаются следующие результаты соответственно:

- входные данные представляют собой строку, поэтому функция возвращает `false`;
- входные данные представляют собой строку в формате UTC ISO, поэтому функция возвращает `true`.

### isDefinite

Возвращает значение `true`, если полученный объект TimexProperty или выражение Timex указывает на допустимую дату. Допустимыми считаются даты, в которых определен год, месяц и день месяца.

```
isDefinite('<input>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<input>	Да	Объект или строка	Входные данные в формате объекта TimexProperty или строки выражения Timex.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<boolean-result>	Логическое	Возвращает <code>true</code> , если полученные входные данные определяют допустимую полную дату, или <code>false</code> в противном случае.

### Примеры

Предположим, у вас есть объект TimexProperty `validFullDate = new TimexProperty("2020-02-20")`, а свойство `Now` имеет значение `true`. В следующих примерах производится проверка, определяет ли этот объект допустимую полную дату.

```
isDefinite('xxxx-12-21')
isDefinite(validFullDate)
```

И возвращаются следующие результаты соответственно:

- `false`
- `true`

### isDuration

Возвращает значение `true`, если полученный объект TimexProperty или выражение Timex указывает на допустимую длительность времени.

```
isDuration('<input>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<code>&lt;input&gt;</code>	Да	Объект или строка	Входные данные в формате объекта TimexProperty или строки выражения Timex.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<code>&lt;boolean-result&gt;</code>	Логическое	Возвращает значение <code>true</code> , если входные данные определяют допустимую длительность, или <code>false</code> в противном случае.

### Примеры

В примерах ниже производится проверка, являются ли входные значения допустимыми отрезками времени.

```
isDuration('PT30M')
isDuration('2012-02')
```

И возвращаются следующие результаты соответственно:

- `true`
- `false`

### isFloat

Возвращает значение `true`, если входное значение является числом с плавающей запятой. Для сохранения согласованности между C# и JavaScript числом с плавающей запятой будет считаться любое число с ненулевым остатком от целочисленного деления на 1.

```
isFloat('<input>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<input>	Да	any	Входное значение для проверки.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<Boolean-result>	Логическое	Возвращает значение <code>true</code> , если входные данные содержат число с плавающей запятой, или <code>false</code> в противном случае.

### Примеры

В следующих примерах производится проверка, является ли входное значение числом с плавающей запятой.

```
isFloat('hello world!')  
isFloat(1.0)  
isFloat(12.01)
```

И возвращаются следующие результаты соответственно:

- входные данные представляют собой строку, поэтому функция возвращает `false`;
- входные данные представляют собой число, у которого остаток от деления равен 0, поэтому функция возвращает `false`;
- входные данные представляют собой число с плавающей запятой, поэтому функция возвращает `true`.

### isInteger

Возвращает значение `true`, если входное значение является целым числом. Для сохранения согласованности между C# и JavaScript целым числом будет считаться любое число с нулевым остатком от целочисленного деления на 1.

```
isInteger('<input>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<input>	Да	any	Входное значение для проверки.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<Boolean-result>	Логическое	Является ли входное значение целым числом.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ

### Примеры

В следующих примерах производится проверка, является ли входное значение целым числом.

```
isInteger('hello world!')
isInteger(1.0)
isInteger(12)
```

И возвращаются следующие результаты соответственно:

- входные данные представляют собой строку, поэтому функция возвращает `false`;
- входные данные представляют собой число, у которого остаток от деления равен 0, поэтому функция возвращает `true`;
- входные данные представляют собой целое число, поэтому функция возвращает `true`.

### isMatch

Возвращает `true`, если полученная строка соответствует указанному шаблону регулярного выражения.

```
isMatch('<target**string>', '<pattern>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<code>&lt;target**string&gt;</code>	Да	строка	Строка для сопоставления.
<code>&lt;pattern&gt;</code>	Да	строка	Шаблон регулярного выражения.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<code>&lt;Boolean-result&gt;</code>	Логическое	Возвращает <code>true</code> , если полученная строка соответствует стандартному шаблону регулярного выражения, или <code>false</code> в противном случае.

### Примеры

В следующем примере проверяется, соответствует ли входное значение заданному шаблону регулярного выражения.

```
isMatch('ab', '^[a-z]{1,2}$')
isMatch('FUTURE', '(?i)fortune|future')
isMatch('12abc34', '([0-9]+)([a-z]+)([0-9]+)')
isMatch('abacaxc', 'ab.*?c')
```

И возвращается то же значение `true`.

## isObject

Возвращает `true`, если входные данные содержат составной объект, или `false` в случае примитива. К примитивам относятся строки, числа и логические значения, а составными считаются типы, содержащие свойства, например классы.

```
isObject('<input>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<code>&lt;input&gt;</code>	Да	any	Входное значение для проверки.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<code>&lt;Boolean-result&gt;</code>	Логическое	Возвращает <code>true</code> , если входные данные содержат составной объект, или <code>false</code> в случае примитива.

### Примеры

В следующих примерах производится проверка, является ли входное значение объектом.

```
isObject('hello world!')  
isObject({userName: "Sam"})
```

И возвращаются следующие результаты соответственно:

- входные данные представляют собой строку, поэтому функция возвращает `false`;
- входные данные представляют собой объект, поэтому функция возвращает `true`.

## IsPresent

Возвращает значение `true`, если полученный объект TimexProperty или выражение Timex указывает на настоящее время.

```
isPresent('<input>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<code>&lt;input&gt;</code>	Да	Объект или строка	Входные данные в формате объекта TimexProperty или строки выражения Timex.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
-----------------------	-----	----------

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<boolean-result>	Логическое	Возвращает <code>true</code> , если входные данные обозначают текущий момент, или <code>false</code> в противном случае.

**Примеры** Предположим, у вас есть объект TimexProperty `validNow = new TimexProperty() { Now = true }`, а свойство `Now` имеет значение `true`. В примерах ниже производится проверка, указывают ли входные значения на текущий момент.

```
isPresent('PT30M')
isPresent(validNow)
```

И возвращаются следующие результаты соответственно:

- `false`
- `true`

### isString

Возвращает `true`, если входное значение является строкой.

```
isString('<input>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<input>	Да	any	Входное значение для проверки.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<Boolean-result>	Логическое	Возвращает <code>true</code> , если получено строковое входное значение, или <code>false</code> в противном случае.

### Примеры

В следующем примере производится проверка, является ли входное значение строкой.

```
isString('hello world!')
isString(3.14)
```

И возвращаются следующие результаты соответственно:

- входные данные представляют собой строку, поэтому функция возвращает `true`;
- входные данные представляют собой число с плавающей запятой, поэтому функция возвращает `false`.

### isTime

Возвращает значение `true`, если полученный объект TimexProperty или выражение Timex указывает на допустимое время. Допустимым считается время, для которого определены час, минута и секунда.

```
isTime('<input>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<code>&lt;input&gt;</code>	Да	Объект или строка	Входные данные в формате объекта TimexProperty или строки выражения Timex.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<code>&lt;boolean-result&gt;</code>	Логическое	Возвращает значение <code>true</code> , если входные данные определяют допустимое время, или <code>false</code> в противном случае.

### Примеры

В этих примерах производится проверка, указывают ли входные значения на допустимое время.

```
isTime('PT30M')
isTime('2012-02-21T12:30:45')
```

И возвращаются следующие результаты соответственно:

- `false`
- `true`

### isTimeRange

Возвращает значение `true`, если полученный объект TimexProperty или выражение Timex указывает на допустимый диапазон времени. Допустимыми считаются диапазоны времени, в которых определено время дня.

```
isTime('<input>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<code>&lt;input&gt;</code>	Да	Объект или строка	Входные данные в формате объекта TimexProperty или строки выражения Timex.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
-----------------------	-----	----------

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<boolean-result>	Логическое	Возвращает значение <code>true</code> , если входные данные определяют допустимый диапазон времени, или <code>false</code> в противном случае.

### Примеры

Предположим, у вас есть объект TimexProperty `validTimeRange = new TimexProperty() { PartOfDay = "morning" }`, а свойство `Now` имеет значение `true`. В этих примерах производится проверка, являются ли входные значения допустимыми диапазонами времени.

```
isTimeRange('PT30M')
isTimeRange(validTimeRange)
```

И возвращаются следующие результаты соответственно:

- `false`
- `true`

### join

Возвращает строку, содержащую все элементы из массива, где каждый символ отделен *разделителем*.

```
join([<collection>], '<delimiter>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<collection>	Да	массив	Массив, который содержит элементы для объединения
<delimiter>	Да	строка	Разделитель, содержащийся между каждым символом в результирующей строке

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<симв.1><разделитель><симв.2> <разделитель>...	строка	Полученная строка, созданная из всех элементов в указанном массиве

### Пример

В этом примере создается строка из всех элементов этого массива с указанным символом . в качестве разделителя.

```
join(createArray('a', 'b', 'c'), '.')
```

И возвращается результат a.b.c.

## jPath

В указанном объекте или строке JSON проверяет наличие узлов или значений, которые соответствуют выражению пути, и возвращает найденные узлы.

jPath(<json>, '<path>')			
ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<json>	Да	any	Объект или строка JSON для поиска узлов или значений, которые соответствуют значению выражения пути.
<path>	Да	any	Выражение пути, используемое для поиска соответствующих узлов или значений JSON.
ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ	
[ <json-node> ]	массив	Список узлов или значений JSON, которые соответствуют указанному выражению пути.	

### Пример для C#

Предположим, у вас есть следующий объект JSON.

```
{  
    "Stores": [  
        "Lambton Quay",  
        "Willis Street"  
    ],  
    "Manufacturers": [  
        {  
            "Name": "Acme Co",  
            "Products": [  
                {  
                    "Name": "Anvil",  
                    "Price": 50  
                }  
            ]  
        },  
        {  
            "Name": "Contoso",  
            "Products": [  
                {  
                    "Name": "Elbow Grease",  
                    "Price": 99.95  
                },  
                {  
                    "Name": "Headlight Fluid",  
                    "Price": 4  
                }  
            ]  
        }  
    ]  
}
```

И выражение пути `$..Products[?(@.Price >= 50)].Name`.

```
jPath(jsonStr, path)
```

В этом примере возвращается результат `["Anvil", "Elbow Grease"]`.

*Пример для JavaScript*

Предположим, у вас есть следующий объект JSON.

```
{
  "automobiles": [
    {
      "maker": "Nissan",
      "model": "Teana",
      "year": 2011
    },
    {
      "maker": "Honda",
      "model": "Jazz",
      "year": 2010
    },
    {
      "maker": "Honda",
      "model": "Civic",
      "year": 2007
    },
    {
      "maker": "Toyota",
      "model": "Yaris",
      "year": 2008
    },
    {
      "maker": "Honda",
      "model": "Accord",
      "year": 2011
    }
  ],
  "motorcycles": [
    {
      "maker": "Honda",
      "model": "ST1300",
      "year": 2012
    }
  ]
}
```

И значение пути `.automobiles{.maker === "Honda" && .year > 2009}.model`.

```
jPath(jsonStr, path)
```

В этом примере возвращается результат `['Jazz', 'Accord']`.

## json

Возвращает значение с типом JSON (нотация объектов JavaScript), объект с типом строки или XML.

```
json('<value>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<code>&lt;value&gt;</code>	Да	Строка или XML	Строка или XML для преобразования

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
-----------------------	-----	----------

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<результат JSON>	строка	Результирующий объект JSON, созданный из заданной строки или XML.

### Пример 1

В этом примере строка преобразуется в JSON.

```
json('{"fullName": "Sophia Owen"}')
```

И возвращается результат.

```
{
    "fullName": "Sophia Owen"
}
```

### Пример 2

В этом примере строка XML преобразуется в JSON.

```
json(xml('<?xml version="1.0"?> <root> <person id="1"> <name>Sophia Owen</name>
<occupation>Engineer</occupation> </person> </root>'))
```

И возвращается результат.

```
{
    "?xml": { "@version": "1.0" },
    "root": {
        "person": [ {
            "@id": "1",
            "name": "Sophia Owen",
            "occupation": "Engineer"
        } ]
    }
}
```

## last

Возвращает последний элемент из коллекции.

```
last('<collection>')
last([<collection>])
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<collection>	Да	строка или массив	Коллекция, в которой нужно найти последний элемент.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<последний элемент коллекции>	Строка или массив соответственно.	Последний элемент в коллекции

### Пример

Эти примеру ищут последний элемент в этих коллекциях:

```
last('abcd')
last(createArray(0, 1, 2, 3))
```

И возвращаются следующие результаты соответственно.

- d
- 3

### lastIndexOf

Возвращает начальную позицию или значение индекса последнего вхождения подстроки. Эта функция не учитывает регистр, а значения индексов начинаются с 0.

```
lastIndexOf('<text>', '<searchText>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<text>	Да	строка или массив	Строки, которая содержит подстроку для поиска
<searchText>	Да	строка	Подстрока для поиска

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<конечное значение индекса>	Целое число	Начальная позиция или значение индекса последнего вхождения указанной подстроки. Если строка не найдена, возвращается число -1.

### Пример 1

В этом примере выполняется поиск начального значения индекса для последнего вхождения подстроки **world** в строке **hello world**.

```
lastIndexOf('hello world', 'world')
```

И возвращается результат 6.

### Пример 2

В этом примере выполняется поиск начального значения индекса для последнего вхождения подстроки **def** в массиве **['abc', 'def', 'ghi', 'def']**.

```
lastIndexOf(createArray('abc', 'def', 'ghi', 'def'), 'def')
```

И возвращается результат 3.

## length

Возвращает длину строки.

```
length('<str>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<str>	Да	строка	Строка, для которой вычисляется длина.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<length>	Целое число	Длина полученной строки.

## Примеры

В этих примерах вычисляется длина строк.

```
length('hello')
length('hello world')
```

И возвращаются следующие результаты соответственно.

- 5
- 11

## less

Проверяет, является ли первое значение меньшим, чем второе. Возвращает `true`, если первое значение меньше второго, или `false` в противном случае.

```
less(<value>, <compareTo>)
less('<value>', '<compareTo>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<value>	Да	Целое число, число с плавающей запятой или строка.	Первое значение, для которого выполняется проверка того, является ли оно меньшим, чем второе.
<compareTo>	Да	Целое число, число с плавающей запятой или строка соответственно.	Элемент для сравнения

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
true или false	Логическое	Возвращает <code>true</code> , если первое значение меньше второго. Возвращает <code>false</code> , если первое значение больше второго или равно ему.

### Примеры

В этих примерах проверяется, является ли первое значение меньшим, чем второе.

```
less(5, 10)
less('banana', 'apple')
```

И возвращаются следующие результаты соответственно:

- `true`
- `false`

### lessOrEquals

Проверяет, является ли первое значение меньшим, чем второе, или равным ему. Возвращает `true`, если первое значение меньше второго или равно ему, или `false` в противном случае.

```
lessOrEquals(<value>, <compareTo>)
lessOrEquals('<value>', '<compareTo>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<code>&lt;value&gt;</code>	Да	Целое число, число с плавающей запятой или строка.	Проверяет, является ли первое значение меньшим, чем второе, или равным ему
<code>&lt;compareTo&gt;</code>	Да	Целое число, число с плавающей запятой или строка соответственно.	Элемент для сравнения

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
true или false	Логическое	Возвращает <code>true</code> , если первое значение меньше второго или равно ему. Возвращает <code>false</code> , если первое значение больше второго.

### Пример

В этих примерах проверяется, является ли первое значение меньшим или равным второму.

```
lessOrEquals(10, 10)
lessOrEquals('apply', 'apple')
```

И возвращаются следующие результаты соответственно:

- `true`
- `false`

### max

Возвращает наибольшее значение из списка или массива. Оба конца списка или массива включаются в область действия.

```
max(<number1>, <number2>, ...)
max([<number1>, <number2>, ...])
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<code>&lt;number1&gt;, &lt;number2&gt;, ...</code>	Да	number	Набор чисел, из которого требуется получить наибольшее значение
<code>[&lt;number1&gt;, &lt;number2&gt;, ...]</code>	Да	Массив чисел.	Массив чисел, из которого требуется получить наибольшее значение

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<code>&lt;макс. значение&gt;</code>	number	Максимальное значение в указанном массиве или набор чисел

### Примеры

В этих примерах извлекается наибольшее значение из набора чисел и массива:

```
max(1, 2, 3)
max(createArray(1, 2, 3))
```

И возвращается результат 3.

### Min

Возвращает наименьшее значение из набора чисел или массива.

```
min(<number1>, <number2>, ...)
min([<number1>, <number2>, ...])
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<number1>, <number2>, ...	Да	number	Набор чисел, из которого требуется получить наименьшее значение
[<number1>, <number2>, ...]	Да	Массив чисел.	Массив чисел, из которого требуется получить наименьшее значение

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<мин. значение>	number	Минимальное значение в указанном массиве или набор чисел.

### Примеры

В этих примерах извлекается наименьшее значение из набора чисел и массива:

```
min(1, 2, 3)
min(createArray(1, 2, 3))
```

И возвращается результат 1.

### mod

Возвращает остаток результата деления двух чисел. Чтобы получить целочисленный результат, см. раздел [div\(\)](#).

```
mod(<dividend>, <divisor>)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<dividend>	Да	number	Число, которое нужно поделить на <i>делитель</i>
<divisor>	Да	number	Число, на которое делится <i>делимое</i> . Не может иметь значение 0.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<остаток>	number	Остаток результата деления первого числа на второе

### Пример

В этом примере первое число делится на второе:

```
mod(3, 2)
```

И возвращается результат 1.

### **month**

Возвращает месяц из указанной метки времени.

```
month('<timestamp>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<timestamp>	Да	строка	Строка, содержащая метку времени

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<number-of-month>	Целое число	Число, обозначающее месяц, для указанной метки времени.

*Пример*

```
month('2018-03-15T13:01:00.000Z')
```

И возвращается результат 3.

### **mul**

Возвращает результат умножения двух чисел.

```
mul(<multiplicand1>, <multiplicand2>)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<multiplicand1>	Да	Целое число или число с плавающей запятой.	Число для умножения на <i>multiplicand2</i>
<multiplicand2>	Да	Целое число или число с плавающей запятой.	Число, на которое умножается <i>multiplicand1</i>

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<результат>	Целое число или число с плавающей запятой.	Произведение от умножения первого числа на второе

*Примеры*

В этом примере первое число умножается на второе:

```
mul(1, 2)
mul(1.5, 2)
```

И возвращаются следующие результаты соответственно:

- 2
- 3

### **newGuid**

Возвращает строку с новым GUID.

```
newGuid()
```

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<Guid-string>	строка	Новая строка в формате GUID, которая имеет длину 36 символов и примерно такой формат: xxxxxxxx-xxxx-4xxx-xxxx-xxxxxxxxxxxx

### *Пример*

```
newGuid()
```

Возвращается результат следующего вида: xxxxxxxx-xxxx-4xxx-xxxx-xxxxxxxxxxxx.

### **not**

Проверяет, имеет ли выражение значение false. Возвращает `true`, если выражение ложно, или `false` в противном случае.

```
not(<expression>)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<expression>	Да	Логическое	Выражение для вычисления

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
true или false	Логическое	Возвращает <code>true</code> , если выражение ложно. Возвращает <code>false</code> , если выражение истинно.

### *Пример 1*

В этих примерах проверяется, все ли указанные выражения ложны:

```
not(false)  
not(true)
```

И возвращаются следующие результаты соответственно:

- выражение имеет значение false, поэтому функция возвращает `true`.
- выражение имеет значение true, поэтому функция возвращает `false`.

### Пример 2

В этих примерах проверяется, все ли указанные выражения ложны:

```
not(equals(1, 2))  
not(equals(1, 1))
```

И возвращаются следующие результаты соответственно:

- выражение имеет значение false, поэтому функция возвращает `true`.
- выражение имеет значение true, поэтому функция возвращает `false`.

### или диспетчер конфигурации служб

Проверяет, является ли хотя бы одно выражение истинным. Возвращает значение `true`, если хотя бы одно выражение истинно, или `false`, если все являются ложными.

```
or(<expression1>, <expression2>, ...)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<code>&lt;expression1&gt;</code> , <code>&lt;expression2&gt;</code> , ...	Да	Логическое	Выражения, которые следует проверить

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<code>true</code> или <code>false</code>	Логическое	Возвращает значение <code>true</code> , если хотя бы одно выражение истинно. Возвращает значение <code>false</code> , если все выражения ложны.

### Пример 1

В этих примерах проверяется, является ли хотя бы одно выражение истинным:

```
or(true, false)  
or(false, false)
```

И возвращаются следующие результаты соответственно:

- по крайней мере одно выражение имеет значение `true`, поэтому функция возвращает `true`.
- оба выражения имеют значения `false`, поэтому функция возвращает `false`.

### Пример 2

В этих примерах проверяется, является ли хотя бы одно выражение истинным:

```
or>equals(1, 1), equals(1, 2))  
or>equals(1, 2), equals(1, 3))
```

И возвращаются следующие результаты соответственно:

- по крайней мере одно выражение имеет значение true, поэтому функция возвращает `true`.
- оба выражения имеют значения false, поэтому функция возвращает `false`.

### rand

Возвращает случайное целое число из указанного диапазона с включительным значением только в начале.

```
rand(<minValue>, <maxValue>)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<minValue>	Да	Целое число	Наименьшее целое число в диапазоне
<maxValue>	Да	Целое число	Целое число, следующее за наибольшим целым числом в диапазоне, которое функция может вернуть

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<случайный результат>	Целое число	Случайное целое число, возвращаемое из указанного диапазона

### Пример

Этот пример получает случайное целое число из указанного диапазона, исключая максимальное значение:

```
rand(1, 5)
```

И возвращается результат 1, 2, 3 или 4.

### range

Возвращает массив целых чисел, который начинается с заданного целого числа.

```
range(<startIndex>, <count>)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<startIndex>	Да	Целое число	Целочисленное значение, которое является первым элементом массива.
<count>	Да	Целое число	Количество целых чисел в массиве

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<range-result>	Целое число	Массив с целыми числами, который начинается с указанного индекса

### Пример

В этом примере создается целочисленный массив, который начинается с указанного индекса 1 и имеет указанное число целых чисел: 4.

```
range(1, 4)
```

И возвращается результат [1, 2, 3, 4].

### removeProperty

Удаляет свойство из объекта и возвращает обновленный объект.

```
removeProperty(<object>, '<property>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<object>	Да	объект	Объект JSON, из которого вы хотите удалить свойство.
<property>	Да	строка	Имя удаляемого свойства.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<обновленный_объект>	объект	Обновленный объект JSON без указанного свойства

### Пример

В этом примере свойство `accountLocation` удаляется из объекта `customerProfile`, который преобразуется в JSON с помощью функции `json()`, и возвращается обновленный объект.

```
removeProperty(json('customerProfile'), 'accountLocation')
```

## replace

Заменяет подстроку указанной строкой и возвращает полученную строку. Эта функция учитывает регистр.

```
replace('<text>', '<oldText>', '<newText>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<text>	Да	строка	Строка, которая содержит заменяемую подстроку
<oldText>	Да	строка	Заменяемая подстрока
<newText>	Да	строка	Строка для замены

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<обновленный текст>	строка	Обновленная строка после замены подстроки. Если подстрока не найдена, функция возвращает исходную строку.

### Пример 1

В этом примере выполняется поиск подстроки **old** в строке **the old string** и заменяет **old** на **new**:

```
replace('the old string', 'old', 'new')
```

В качестве результата возвращается строка **the new string**.

### Пример 2

При использовании escape-символов обработчик выражений обрабатывает их самостоятельно. Эта функция заменяет строки правильными escape-символами.

```
replace('hello\"', '\"', '\n')
replace('hello\n', '\n', '\\\\')
@"replace('hello\\', '\\', '\\\\\\')"
@"replace('hello\\n', '\n', '\\\\\\")"
```

И возвращаются следующие результаты соответственно.

- hello\n
- hello\\
- @"hello\\\"
- @"hello\\\"

## replacelgnoreCase

Заменяет подстроку указанной строкой и возвращает полученную строку. Эта функция не учитывает регистр.

```
replaceIgnoreCase('<text>', '<oldText>', '<newText>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<text>	Да	строка	Строка, которая содержит заменяемую подстроку
<oldText>	Да	строка	Заменяемая подстрока
<newText>	Да	строка	Строка для замены

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<обновленный текст>	строка	Обновленная строка после замены подстроки. Если подстрока не найдена, возвращает исходную строку.

### Пример

В этом примере выполняется поиск подстроки **old** в строке **the old string** и заменяет **old** на **new**:

```
replace('the old string', 'old', 'new')
```

В результате возвращается строка **the new string**.

### round

Округляет значение до ближайшего целого или до указанного числа дробных разрядов.

```
round('<number>', '<precision-digits>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<number>	Да	number	Входной номер
<точность-цифры>	Нет	Целое число	Указанное число цифр дробной части. Значение по умолчанию равно 0.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<Возвращаемое значение>	number	Возвращаемое значение входных данных, округленное на указанное число цифр дробной части

### Пример 1

В этом примере округляется число 10,333:

```
round(10.333)
```

И возвращает число 10.

### Пример 2

Этот пример Округляет число 10,3313 до 2 цифр дробной части:

```
round(10.3313, 2)
```

И возвращает число 10,33.

### select

Обрабатывает каждый элемент и возвращают новую коллекцию преобразованных элементов.

```
select([<collection-instance>], <iteratorName>, <function>)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<collection-instance>	Да	массив	Коллекция с элементами.
<iteratorName>	Да	Имя итератора.	Ключевой элемент
<function>	Да	expression	Функция, которая содержит <code>iteratorName</code> .

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<new-collection>	массив	Новая коллекция, в которой к каждому элементу применено преобразование указанной функцией.

### Пример 1

В этом примере создается новая коллекция.

```
select(createArray(0, 1, 2, 3), x, x + 1)
```

И возвращается результат [1, 2, 3, 4].

### Пример 2

В этих примерах создается новая коллекция:

```
select(json("{'name': 'jack', 'age': '15'}"), x, concat(x.key, ':', x.value))
select(json("{'name': 'jack', 'age': '15'}"), x=> concat(x.key, ':', x.value))
```

И возвращают результат ["имя: гнездо", "Age: 15"]. Обратите внимание, что второе выражение является лямбда-выражением, которое может оказаться более удобочитаемым.

### сентенцекасе

Сделать прописной первую букву первого слова в строке в необязательном формате языкового стандарта.

```
sentenceCase('<text>', '<locale>'?)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<text>	Да	строка	Исходная строка
<языковой стандарт>	нет	строка	Дополнительный языковой стандарт сведений о культуре

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
Результатирующая строка	строка	Возвращает результат предложения Case

#### Пример 1

В этих примерах первая буква в строке заменяется на прописные:

```
sentenceCase('a')
sentenceCase('abc def')
sentenceCase('aBC dEF')
```

И возвращаются следующие результаты соответственно:

- A
- ABC DEF
- ABC DEF

#### Пример 2

В этих примерах первая буква в строке задается в указанном формате языкового стандарта:

```
sentenceCase('a', 'fr-FR')
sentenceCase('abc', 'en-US')
sentenceCase('aBC', 'fr-FR')
```

И возвращаются следующие результаты соответственно:

- A
- ABC
- ABC

### setPathToValue

Извлекает значение указанного свойства из объекта JSON.

```
setPathToValue(<path>, <value>)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<Path>	Да	объект	Путь, который нужно задать.
<value>	Да	объект	Значение, которое нужно присвоить этому пути.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
value	объект	Задаваемое значение.

### Пример 1

В приведенном ниже примере задается значение пути 1.

```
setPathToValue(path.x, 1)
```

И возвращается результат 1. `path.x` получает значение 1.

### Пример 2

В примере ниже присваивается значение.

```
setPathToValue(path.array[0], 7) + path.array[0]
```

И возвращается результат 14.

## setProperty

Задает значение свойства объекта и возвращает обновленный объект. Чтобы добавить новое свойство, используйте эту функцию или функцию [addProperty\(\)](#).

```
setProperty(<object>, '<property>', <value>)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<object>	Да	объект	Объект JSON, для которого вы хотите задать свойство.
<property>	Да	строка	Имя указываемого свойства.
<value>	Да	any	Значение, задаваемое для указанного свойства

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<обновленный_объект>	объект	Обновленный объект JSON, свойство которого задается

### Пример

В этом примере указывается свойство `accountNumber` в объекте `customerProfile`, который преобразуется в JSON с помощью функции `json()`. Функция присваивает значение, которое формируется функцией `newGuid()`, и возвращает обновленный объект JSON.

```
setProperty(json('customerProfile'), 'accountNumber', newGuid())
```

### skip

Удаляет элементы из начала коллекции и возвращает все оставшиеся элементы.

```
skip([<collection>], <count>)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<collection>	Да	массив	Коллекция, элементы которой требуется удалить
<count>	Да	Целое число	Положительное целое число для количества элементов в начале коллекции, которые требуется удалить

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<обновленная коллекция>	массив	Обновленная коллекция после удаления указанных элементов.

### Пример

В этом примере удаляется один элемент, число `1`, из начала указанного массива.

```
skip(createArray(0, 1, 2, 3), 1)
```

И возвращается массив с остальными его элементами: `[1,2,3]`.

### sortBy

Сортирует элементы коллекции по возрастанию и возвращает упорядоченную коллекцию.

```
sortBy([<collection>], '<property>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<collection>	Да	строка или массив	Коллекция для сортировки.
<property>	Нет	строка	Сортирует элементы объектов в коллекции по указанному свойству, если оно задано.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<new-collection>	массиве	Новая коллекция с отсортированными элементами.

### Пример 1

В этом примере сортируется следующая коллекция.

```
sortBy(createArray(1, 2, 0, 3))
```

И возвращается результат [0, 1, 2, 3] .

### Пример 2

Предположим, что у вас есть следующая коллекция.

```
{
  'nestedItems': [
    {'x': 2},
    {'x': 1},
    {'x': 3}
  ]
}
```

В этом примере создается новая коллекция, отсортированная по значению свойства объекта x.

```
sortBy(nestedItems, 'x')
```

И возвращается результат.

```
{
  'nestedItems': [
    {'x': 1},
    {'x': 2},
    {'x': 3}
  ]
}
```

### sortByDescending

Сортирует элементы коллекции по убыванию и возвращает упорядоченную коллекцию.

```
sortBy([<collection>], '<property>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<collection>	Да	строка или массив	Коллекция для сортировки.
<property>	Нет	строка	Сортирует элементы объектов в коллекции по указанному свойству, если оно задано.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<new-collection>	массив	Новая коллекция с отсортированными элементами.

### Пример 1

В этом примере создается новая отсортированная коллекция.

```
sortByDescending(createArray(1, 2, 0, 3))
```

И возвращается результат [3, 2, 1, 0].

### Пример 2

Предположим, что у вас есть следующая коллекция.

```
{
  'nestedItems': [
    {'x': 2},
    {'x': 1},
    {'x': 3}
  ]
}
```

В этом примере создается новая коллекция, отсортированная по значению свойства объекта x.

```
sortByDescending(nestedItems, 'x')
```

Возвращается такой результат:

```
{
  'nestedItems': [
    {'x': 3},
    {'x': 2},
    {'x': 1}
  ]
}
```

**split**

Возвращает массив, содержащий подстроки, разделенные запятыми, основываясь на указанном символе разделителя в исходной строке.

```
split('<text>', '<delimiter>';?)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<text>	Да	строка	Строка для разделения на подстроки в зависимости от указанного разделителя в исходной строке. Если текст имеет значение NULL, он будет считаться пустой строкой.
<delimiter>	Нет	строка	Символ в исходной строке для использования в качестве разделителя. Если разделитель не предоставлен или имеет значение NULL, по умолчанию используется пустая строка.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
[<substring1>, <substring2>, ...]	массив	Массив, содержащий подстроки из исходной строки, разделенные запятыми

### Примеры

В этом примере создается массив подстрок из указанной строки путем разбивки с использованием указанного символа разделителя.

```
split('a**b**c', '**')
split('hello', '')
split('', 'e')
split('', '')
split('hello')
```

В качестве результатов возвращаются следующие массивы соответственно.

- ["a", "b", "c"]
- ["h", "e", "l", "l", "o"]
- [""], []
- ["h", "e", "l", "l", "o"]

### startOfDay

Возврат начала дня для метки времени в необязательном формате языкового стандарта.

```
startOfDay('<timestamp>', '<format>?', '<locale>'?)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<timestamp>	Да	строка	Строка, содержащая метку времени
<format>	Нет	строка	Пользовательский шаблон формата. По умолчанию для метки времени используется формат UTC ISO (YYYY-MM-DDTHH:mm:ss.fffZ), который соответствует стандарту <a href="#">ISO 8601</a> .
<языковой стандарт>	нет	строка	Дополнительный языковой стандарт сведений о культуре

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
'<updated-timestamp>'	строка	Обновленная метка времени, которая соответствует началу нулевого часа того же дня.

### Пример 1

В этом примере выполняется поиск начала дня.

```
startOfDay('2018-03-15T13:30:30.000Z')
```

И возвращается результат 2018-03-15T00:00:00.000Z.

### Пример 2

В этом примере выполняется поиск начала дня с локальным языком fr-FR:

```
startOfDay('2018-03-15T13:30:30.000Z', '', 'fr-FR')
```

И возвращает результат 15/03/2018 00:00:00.

## startOfHour

Возврат начала часа для отметки времени в необязательном формате языкового стандарта.

```
startOfHour('<timestamp>', '<format>?', '<locale>'?)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<timestamp>	Да	строка	Строка, содержащая метку времени

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<format>	Нет	строка	Пользовательский шаблон формата. По умолчанию для метки времени используется формат UTC ISO (YYYY-MM-DDTHH:mm:ss.fffZ), который соответствует стандарту <a href="#">ISO 8601</a> .
<языковой стандарт>	нет	строка	Дополнительный языковой стандарт сведений о культуре

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
'<updated-timestamp>'	строка	Указанная метка времени, которая соответствует началу нулевой минуты дня.

### Пример 1

В этом примере выполняется поиск начала часа.

```
startOfHour('2018-03-15T13:30:30.000Z')
```

И возвращается результат 2018-03-15T13:00:00.000Z.

### Пример 2

В этом примере выполняется поиск начала часа с локальным языком fr-FR:

```
startOfHour('2018-03-15T13:30:30.000Z', '', 'fr-FR')
```

И возвращает результат 15/03/2018 13:00:00.

### startOfMonth

Возврат начала месяца для отметки времени в необязательном формате языкового стандарта.

```
startOfMonth('<timestamp>', '<format>?', '<locale>?')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<timestamp>	Да	строка	Строка, содержащая метку времени

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<format>	Нет	строка	Пользовательский шаблон формата. По умолчанию для метки времени используется формат UTC ISO (YYYY-MM-DDTHH:mm:ss.fffZ), который соответствует стандарту <a href="#">ISO 8601</a> .
<языковой стандарт>	нет	строка	Дополнительный языковой стандарт сведений о культуре

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
'<updated-timestamp>'	строка	Указанная метка времени, которая соответствует началу нулевого часа первого дня месяца.

### Пример 1

В этом примере выполняется поиск начала месяца.

```
startOfMonth('2018-03-15T13:30:30.000Z')
```

И возвращается результат 2018-03-01T00:00:00.000Z.

### Пример 2

В этом примере выполняется поиск начала месяца с локальным языком fr-FR:

```
startOfMonth('2018-03-15T13:30:30.000Z', '', 'fr-FR')
```

И возвращает результат 01/03/2018 00:00:00.

### startsWith

Проверяет, начинается ли строка с определенной подстроки. Возвращает `true`, если подстрока найдена, или `false` в противном случае. Эта функция не учитывает регистр.

```
startsWith('<text>', '<searchText>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<text>	Да	строка	Проверяемая строка
<searchText>	Да	строка	Начальная подстрока для поиска.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
true или false	Логическое	Возвращает значение <code>true</code> , если обнаружена начальная подстрока. Возвращает <code>false</code> , если не обнаружена.

### Пример 1

В этом примере проверяется, начинается ли строка `hello world` строкой `hello`:

```
startsWith('hello world', 'hello')
```

И возвращается результат `true`.

### Пример 2

В этом примере проверяется, начинается ли строка `hello world` строкой `greeting`:

```
startsWith('hello world', 'greeting')
```

И возвращается результат `false`.

## строка

Возврат строковой версии значения в необязательном формате языкового стандарта.

```
string(<value>, '<locale>'?)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<code>&lt;value&gt;</code>	Да	any	Значение, которое необходимо преобразовать
<code>&lt;языковой стандарт&gt;</code>	нет	строка	Дополнительный языковой стандарт сведений о культуре

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<code>&lt;строковое значение&gt;</code>	строка	Указанное значение в строковом формате.

### Пример 1

В этом примере создается строковое представление числа 10.

```
string(10)
```

И возвращается строковое значение 10.

## Пример 2

В этом примере создается строка для указанного объекта JSON и используется символ обратной косой черты \\ в качестве escape-символа для двойных кавычек " .

```
string( { "name": "Sophie Owen" } )
```

И возвращает результат {"Name": "Софи о'мэлли Owen"}

## Пример 3

В этих примерах создается строковый номер 10 в определенном языковом стандарте:

```
string(100.1, 'fr-FR')
string(100.1, 'en-US')
```

И возвращает следующие строки соответственно:

- 100, 1
- \* 100,1

### sub

Вычитает второе число из первого числа и возвращает результат.

```
sub(<minuend>, <subtrahend>)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<minuend>	Да	number	Число, из которого вычитается <i>вычитаемое</i>
<subtrahend>	Да	number	Число, которое вычитается из <i>уменьшаемого</i>

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<результат>	number	Вычитается второе число из первого числа и возвращается результат

## Пример

В этом примере из первого числа вычитается второе число:

```
sub(10.3, .3)
```

И возвращается результат 10.

### subArray

Возвращает подмассив на основе указанной начальной и конечной позиций. Значения индекса

начинаются с числа 0.

```
subArray(<Array>, <startIndex>, <endIndex>)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<array>	Да	массив	Массив, из которого нужно извлечь подмассив.
<startIndex>	Да	Целое число	Положительное число (или 0), которое обозначает начальную позицию или значение индекса.
<endIndex>	Да	Целое число	Положительное число (или 0), которое обозначает конечную позицию или значение индекса.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<subarray-result>	массив	Подмассив с указанным количеством элементов, начиная с указанной позиции индекса в исходной строке

### Пример

В этом примере создается подмассив из указанного массива.

```
subArray(createArray('H', 'e', 'l', 'l', 'o'), 2, 5)
```

И возвращается результат ["l", "l", "o"] .

### substring

Возвращает символы из строки, начиная с указанной позиции или индекса. Значения индекса начинаются с числа 0.

```
substring('<text>', <startIndex>, <length>)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<text>	Да	строка	Строка, из которой нужно извлечь подстроку.

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<startIndex>	Да	Целое число	Положительное число (или 0), которое обозначает начальную позицию или значение индекса для подмассива.
<length>	Да	Целое число	Положительное число символов для подмассива в подстроке.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<результат подстроки>	строка	Подстрока с указанным количеством символов, начиная с указанной позиции индекса в исходной строке

### Пример

В этом примере создается пятисимвольная подстрока из указанной строки, начиная со значения индекса 6:

```
substring('hello world', 6, 5)
```

И возвращается результат **world**.

### subtractFromTime

Вычтите количество единиц времени из метки времени в необязательном формате языкового стандарта. См. раздел [getPastTime\(\)](#).

```
subtractFromTime('<timestamp>', <interval>, '<timeUnit>', '<format>?', '<locale>?')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<timestamp>	Да	строка	Строка, содержащая метку времени
<interval>	Да	Целое число	Число единиц времени для вычитания
<timeUnit>	Да	строка	Единица измерения времени, которая будет применяться к значению <i>interval</i> . Поддерживаются следующие единицы: Second (секунда), Minute (минута), Hour (час), Day (день), Week (неделя), Month (месяц) и Year (год).

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<format>	Нет	строка	Пользовательский шаблон формата. По умолчанию для метки времени используется формат UTC ISO (YYYY-MM-DDTHH:mm:ss.fffZ), который соответствует стандарту <a href="#">ISO 8601</a> .
<языковой стандарт>	нет	строка	Дополнительный языковой стандарт сведений о культуре

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<обновленная_метка_времени>	строка	Метка времени, от которой отнято указанное число единиц времени

### Пример 1

Этот пример вычитает один день из следующей метки времени:

```
subtractFromTime('2018-01-02T00:00.000Z', 1, 'Day')
```

И возвращается результат 2018-01-01T00:00:00.000Z.

### Пример 2

В этом примере вычитается один день из метки времени с использованием формата D :

```
subtractFromTime('2018-01-02T00:00.000Z', 1, 'Day', 'D')
```

И возвращает результат **понедельник, Январь, 1, 2018.**

### Пример 3

В этом примере вычитается 1 час из метки времени в языковом стандарте de-DE :

```
subtractFromTime('2018-03-15T13:00:00.000Z', 1, 'Hour', '', 'de-DE')
```

И возвращает результат 15.03.18 12:00:00.

### Sum

Возвращает результат сложения чисел из списка.

```
sum([<list of numbers>])
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
[<list of numbers>]	Да	Массив чисел.	Числа для добавления

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<итоговая_сумма>	number	Возвращает результат сложения указанных чисел

### Пример

В этом примере добавляются указанные числа:

```
sum(createArray(1, 1.5))
```

И возвращается результат 2,5.

### take

Возвращает элементы, расположенные в начале коллекции.

```
take('<collection>', <count>)
take([<collection>], <count>)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<collection>	Да	строка или массив	Коллекция, элементы которой требуется получить
<count>	Да	Целое число	Положительное целое число для количества элементов в начале коллекции, которые требуется получить.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<подмножество> или [<подмножество>]	Строка или массив соответственно	Строка или массив, который содержит заданное число элементов, взятых из первой части исходной коллекции

### Пример

Эти примеры получают указанное количество элементов из первой части этих коллекций:

```
take('abcde', 3)
take(createArray(0, 1, 2, 3, 4), 3)
```

И возвращаются следующие результаты соответственно:

- abc
- [0, 1, 2]

### ticks

Возвращает значение свойства количества тактов для указанной метки времени. Один такт соответствует интервалу в 100 наносекунд.

```
ticks('<timestamp>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<timestamp>	Да	строка	Строка для метки времени

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<число тактов>	integer (bigint в JavaScript)	Число тактов с момента в указанной метке времени

### Пример

В этом примере метка времени преобразуется в значение количества тактов.

```
ticks('2018-01-01T08:00:00.000Z')
```

И возвращается результат 636503904000000000.

### тикстодайс

Преобразует значение свойства тактов в число дней.

```
ticksToDays('ticks')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<тактов>	Да	Целое число	Значение свойства тактов для преобразования

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<Количество дней>	number	Число дней, преобразованное из значения свойства тактов

### Пример

В этом примере значение свойства тактов преобразуется в число дней:

```
ticksToDays(2193385800000000)
```

И возвращает число 2538,64097222.

### тиксточаурс

Преобразуйте значение свойства тактов в число часов.

```
ticksToHours('ticks')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<тактов>	Да	Целое число	Значение свойства тактов для преобразования

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<количество часов>	number	Число часов, преобразованное из значения свойства тактов

### Пример

В этом примере значение свойства тактов преобразуется в число часов:

```
ticksToHours(2193385800000000)
```

И возвращает число 60927.38333333331.

### тикстоминутес

Преобразуйте значение свойства тактов в число минут.

```
ticksToMinutes('ticks')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<тактов>	Да	Целое число	Значение свойства тактов для преобразования

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<количество минут>	number	Число минут, преобразованное из значения свойства тактов

### Пример

В этом примере значение свойства тактов преобразуется в число минут:

```
ticksToMinutes(2193385800000000)
```

И возвращает число 3655643,0185.

#### : заглавный регистр

Сделать прописной первую букву каждого слова в строке в необязательном локальном формате.

```
titleCase('<text>', '<locale>?')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<text>	Да	строка	Исходная строка
<языковой стандарт>	нет	строка	Дополнительный языковой стандарт сведений о культуре

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
Результатирующая строка	строка	Результат варианта заголовка

#### Пример 1

В этих примерах первая буква каждого слова в строке заменяется на прописные:

```
titleCase('a')
titleCase('abc def')
titleCase('aBC dEF')
```

И возвращаются следующие результаты соответственно:

- A
- ABC DEF
- ABC DEF

#### Пример 2

Эти примеры заменяют первую букву строки в формате en-US :

```
titleCase('a', 'en-US')
titleCase('aBC dEF', 'en-US')
```

И возвращаются следующие результаты соответственно:

- A
- ABC DEF

#### toLower

Возврат строки в нижнем регистре в необязательном формате языкового стандарта. Если символ в строке не имеет версии в нижнем регистре, он добавляется в возвращаемую строку без изменений.

```
toLowerCase('<text>', '<locale>'?)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<text>	Да	строка	Строка, возвращаемая в нижнем регистре
<языковой стандарт>	нет	строка	Дополнительный языковой стандарт сведений о культуре

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<текст в нижнем регистре>	строка	Исходная строка в нижнем регистре

### Пример 1

В этом примере строка преобразуется в нижний регистр.

```
toLowerCase('Hello World')
```

И возвращается результат **hello world**.

### Пример 2

В этом примере строка преобразуется в нижний регистр в формате fr-FR :

```
toUpperCase('Hello World', 'fr-FR')
```

И возвращается результат **hello world**.

### toUpperCase

Возврат строки в верхнем регистре в необязательном формате языкового стандарта. Если символ в строке не имеет версии в верхнем регистре, он добавляется в возвращаемую строку без изменений.

```
toUpperCase('<text>', '<locale>'?)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<text>	Да	строка	Строка, возвращаемая в верхнем регистре
<языковой стандарт>	нет	строка	Дополнительный языковой стандарт сведений о культуре

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<текст в верхнем регистре>	строка	Исходная строка в верхнем регистре

### Пример 1

В этом примере строка преобразуется в верхний регистр.

```
toUpperCase('Hello World')
```

И возвращается результат HELLO WORLD.

### Пример 2

В этом примере строка преобразуется в верхний регистр в формате fr-FR :

```
toUpperCase('Hello World', 'fr-FR')
```

И возвращается результат HELLO WORLD.

### trim

Удаляет все начальные и конечные пробелы и возвращает обновленную строку.

```
trim('<text>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<text>	Да	строка	Строка, которая содержит начальные и конечные пробелы для удаления

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<обновленный текст>	строка	Обновленная версия исходной строки без начальных и конечных пробелов

### Пример

В этом примере удаляются начальные и конечные пробелы из строки " Hello World " .

```
trim(' Hello World ')
```

И возвращается сокращенный результат Hello World.

### union

Возвращает коллекцию, которая содержит все элементы из указанных коллекций. Чтобы появиться в результатах, элемент должен содержаться в любой коллекции, переданной этой функции. Если один или

несколько элементов имеют одинаковое имя, в результатах появляется последний элемент с таким именем.

```
union('<collection1>', '<collection2>', ...)  
union([<collection1>], [<collection2>], ...)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<collection1>, <collection2>, ...	Да	Массив или объект, но не оба типа сразу.	Коллекции, из которых нужно получить элементы.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<обновленная коллекция>	Массив или объект соответственно.	Коллекция, которая содержит все элементы из указанных коллекций. Дубликаты не включаются.

### Пример

В этом примере извлекаются все элементы из следующих коллекций:

```
union(createArray(1, 2, 3), createArray(1, 2, 10, 101))
```

И возвращается результат [1, 2, 3, 10, 101].

### unique

Удаляет из массива все дубликаты.

```
unique([<collection>])
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<collection>	Да	массив	Коллекция для преобразования.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<new-collection>	массив	Новая коллекция, из которой удалены все повторяющиеся элементы.

### Пример 1

В этом примере удаляются повторяющиеся элементы из следующего массива:

```
unique(createArray(1, 2, 1))
```

И возвращается результат [1, 2] .

### **uriComponent**

Возвращает двоичную версию компонента URI.

```
uriComponent('<value>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<value>	Да	строка	Строка для преобразования в формат закодированного URI

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<закодированный URI>	строка	Строка с закодированным URI, содержащая escape-символы

#### *Пример*

В этом примере для строки создается версия с закодированным URI.

```
uriComponent('https://contoso.com')
```

И возвращается результат http%3A%2F%2Fcontoso.com.

### **uriComponentToString**

Возвращает декодированную версию строки с закодированным URI.

```
uriComponentToString('<value>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<value>	Да	строка	Декодируемая строка с закодированным URI

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<двоичная версия закодированного URI>	строка	Декодированная версия строки с закодированным URI

#### *Пример*

В этом примере создается декодированная версия строки с закодированным URI.

```
uriComponentToString('http%3A%2F%2Fcontoso.com')
```

И возвращается результат <https://contoso.com>.

### uriHost

Возвращает значение узла из исходного значения URI.

```
uriHost('<uri>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<uri>	Да	строка	URI, из которого нужно получить значение узла.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<значение узла>	строка	Значение узла из указанного URI.

#### Пример

В этом примере выполняется поиск значение узла из следующего значения URI.

```
uriHost('https://www.localhost.com:8080')
```

И возвращается результат [www.localhost.com](http://www.localhost.com).

### uriPath

Возвращает значение пути из значения URI.

```
uriPath('<uri>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<uri>	Да	строка	URI, из которого нужно получить значение пути.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<значение пути>	строка	Значение пути из указанного URI.

#### Пример

В этом примере выполняется значения сегмента пути из следующего значения URI.

```
uriPath('http://www.contoso.com/catalog/shownew.htm?date=today')
```

И возвращается результат /catalog/shownew.htm.

### uriPathAndQuery

Возвращает значение пути и запроса из значения URI.

```
uriPathAndQuery('<uri>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<uri>	Да	строка	URI, из которого нужно получить значения пути и запроса.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<значение запроса пути>	строка	Значения пути и запроса из указанного URI.

### Пример

В этом примере выполняется поиск значений пути и запроса из следующего значения URI.

```
uriPathAndQuery('http://www.contoso.com/catalog/shownew.htm?date=today')
```

И возвращается результат /catalog/shownew.htm?date=today.

### uriPort

Возвращает значение порта из значения URI.

```
uriPort('<uri>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<uri>	Да	строка	URI, из которого нужно получить значение пути.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<значение порта>	строка	Значение порта из указанного URI.

### Пример

В этом примере выполняется поиск значения порта из следующего значения URI.

```
uriPort('http://www.localhost:8080')
```

И возвращается результат 8080.

### uriQuery

Возвращает значение запроса из значения URI.

```
uriQuery('<uri>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<uri>	Да	строка	URI, из которого нужно получить значение запроса.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<значение запроса>	строка	Значение запроса из указанного URI.

### Пример

В этом примере выполняется поиск значения запроса из следующего значения URI.

```
uriQuery('http://www.contoso.com/catalog/shownew.htm?date=today')
```

И возвращается результат `?date=today`.

### uriScheme

Возвращает значение схемы из значения URI.

```
uriScheme('<uri>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<uri>	Да	строка	URI, из которого нужно получить значение запроса.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<значение схемы>	строка	Значение схемы из указанного URI.

### Пример

В этом примере выполняется поиск значения схемы из следующего значения URI.

```
uriQuery('http://www.contoso.com/catalog/shownew.htm?date=today')
```

И возвращается результат `http`.

### utcnow

Возврат текущей метки времени в необязательном формате языкового стандарта в виде строки.

```
utcNow('<format>', '<locale>'?)
```

Кроме того, можно указать другой формат с помощью параметра *<format>*.

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<i>&lt;format&gt;</i>	Нет	строка	Пользовательский шаблон формата. По умолчанию для метки времени используется формат UTC ISO (YYYY-MM-DDTHH:mm:ss.fffZ), который соответствует стандарту ISO 8601.
<i>&lt;языковой стандарт&gt;</i>	нет	строка	Дополнительный языковой стандарт сведений о культуре

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<i>&lt;текущая метка времени&gt;</i>	строка	Текущая дата и время

### Пример 1

Предположим, что текущая дата — **15 апреля 2018 года**, а время — 13:00:00. В этом примере извлекается метка времени.

```
utcNow()
```

И возвращается результат 2018-04-15T13:00:00.000Z.

### Пример 2

Предположим, что текущая дата — **15 апреля 2018 года**, а время — 13:00:00. В этом примере извлекается текущая метка времени в альтернативном формате D.

```
utcNow('D')
```

И возвращается результат Sunday, April 15, 2018.

### Пример 3

Предположим, что текущая дата — **15 апреля 2018 года**, а время — 13:00:00. В этом примере возвращается текущая метка времени с **использованием локали de**:

```
utcNow('', 'de-DE')
```

И возвращает результат 15.04.18 13:00:00.

### где

Применяет фильтр к каждому элементу и возвращает новую коллекцию тех элементов, которые

соответствуют указанному условию.

```
where([<collection-instance>], <iteratorName>, <function>)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<collection-instance>	Да	массив	Коллекция с элементами.
<iteratorName>	Да	Имя итератора.	Ключевой элемент
<function>	Да	expression	Функция условия, которая применяется для фильтрации элементов.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<new-collection/new-object>	Массив или объект.	Новая коллекция, в которой к каждому элементу применен фильтр, заданный функцией.

### Пример 1

В этом примере создается новая коллекция.

```
where(createArray(0, 1, 2, 3), x, x > 1)
```

И возвращается результат [2, 3].

### Пример 2

В этих примерах создается новая коллекция:

```
where(json("{'name': 'jack', 'age': '15'}"), x, x.value == 'jack')
where(json("{'name': 'jack', 'age': '15'}"), x=> x.value == 'jack')
```

И возвращают результат ["имя: гнездо", "Age: 15"]. Обратите внимание, что второе выражение является лямбда-выражением, которое может оказаться более удобочитаемым.

## Xml

**Только для C#.** Возвращает строку, которая содержит объект JSON, в формате XML.

```
xml('<value>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
----------	-------------	-----	----------

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<value>	Да	строка	Строка с объектом JSON для преобразования. Объект JSON должен содержать только одно корневое свойство, которое не может быть массивом. Используйте escape-символ \ для двойных кавычек ("").

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<версия XML>	объект	Закодированный XML для заданной строки или объекта JSON

### Пример 1

В этом примере создается версия XML для строки, которая содержит объект JSON.

```
xml(json('{"name": "Sophia Owen"}'))
```

И возвращается полученный текст XML.

```
<name>Sophia Owen</name>
```

### Пример 2

Предположим, что у вас есть представленный ниже объект JSON `person`.

```
{
  "person": {
    "name": "Sophia Owen",
    "city": "Seattle"
  }
}
```

В этом примере создается строка XML, содержащая этот объект JSON.

```
xml(json('{"person": {"name": "Sophia Owen", "city": "Seattle"}}'))
```

И возвращается полученный текст XML.

```
<person>
  <name>Sophia Owen</name>
  <city>Seattle</city>
<person>
```

### xPath

**Только для C#.** Проверяет XML на наличие узлов или значений, которые соответствуют выражению XPath, и возвращает соответствующие узлы или значения. Выражение XPath (или просто XPath) помогает перемещаться по структуре документа XML, чтобы вы могли выбирать узлы или вычислять значения в содержимом XML.

```
xPath('<xml>', '<xpath>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<xml>	Да	any	Строка XML для поиска узлов и значений, которые соответствуют значению выражения XPath
<xPath>	Да	any	Выражение XPath, используемое для поиска соответствующих узлов или значений XML

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<узел XML>	XML	Узел XML, где только один узел соответствует указанному выражению XPath
<value>	any	Значение из узла XML, где только одно значение соответствует указанному выражению XPath
<[ , ...] -или- [ , ...]>	массиве	Массив с узлами XML или значениями, которые соответствуют указанному выражению XPath

### Пример 1

В этом примере выполняется поиск узлов, которые соответствуют узлу `<name></name>` в указанных аргументах, и возвращается массив с этими значениями узлов:

```
xpath(items, '/produce/item/name')
```

Аргументы содержат строки `items` с таким кодом XML:

```
"<?xml version="1.0"?> <produce> <item> <name>Gala</name> <type>apple</type> <count>20</count> </item>
<item> <name>Honeycrisp</name> <type>apple</type> <count>10</count> </item> </produce>"
```

Ниже приведен результирующий массив с узлами, которые соответствуют условию `<name></name>`.

```
[ <name>Gala</name>, <name>Honeycrisp</name> ]
```

### Пример 2

В следующем примере 1 выполняется поиск узлов, которые соответствуют условию `<count></count>`, затем значения этих узлов суммируются с помощью функции `sum()`.

```
xpath(xml(parameters('items')), 'sum(/produce/item/count)')
```

И возвращается результат 30.

### year

Возвращает год из указанной метки времени.

```
year('<timestamp>')
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
< <i>timestamp</i> >	Да	строка	Строка, содержащая метку времени

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
< <i>year</i> >	Целое число	Значение года из указанной метки времени.

### Пример

В этом примере оценивается значение года в метке времени.

```
year('2018-03-15T00:00:00.000Z')
```

И возвращается результат 2018.

# Справочные материалы по API для адаптивных выражений

27.10.2020 • 2 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

## Ссылки на API

Адаптивные выражения не зависят от языка. Справочники по API доступны для разработчиков-роботов, работающих с адаптивными выражениями на следующих языках:

- [C#](#)
- [JavaScript](#)

# Справочник по API для создания языка

27.10.2020 • 2 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

## Ссылки на API

Шаблоны создания языка не зависят от языка. Справочники по API доступны для разработчиков-роботов, работающих с созданием языка, на следующих языках:

- [C#](#)
- [JavaScript](#)

# Формат файлов .lg

27.03.2021 • 14 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Файл. LG описывает шаблоны создания языка со ссылками на сущности и их композицией. В этой статье рассматриваются различные концепции, представленные в формате файла LG.

## Специальные символы

### Комментарии

> позволяет создать комментарий. Средство синтаксического анализа будет пропускать все строки с этим префиксом.

```
> This is a comment.
```

### Escape-символ

Используйте \ в качестве escape-символа.

```
# TemplateName
- You can say cheese and tomato \[toppings are optional\]
```

## Массивы и объекты

### Создание массива

Чтобы создать массив, используйте синтаксис \$ {[Object1, Object2,...]} . Например, это выражение:

```
${{['a', 'b', 'c']}}
```

Возвращает массив `['a', 'b', 'c']` .

### Создание объекта

Чтобы создать объект, используйте \$ {{Key1: value1, Key2: значение2,...}} . Например, это выражение:

```
${{user: {name: "Wilson", age: 27}}}
```

Возвращает следующий объект JSON:

```
{
  "user": {
    "name": "Wilson",
    "age": 27
  }
}
```

## Шаблоны

**Шаблоны** можно считать основным понятием в системе создания текста. Каждый шаблон имеет имя и что-то одно из следующего:

- список текстовых значений для однократных вариантов;
- определение структурированного содержимого;
- коллекция условий, в каждом из которых есть:
  - [адаптивное выражение](#);
  - список текстовых значений для однократных вариаций по условиям.

## Имена шаблонов

Имена шаблонов чувствительны к регистру и могут содержать только буквы, символы подчеркивания и цифры. Ниже приведен пример шаблона с именем `TemplateName`.

```
# TemplateName
```

Шаблоны не могут начинаться с цифры и любой части имени шаблона, разделенной на . не может начинаться с цифры.

## Варианты ответа шаблона

Вариации выражаются в виде списков разметки Markdown. Вы можете добавить для каждой вариации префикс с помощью символов - , ' или + .

```
# Template1
- text variation 1
- text variation 2
- one
- two

# Template2
* text variation 1
* text variation 2

# Template3
+ one
+ two
```

## Шаблон простого ответа

Шаблоны простого ответа могут содержать один или несколько вариантов текста, используемых для составления и расширения ответов. Один из доступных вариантов выбирается случайным образом библиотекой создания текста.

Вот пример такого шаблона с двумя вариантами ответа.

```
> Greeting template with two variations.
# GreetingPrefix
- Hi
- Hello
```

## Шаблон условного ответа

Шаблоны условного ответа позволяют создавать содержимое на основе условия. Эти условия выражаются с помощью [адаптивных выражений](#).

## IMPORTANT

Такие шаблоны не могут быть вложенными в одном шаблоне условного ответа. Для реализации вложенности используйте строение [структурированного шаблона ответа](#).

### Шаблон IF-ELSE

Шаблон IF-ELSE позволяет создать шаблон, который выбирает коллекцию на основе каскадной структуры условий. Оценка производится сверху вниз и прекращается, когда обнаруживается условие со значением `true` или блок ELSE.

Условные выражения заключаются в скобки `${}`. Ниже приведен пример с определением простого шаблона условного ответа IF-ELSE.

```
> time of day greeting reply template with conditions.  
# timeOfDayGreeting  
- IF: ${timeOfDay == 'morning'}  
  - good morning  
- ELSE:  
  - good evening
```

Вот еще один пример с определением шаблона условного ответа IF-ELSE. Обратите внимание, что ссылки на другие шаблоны простого или условного ответа можно включать в вариации любых других условий.

```
# timeOfDayGreeting  
- IF: ${timeOfDay == 'morning'}  
  - ${morningTemplate()}  
- ELSEIF: ${timeOfDay == 'afternoon'}  
  - ${afternoonTemplate()}  
- ELSE:  
  - I love the evenings! Just saying. ${eveningTemplate()}
```

### Шаблон SWITCH

Шаблон SWITCH позволяет создать шаблон, который сравнивает значение некоторого выражения с предложениями CASE и формирует результат по правилам для этого варианта. Условные выражения заключаются в скобки `${}`.

Ниже показано, как указать блок переключателя SWITCH CASE DEFAULT в системе создания текста.

```
# TestTemplate  
- SWITCH: ${condition}  
- CASE: ${case-expression-1}  
  - output1  
- CASE: ${case-expression-2}  
  - output2  
- DEFAULT:  
  - final output
```

А вот более сложный пример SWITCH CASE DEFAULT.

```
> Note: Any of the cases can include reference to one or more templates.  
# greetInAWeek  
- SWITCH: ${dayOfWeek(utcNow())}  
- CASE: ${0}  
  - Happy Sunday!  
-CASE: ${6}  
  - Happy Saturday!  
-DEFAULT:  
  - ${apology-phrase()}, ${defaultResponseTemplate()}
```

#### NOTE

Как и в условных шаблонах, шаблоны переключателей не могут быть вложенными.

### Структурированный шаблон ответа

Структурированный шаблон ответа позволяет определить сложную структуру с поддержкой большого набора функций из системы создания текста, таких как использование шаблонов, компоновка и подстановка, передавая задачу составления структурированного ответа объекту, вызывающему библиотеку создания текста.

Для ботов реализована встроенная поддержка следующих возможностей:

- определение действий;
- определение карточек.

Дополнительные сведения см. в статье [Шаблон структурированного ответа](#).

## Строение и расширение шаблона

### Ссылки на шаблоны

Текст вариации может содержать ссылки на другой именованный шаблон, чтобы упростить построение и разрешение в сложных ответах. Ссылки на другие именованные шаблоны обозначаются скобками, например так: \${<TemplateName>()} .

```
> Example of a template that includes composition reference to another template.  
# GreetingReply  
- ${GreetingPrefix()}, ${timeOfDayGreeting()}\n\n# GreetingPrefix  
- Hi  
- Hello\n\n# timeOfDayGreeting  
- IF: ${timeOfDay == 'morning'}  
  - good morning  
- ELSEIF: ${timeOfDay == 'afternoon'}  
  - good afternoon  
- ELSE:  
  - good evening
```

Вызов шаблона `GreetingReply` может приводить к одному из следующих разрешений расширения:

```
Hi, good morning  
Hi, good afternoon  
Hi, good evening  
Hello, good morning  
Hello, good afternoon  
Hello, good evening
```

## Сущности

При использовании непосредственно в тексте однократной вариации ссылки на сущности заключаются в фигурные скобки, например так: \${entityName}, или указываются без фигурных скобок в качестве параметра.

Сущности можно использовать в качестве параметра в следующих случаях:

- во [встроенной функции](#);
- в [условии шаблона условного ответа](#);
- для [вызыва разрешения шаблона](#).

## Использование встроенных функций в вариациях

[Встроенные функции](#), которые поддерживаются [адаптивными выражениями](#), можно использовать прямо в тексте однократной вариации, чтобы дополнительно расширить возможности составления текста. Чтобы использовать выражение как встроенное, просто заключите его в скобки.

```
# RecentTasks
- IF: ${count(recentTasks) == 1}
    - Your most recent task is ${recentTasks[0]}. You can let me know if you want to add or complete a task.
- ELSEIF: ${count(recentTasks) == 2}
    - Your most recent tasks are ${join(recentTasks, ', ', ' and ')}. You can let me know if you want to add or complete a task.
- ELSEIF: ${count(recentTasks) > 2}
    - Your most recent ${count(recentTasks)} tasks are ${join(recentTasks, ', ', ' and ')}. You can let me know if you want to add or complete a task.
- ELSE:
    - You don't have any tasks.
```

В приведенном выше примере встроенная функция [join](#) используется для перечисления всех значений коллекции `recentTasks`.

Поскольку шаблоны и встроенные функции имеют одинаковую сигнатуру вызова, шаблоны не могут иметь имена, совпадающие с именами встроенных функций.

Имя шаблона не должно совпадать с именем встроенной функции. Встроенные функции имеют приоритет при разрешении конфликтов. Чтобы избежать этих проблем, добавляйте префикс `lg.` к ссылке на имя шаблона. Пример:

```
> Custom length function with one parameter.  
# length(a)  
- This is user's customized length function  
  
# myfunc1  
> will call prebuilt function length, and return 2  
- ${length('hi')}  
  
# mufunc2  
> this calls the lg template and output 'This is user's customized length function'  
- ${lg.length('hi')}
```

## Многострочный текст в вариациях

Каждая однократная вариация может содержать многострочный текст, заключенный в тройные кавычки.

```
# MultiLineExample  
- ````This is a multiline list  
  - one  
  - two  
  ...  
- ````This is a multiline variation  
  - three  
  - four  
...````
```

В многострочных вариациях можно запрашивать расширение шаблона и подстановку сущностей, заключая нужную операцию в скобки: \${}.

```
# MultiLineExample  
- ````  
  Here is what I have for the order  
  - Title: ${reservation.title}  
  - Location: ${reservation.location}  
  - Date/ time: ${reservation.dateTimeReadBack}  
...````
```

Благодаря поддержке многострочного текста вы можете передавать в подсистему создания текста для полноценной обработки достаточно сложные структуры JSON и XML (например, текст в оболочке SSML для управления речевым ответом бота).

## Параметризация шаблонов

Чтобы шаблоны было проще использовать повторно в разных контекстах, их можно параметризовать. Это означает, что разные вызывающие объекты будут передавать в шаблон разные значения, которые будут применяться для разрешения расширений.

```
# timeOfDayGreetingTemplate (param1)
- IF: ${param1 == 'morning'}
  - good morning
- ELSEIF: ${param1 == 'afternoon'}
  - good afternoon
- ELSE:
  - good evening

# morningGreeting
- ${timeOfDayGreetingTemplate('morning')}

# timeOfDayGreeting
- ${timeOfDayGreetingTemplate(timeOfDay)}
```

## Импорт внешних ссылок

Шаблоны создания текста можно разделить на отдельные файлы и в любом файле указывать ссылки на другой файл. Для импорта шаблонов, определенных в другом файле, можно использовать ссылки в формате разметки Markdown.

```
[Link description](filePathOrUri)
```

Такое действие подключает все шаблоны, определенные в целевом файле. Следите за уникальностью имен шаблонов (ее можно обеспечить добавлением пространств имен с помощью

```
# \<namespace>. \<templatename> )
```

 во всех подключаемых файлах.

```
[Shared](../shared/common.1g)
```

## Функции, внедренные с помощью Создания текста

[Адаптивные выражения](#) предоставляют возможность внедрять пользовательские наборы функций. Дополнительные сведения можно получить в статье [Внедрение функций из библиотеки создания текста](#).

## Параметры

Разработчики могут задать параметры средства синтаксического анализа, чтобы дополнительно настроить способ оценки входных данных. Используйте нотацию `> !#`, чтобы задать параметры средства синтаксического анализа.

### IMPORTANT

Последний параметр в файле переопределяет все предыдущие параметры в том же документе.

### Параметр strict

Если разработчик не хочет получать результат NULL для вычисляемого выражения, следует применить параметр `strict`. Ниже показан пример простого применения `strict`.

```
> !# @strict = true
# template
- hi
```

Если включен параметр `strict`, для значений NULL будут создаваться ошибки с понятным сообщением.

```
# welcome
- hi ${name}
```

Если имя имеет значение NULL, диагностическое сообщение будет выглядеть так: 'name' evaluated to null. [welcome] Error occurred when evaluating '- hi \${name}' (Результат вычисления выражения "name" равен значению NULL. [приветствие] Произошла ошибка при вычислении "-hi \${Name}"). Если параметр strict не задан или имеет значение false, возвращается результат в допустимом формате. Тогда указанный выше пример кода создаст текст hi null.

### Параметр replaceNull

Разработчики могут создавать делегаты для замены значений NULL в вычисляемых выражениях с помощью параметра replaceNull :

```
> !# @replaceNull = ${path} is undefined
```

В приведенном выше примере входные данные NULL в переменной `path` будут заменены текстом `${path} is undefined` (значение пути не определено). Для следующих входных данных, где `user.name` имеет значение NULL:

```
hi ${user.name}
```

будет возвращен результат `hi user.name is undefined`.

### Параметр lineBreakStyle

Разработчики могут задать метод отрисовки разрывов строк в системе создания текста с помощью параметра `lineBreakStyle`. В настоящее время поддерживаются два метода:

- `default` — разрывы строк в многострочном тексте создают обычные разрывы строк;
- `markdown` — разрывы строк в многострочном тексте будут автоматически преобразованы в две строки, чтобы создать новую строку.

В приведенном ниже примере показано, как задать параметр `lineBreakStyle` со значением `markdown`.

```
> !# @lineBreakStyle = markdown
```

### Параметр Namespace

Можно зарегистрировать пространство имен для шаблонов LG, которые необходимо экспортовать. Если пространство имен не указано, будет задано имя файла без расширения.

В приведенном ниже примере показано, как задать параметру пространства имен значение `foo`.

```
> !# @Namespace = foo
```

### Параметр Exports

Можно указать список шаблонов LG для экспорта. Экспортированные шаблоны вызываются так же, как готовые функции.

В приведенном ниже примере показано, как задать параметру экспорта значение `template1, template2`.

```
> !# @Namespace = foo
> !# @Exports = template1, template2

# template1(a, b)
- ${a + b}

# template2(a, b)
- ${join(a, b)}
```

Для вызова экспортированных шаблонов следует использовать

```
foo.template1(1,2), foo.template2(['a', 'b', 'c'], ',').
```

## Дополнительные ресурсы

- [Справочник по API для C#](#)
- [Справочник по API JavaScript](#)
- Прочтайте статью [Отладка с помощью адаптивных средств](#), чтобы узнать, как анализировать и отлаживать файлы. LG.

# Структурированный шаблон ответа

27.03.2021 • 9 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Структурированный шаблон ответа позволяет разработчикам определять сложную структуру с поддержкой большого набора функций компонента [Создание текста](#), таких как работа с шаблонами и композиция. При этом за интерпретацию структурированного ответа отвечает объект, вызывающий библиотеку Создателя текста.

Для приложений-роботов предоставляется следующая поддержка:

- определение [действия](#)
- определение [карты](#)

Шаблон [действия "Bot Framework"](#) содержит несколько настраиваемых полей. Ниже перечислены наиболее часто используемые свойства, которые можно настроить с помощью определения шаблона действия.

Свойство	Вариант использования
текст	Отображаемый текст, на основе которого канал создает визуальное представление.
Speak	Произносимый текст, на основе которого канал создает звуковое представление.
Вложения	Список вложений с указанием типов. Используется каналами для отображения в виде карточек пользовательского интерфейса или других универсальных типов вложенных файлов.
SuggestedActions	Список действий, отображаемых в качестве предложений для пользователя.
InputHint	Управляет состоянием потока записи звука на устройствах, поддерживающих речевой ввод. Возможные значения: <code>accepting</code> , <code>expecting</code> или <code>ignoring</code> .

По умолчанию сопоставитель шаблонов не реализует резервное поведение. Если свойство не указано, оно остается неопределенным. Например, значение свойства `Speak` не передается автоматически в свойство `Text`, если задано только свойство `Text`.

## Определение

Пример определения структурированного шаблона:

```

# TemplateName
> this is a comment
[Structure-name
  Property1 = <plain text> .or. <plain text with template reference> .or. <expression>
  Property2 = list of values are denoted via '|'. e.g. a | b
> this is a comment about this specific property
  Property3 = Nested structures are achieved through composition
]

```

Пример простого шаблона текста:

```

# AskForAge.prompt
[Activity
  Text = ${GetAge()}
  Speak = ${GetAge()}
]

# GetAge
- how old are you?
- what is your age?

```

Пример текста с предложенным действием. Чтобы обозначить список, используйте | .

```

> With '|' you are making attachments a list.
# AskForAge.prompt
[Activity
  Text = ${GetAge()}
  SuggestedActions = 10 | 20 | 30
]

```

Пример определения [карты для имиджевого баннера](#):

```

# HeroCard
[Herocard
  title = Hero Card Example
  subtitle = Microsoft Bot Framework
  text = Build and connect intelligent bots to interact with your users naturally wherever they are, from
text/sms to Skype, Slack, Office 365 mail and other popular services.
  images = https://sec.ch9.ms/ch9/7fff5/e07cfef0-aa3b-40bb-9baa-
7c9ef8ff7fff5/buildreactionbotframework_960.jpg
  buttons = Option 1| Option 2| Option 3
]

```

#### NOTE

LG предоставляет некоторую вариативность в определении карты, которая преобразуется в соответствие с [определенением карты SDK](#). Например, оба `image` поля и `images` поддерживаются во всех определениях карт в LG, хотя `images` поддерживаются только в определении карты SDK.

Значения, определенные во всех `image` `images` полях и в карте херокард или эскиза, объединяются и преобразуются в список изображений в созданной карточке. Для других типов карт в поле будет присвоено Последнее значение, заданное в шаблоне `image`. Значения, назначаемые полю, `image/images` могут быть строкой, [адаптивным выражением](#) или массивом в формате с помощью | .

Объединение указанных выше шаблонов:

```

# AskForAge.prompt
[Activity
    Text = ${GetAge()}
    Speak = ${GetAge()}
    Attachments = ${HeroCard()}
    SuggestedActions = 10 | 20 | 30
    InputHint = expecting
]

# GetAge
- how old are you?
- what is your age?

# HeroCard
[Herocard
    title = Hero Card Example
    subtitle = Microsoft Bot Framework
    text = Build and connect intelligent bots to interact with your users naturally wherever they are, from
text/sms to Skype, Slack, Office 365 mail and other popular services.
    images = https://sec.ch9.ms/ch9/7ff5/e07cfef0-aa3b-40bb-9baa-
7c9ef8ff7ff5/buildreactionbotframework_960.jpg
    buttons = Option 1| Option 2| Option 3
]

```

По умолчанию любая ссылка на шаблон вычисляется один раз во время вычисления структурированного шаблона.

Например, `# AskForAge.prompt` возвращает один и тот же текст разрешения для обоих свойств `Speak` и `Text`.

```

# AskForAge.prompt
[Activity
    Text = ${GetAge()}
    Speak = ${GetAge()}
]

# GetAge
- how old are you?
- what is your age?

```

Вы можете применить `<TemplateName>!()`, чтобы запросить новую оценку каждой ссылки в структурированном шаблоне.

В приведенном ниже примере `Speak` и `Text` могут иметь разные текстовые разрешения, так как `GetAge` заново оценивается для каждого экземпляра.

```

[Activity
    Text = ${GetAge()}
    Speak = ${GetAge!()}
]

# GetAge
- how old are you?
- what is your age?

```

Вот пример отображения карусели карточек:

```

# AskForAge.prompt
[Activity
> Defaults to carousel layout in case of list of cards
    Attachments = ${foreach($cardValues, item, HeroCard(item))}
]

# AskForAge.prompt_2
[Activity
> Explicitly specify an attachment layout
    Attachments = ${foreach($cardValues, item, HeroCard(item))}
    AttachmentLayout = list
]

# HeroCard (title, subtitle, text)
[Herocard
    title = ${title}
    subtitle = ${subtitle}
    text = ${text}
    images = https://sec.ch9.ms/ch9/7ff5/e07cfef0-aa3b-40bb-9baa-
7c9ef8ff7ff5/buildreactionbotframework_960.jpg
    buttons = Option 1| Option 2| Option 3
]

```

Используйте \ в качестве escape-символа.

```

> You can use '\' as an escape character
> \${GetAge()} would not be evaluated as expression, would be parsed as '\${getAge()}' string
# AskForAge.prompt
[Activity
    Text = \${GetAge()}
    SuggestedActions = 10 \| cards | 20 \| cards
]

```

## Композиция структурированного шаблона

В структурированных шаблонах поддерживается следующее поведение композиции:

- Композиция учитывает контекст структуры. Если целевой шаблон, на который указывает ссылка, также является структурированным шаблоном, типы структуры должны совпадать. Например, можно создать ссылку на ActivityTemplate в другом шаблоне ActivityTemplate.
- Ссылки на шаблоны простого или условного ответа могут находиться в любом месте в пределах структурированного шаблона.

Предположим, что у вас есть следующий шаблон:

```

# T1
[Activity
    Text = ${T2()}
    Speak = foo bar ${T3().speak}
]

# T2
- This is awesome

# T3
[Activity
    Speak = I can also speak!
]

```

Вызов `evaluateTemplate('T1')` создаст следующую внутреннюю структуру.

```
[Activity
    Text = This is awesome
    Speak = I can also speak!
]
```

## Полная ссылка на другой структурированный шаблон

Вы можете указать ссылку на структурированный шаблон в виде свойства в другом структурированном шаблоне или в виде ссылки в шаблоне простого или условного ответа.

Ниже представлен пример полной ссылки на другой структурированный шаблон:

```
# ST1
[MyStruct
    Text = foo
    ${ST2()}
]
# ST2
[MyStruct
    Speak = bar
]
```

При использовании этого содержимого вызов `evaluateTemplate('ST1')` создаст следующую внутреннюю структуру:

```
[MyStruct
    Text = foo
    Speak = bar
]
```

Если одно и то же свойство существует и в вызывающем, и в вызываемом шаблоне, содержимое вызывающего объекта переопределит любое содержимое в вызываемом шаблоне.

Ниже приведен пример:

```
# ST1
[MyStruct
    Text = foo
    ${ST2()}
]
# ST2
[MyStruct
    Speak = bar
    Text = zoo
]
```

При использовании этого контекста вызов `evaluateTemplate('ST1')` создаст следующую внутреннюю структуру:

```
[MyStruct
    Text = foo
    Speak = bar
]
```

Обратите внимание, что такой стиль композиции может существовать только на корневом уровне. Если в свойстве есть ссылка на другой структурированный шаблон, при разрешении учитывается контекст

этого свойства.

## Ссылка на внешний файл в структурированном вложении

Есть две встроенные функции для создания ссылок на внешние файлы.

1. `fromFile(fileAbsoluteOrRelativePath)` загружает указанный файл. Возвращаемое этой функцией содержимое будет поддерживать вычисление содержимого. Вычисляются ссылки на шаблоны, свойства и выражения.
2. `ActivityAttachment(content, contentType)` устанавливает значение `contentType`, если оно не указано в содержимом.

Эти две встроенные функция позволяют получить любое внешнее содержимое, в том числе любые типы карточек. Используйте следующие структурированные шаблоны создания текста, чтобы определить действие:

```
# AdaptiveCard
[Activity
    Attachments = ${ActivityAttachment(json(fromFile('.../card.json')), 'adaptiveCard')}
]

# HeroCard
[Activity
    Attachments = ${ActivityAttachment(json(fromFile('.../card.json')), 'heroCard')}
]
```

Вы также можете использовать вложения, как показано ниже:

```
# AdaptiveCard
[Attachment
    contenttype = adaptivecard
    content = ${json(fromFile('.../card.json'))}
]

# HeroCard
[Attachment
    contenttype = herocard
    content = ${json(fromFile('.../card.json'))}
]
```

## Дополнительные сведения

- [Справочник по API для C#](#)
- [Справочник по API JavaScript](#)
- Прочтите статью [Отладка с помощью аддитивных средств](#), чтобы узнать, как анализировать и отлаживать шаблоны.

# Функции, внедренные из библиотеки Создания текста

27.03.2021 • 4 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье приводятся сведения о внедрении функций из библиотеки [Создания текста \(LG\)](#).

## ActivityAttachment

Возвращает `activityAttachment`, созданный на основе объекта и типа.

```
ActivityAttachment(<collection-of-objects>)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<code>&lt;content&gt;</code>	Да	объект	Объект, содержащий информацию о вложении.
<code>&lt;type&gt;</code>	Да	строка	Строка, представляющая тип вложения.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<code>&lt;activityAttachment&gt;</code>	объект	<code>activityAttachment</code> , сформированный из входных данных.

*Пример.*

В этом примере коллекция объектов преобразуется в `activityAttachment`.

Предположим, что у вас есть следующий шаблон:

```
# externalHeroCardActivity(type, title, value)
[Activity
    attachments = ${ActivityAttachment(jsonFromFile('.\\herocard.json')), 'herocard'}
]
```

И вот такой `herocard.json`:

```
{
  "title": "titleContent",
  "text": "textContent",
  "Buttons": [
    {
      "type": "imBack",
      "Title": "titleContent",
      "Value": "textContent",
      "Text": "textContent"
    }
  ],
  "tap": {
    "type": "${type}",
    "title": "${title}",
    "text": "${title}",
    "value": "${value}"
  }
}
```

Теперь вы вызываете `externalHeroCardActivity()` в качестве функции:

```
externalHeroCardActivity('signin', 'Signin Button', 'http://login.microsoft.com')
```

И она возвращает `herocard`:

```
{
  "lgType" = "attachment",
  "contenttype" = "herocard",
  "content" = {
    "title": "titleContent",
    "text": "textContent",
    "Buttons": [
      {
        "type": "imBack",
        "Title": "titleContent",
        "Value": "textContent",
        "Text": "textContent"
      }
    ],
    "tap": {
      "type": "signin",
      "title": "Signin Button",
      "text": "Signin Button",
      "value": "http://login.microsoft.com"
    }
  }
}
```

## ЕКСПАНДТЕКСТ

Оцените обычный текст в объекте и возвратите развернутые текстовые данные.

```
expandText(<object>)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
----------	-------------	-----	----------

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<object>	Да	объект	Объект с текстом, который необходимо развернуть.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<evaluated-result>	объект	Развернутые текстовые данные.

### Пример

В этом примере вычисляется обычный текст в объекте JSON и возвращается развернутый текстовый результат.

Предположим, у вас есть следующий объект:

```
{
  "@answer": "hello ${user.name}",
  "user": {
    "name": "vivian"
  }
}
```

Вызов `expandText(@answer)` приведет к появлению объекта **Hello Вивиан**.

## шаблон

Возвращает вычисленный результат для заданного имени шаблона и области.

```
template(<templateName>, '<param1>', '<param2>', ...)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<templateName>	Да	строка	Строка, представляющая имя шаблона.
<param1>, <param2>, ...	Да	Объект	Параметры, которые передаются в сборку.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<evaluated-result>	объект	Результат, полученный путем оценки шаблона как функции.

### Пример

В этом примере вычисляется результат вызова шаблона в качестве функции.

Предположим, что у вас есть следующий шаблон:

```
# welcome(userName)
- Hi ${userName}
- Hello ${userName}
- Hey ${userName}
```

Вызов `template("welcome", "DL")` приведет к получению одного из следующих значений:

- *Hi DL*
- *Hello DL*
- *Hey DL*

## fromFile

Возвращает результат оценки выражения в указанном файле.

```
fromFile(<filePath>)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<code>&lt;filePath&gt;</code>	Да	строка	Относительный или абсолютный путь к файлу, который содержит выражения.

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<code>&lt;результат&gt;</code>	строка	Строковое представление результата оценки выражения.

### Пример

В этом примере результат вычисляется на основе указанного файла.

Предположим, у вас есть файл с именем `/home/user/test.txt`. В этом файле размещено следующее содержимое:

```
`you have ${add(1,2)} alarms`  
fromFile('/home/user/test.txt')
```

Функция `fromFile()` вычислит выражение и заменит исходное выражение результатом этого вычисления.

Вызов `fromFile('/home/user/test.txt')` возвращает строку *you have 3 alarms*.

## isTemplate

Возвращает результат проверки того, включено ли указанное имя шаблона в средство оценки.

```
isTemplate(<templateName>)
```

ПАРАМЕТР	ОБЯЗАТЕЛЬНО	ТИП	ОПИСАНИЕ
<templateName>	Да	Строка	Имя шаблона для проверки.

ВОЗВАЩАЕМОЕ ЗНАЧЕНИЕ	ТИП	ОПИСАНИЕ
<результат>	Логическое	Указывает, включено ли указанное имя шаблона в средство оценки.

### Пример

В этом примере функция `isTemplate()` применяется для проверки того, включено ли указанное имя шаблона в средство оценки. Предположим, у вас есть три таких шаблона:

```
# welcome
- hi

# show-alarms
- 7:am and 8:pm

# add-to-do
- you add a task at 7:pm
```

`isTemplate("welcome")` Результатом вызова будет `true`. `isTemplate("delete-to-do")` Результатом вызова будет `false`.

## Дополнительные сведения

- [Формат файлов LG](#)
- [Шаблон структурированного ответа](#)

# Формат LU-файла

27.03.2021 • 23 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

LU-файл содержит напоминающие Markdown простые текстовые определения понятий LUIS. В этой статье описаны несколько понятий, которые выражаются в формате LU-файла.

## Добавление комментариев

> позволяет создать комментарий. Ниже приведен пример:

```
> This is a comment and will be ignored.  
  
# Greeting  
- hi  
- hello
```

## Блокировка с намерением

**Намерение** представляет действие, которое хочет выполнить пользователь. Намерение — это назначение или цель, выраженная во введенных пользователем данных, например бронирование авиабилета, оплата счета или поиск новостной статьи. Например, приложение для путешествий может определять намерение *BookFlight*. Вы определяете и именуете намерения, соответствующие этим действиям.

Ниже приведен LU-файл, принимающий простое намерение `Greeting` со списком примеров речевых фрагментов, которые позволяют пользователям выразить это намерение. Используйте - символы, + или \* \*, чтобы обозначить список. Нумерованные списки не поддерживаются.

```
# Greeting  
- Hi  
- Hello  
- Good morning  
- Good evening
```

`#<intent-name>` описывает новый раздел определения намерений. Каждая строка после определения намерения является примером речевого фрагмента, описывающего это намерение. Можно добавить сразу несколько определений намерений в один файл, как показано в примере ниже.

```
# Greeting  
- Hi  
- Hello  
- Good morning  
- Good evening  
  
# Help  
- help  
- I need help  
- please help
```

Каждый раздел обозначен нотацией `#<intent name>`. Обратите внимание, что при анализе Lu файла

пропускаются пустые строки.

## Сущность

Сущность представляет подробные сведения, относящиеся к речевому фрагменту. Например, в речевом фрагменте *Book a ticket to Paris* (Забронировать билет в Париж) слово *Paris* (Париж) является расположением.

ПРИМЕР РЕЧЕВОГО ФРАГМЕНТА ПОЛЬЗОВАТЕЛЯ	СУЩНОСТИ
«Заносите перелет в _ * Сиэтле * *?»	Seattle
"Когда <b>открыт</b> ваш магазин?"	открыт
"Запланировать встречу в 13:00 с <b>Бобом</b> в Distribution"	13:00, Боб

### Определение

Сущности объявляются следующим образом.

```
@ <entity-type> <entity-name> [[hasRole[s]] <comma-separated-list-of-roles>] [hasFeature[s] <comma-separated-list-of-features>]
```

Сущности, для которых требуется определение (например, сущности списка и регулярных выражений), представляются с помощью приведенной ниже нотации.

```
@ <entity-name> = <definition>
```

Объявление и определение также можно объединить в одну строку.

```
@ <entity-type> <entity-name> [[hasRoles] <comma-separated-list-of-roles>] = <definition>
```

entity type Параметры, entity name И definition являются обязательными, и roles являются необязательными. Дополнительные сведения см. в разделе [роли](#).

Имена сущностей, содержащие пробелы, можно заключать в кавычки. Обратите внимание на то, что имена встроенных сущностей не могут содержать пробелы.

Ниже приведен пример:

```
@ ml "this is a simple entity" role1, role2
@ ml 'this is a simple entity' hasRoles role1, role2
```

### Сущность машинного обучения

Сущности машинного обучения учатся на контексте в речевом фрагменте. Группирование сущностей по родительскому элементу выполняется независимо от типа сущности. Это делает значимым вариативность расположения в примерах речевых фрагментов.

В приведенном ниже примере сущность машинного обучения name определена как firstName И lastName .

```
@ ml name firstName, lastName
```

Любой сущности с меткой, тип которой не присвоен явно, средство синтаксического анализа по умолчанию присваивает тип сущности машинного обучения.

```
# getUserName
- my name is {username=vishwac}

> Without an explicit entity definition, 'userName' defaults to 'ml' entity type.
```

Поддерживаются следующие типы [встроенных сущностей](#) LUIS.

- age
- datetimeV2
- Измерение
- email
- geographyV2
- KeyPhrase
- money
- number
- ordinal
- ordinalV2
- процент
- personName
- phonenumbers
- Температура
- url
- DATETIME

Ниже приведены примеры предварительно созданных сущностей.

```
@ prebuilt number numOfGuests, age
@ prebuilt datetimeV2 fromDate, toDate
@ prebuilt age userAge
```

Не все типы встроенных сущностей доступны для всех языковых стандартов. Ознакомьтесь с [доступными сущностями для различных языков и региональных параметров в модели LUIS](#), чтобы узнать о поддержке встроенных сущностей для языковых стандартов.

### Сущность списка

[Список сущностей](#) представляет собой фиксированный, закрытый набор связанных слов вместе с их синонимами. Они извлекаются на основе точного совпадения текста.

В приведенном ниже примере определена сущность списка, которая содержит синонимы для цветов.

```

@ list color favColor, screenColor
@ color =
  - <normalized-value> :
    - <synonym1>
    - <synonym2>
    - ...
  - <normalized-value> :
    - <synonym1>, <synonym2>, ...

> Alternate definition

@ list color favColor, screenColor =
  - <normalized-value> :
    - <synonym1>; <synonym2>; ...

```

При использовании сущностей списка необходимо добавлять значение из списка непосредственно в речевой фрагмент, а не в метку сущности или какое-либо другое значение.

### Составная сущность

[Составная сущность](#) образуется из других сущностей, таких как встроенные или простые сущности, сущности регулярного выражения и сущности списка. Эти отдельные сущности формируют единую сущность.

Ниже приведен пример простой составной сущности.

```

@ composite deviceTemperature from, to
@ deviceTemperature =
  - child1, child2

> Alternate definition

@ composite deviceTemperature from, to = [child1, child2]

```

Вот пример более сложного определения.

```

# setThermostat
> This utterance labels "thermostat to 72" as a composite entity `deviceTemperature`.
  - Please set {deviceTemperature = thermostat to 72}
> This is an example utterance that labels "owen" as a customDevice (ml entity) and wraps "owen to 72" with
the `deviceTemperature` composite entity.
  - Set {deviceTemperature = {customDevice = owen} to 72}

> Defines a composite entity `deviceTemperature` that has device (list entity), customDevice (ml entity),
and temperature (prebuilt entity) as children.

@ composite deviceTemperature = [device, customDevice, temperature]

@ list device =
  - thermostat :
    - Thermostat
    - Heater
    - AC
    - Air conditioner
  - refrigerator :
    - Fridge
    - Cooler

@ ml customDevice

@ prebuilt temperature

```

## Сущность регулярного выражения

Сущность регулярного выражения извлекает сущность на основе предоставленного шаблона регулярного выражения.

Ниже приведен пример простого определения сущности регулярного выражения.

```
> from, to are roles to href-number.  
@ regex href-number from, to  
@ href-number = /href-[0-9]{6}/  
  
> Alternate definition  
  
> from, to are roles to href-number.  
@ regex href-number from, to = /href-[0-9]{6}/
```

## Роли

Роль — это именованный псевдоним для сущности на основе контекста в речевом фрагменте. Роли можно использовать с любым встроенным или настраиваемым типом сущности (кроме списков фраз). Они используются в примерах речевых фрагментов и шаблонов.

В примере ниже сущность **Location** имеет две роли: `origin` И `destination`.

Сущность	Роль	Назначение
Расположение	origin	Где отчасти плоскости
Расположение	ресурс destination	Где на плоскости

Роли в формате LU-файла могут быть определены явно или неявно. Явное определение роли следует после нотации.

```
@ <entityType> <entityName> [hasRole[s]] role1, role2, ...
```

Ниже приведено несколько способов явного определения сущностей и их ролей.

```
> # ml entity definition with roles  
> the following are synonymous:  
  
@ ml name role1, role2  
  
@ ml name hasRoles role1, role2  
  
@ ml name  
@ name hasRoles role1, role2  
  
@ ml name  
@ name hasRole role1  
@ name hasRole role2
```

Вы можете ссылаться на неявно определенные роли непосредственно в шаблонах и речевых фрагментах с метками в следующем формате.

```
{@<entityName>:<roleName>}
```

В приведенном ниже примере можно увидеть, как роли `userName:firstName` И `userName:lastName` неявно определены.

```
# AskForUserName
- {userName:firstName=vishwac} {userName:lastName=kannan}
- I'm {userName:firstName=vishwac}
- my first name is {userName:firstName=vishwac}
- {userName=vishwac} is my name

> This definition is same as including an explicit definition for userName with 'lastName', 'firstName' as roles

> @ ml(userName hasRoles lastName, firstName)
```

В [шаблонах](#) можно использовать роли с `{<entityName>:<roleName>}` нотацией. Ниже приведен пример:

```
# getUserName
- call me {name:userName}
- I'm {name:userName}
- my name is {name:userName}
```

Можно также определить несколько ролей для сущности в шаблонах, показанных ниже:

```
> Roles can be specified for list entity types as well - in this case fromCity and toCity are added as roles to the 'city' list entity defined further below

# BookFlight
- book flight from {city:fromCity} to {city:toCity}
- [can you] get me a flight from {city:fromCity} to {city:toCity}
- get me a flight to {city:toCity}
- I need to fly from {city:fromCity}

$city:Seattle=
- Seattle
- Tacoma
- SeaTac
- SEA

$city:Portland=
- Portland
- PDX
```

## Шаблоны

[Шаблоны](#) позволяют определить набор правил, расширяющих модель машинного обучения. Вы можете определить шаблоны в LU-файле, определив сущность в речевом фрагменте без значения с меткой.

Например, следующее определение будет рассматриваться как шаблон с `alarmTime`, установленным в качестве шаблона.

```
# DeleteAlarm
- delete the {alarmTime} alarm
```

Этот пример будет рассматриваться как речевой фрагмент, так как он содержит значение с меткой `7AM`.

```
# DeleteAlarm
- delete the {alarmTime=7AM} alarm
```

#### NOTE

Любой речевой фрагмент без по крайней мере одного значения с меткой будет рассматриваться как шаблон.

Любая сущность без явного значения с меткой будет по умолчанию считаться шаблоном.

## Определение списка фраз

**Список фраз** — это список слов, фраз, чисел или других символов, которые помогут найти концепцию, которую вы пытаетесь определить. В этом списке учитывается регистр букв.

Для описания сущностей списка фраз используется приведенная ниже нотация.

```
@ phraselist <Name>
  - <synonym1>
  - <synonym2>
```

Вот пример определения списка фраз.

```
@ phraseList Want
@ phraseList Want =
  - require, need, desire, know

> You can also break up the phrase list values into an actual list

@ phraseList Want =
  - require
  - need
  - desire
  - know
```

По умолчанию синонимы задаются как невзаимозаменяемые. При необходимости можно задать синонимы в определении как взаимозаменяемые. Ниже приведен пример:

```
@ phraselist question(interchangeable) =
  - are you
  - you are
```

Списки фраз можно пометить как `disabled`, используя следующую нотацию.

```
@ phraselist abc disabled

> also same as this
@ phraselist question(interchangeable) =
  - are you
  - you are

@ question disabled
```

По умолчанию списки фраз доступны для всех моделей. Однако если явным образом начать назначать списки фраз в качестве признака (дескриптора) для других моделей, некоторые списки фраз не будут доступны для всех моделей. Чтобы явно сделать список фраз всегда доступным для всех моделей, используйте следующее.

```
@ phraselist abc enabledForAllModels
```

## Привязка признаков к определенной модели

Списки фраз можно добавить в качестве признака в:

- другое намерение;
- другую сущность;
- дочерний элемент в n-уровневой сущности.

Ниже приведен пример того, как можно определить список фраз в качестве признака для другой модели.

```
> phrase list definition

@ phraseList PLCity(interchangeable) =
    - seattle
    - space needle
    - SEATAC
    - SEA

> phrase list as feature to intent (also applicable to entities)

@ intent getUserProfileIntent usesFeature PLCity

> phrase list as a feature to an ml entity

@ ml myCity usesFeature PLCity

@ regex regexZipcode = /[0-9]{5}/

> phrase list as feature to n-depth entity with phrase list as a feature

@ ml address fromAddress, toAddress
@ address =
    - @ number 'door number'
    - @ ml streetName
    - @ ml location
        - @ ml city usesFeature PLCity
        - @ regexZipcode zipcode
```

## Добавление сущности или намерения в качестве признака

Ниже приведены примеры добавления намерений и сущностей в качестве признака с использованием `usesFeature`.

```

> entity definition - @ <entityType> <entityName> [<roles>]

@ prebuilt personName
@ prebuilt age

> entity definition with roles

@ ml userName hasRoles fistName, lastName

> add an entity as a feature to another entity

@ userName usesFeature personName

> add an entity as feature to an intent

@ intent getUserNameIntent usesFeature personName

> Intent definition

# getUserNameIntent
- utterances

> multiple entities as a feature to a model

@ intent getUserNameIntent usesFeature age, personName

> intent as a feature to another intent

@ intent getUserProfileIntent usesFeature getUserNameIntent

# getUserProfileIntent
- utterances

```

## Сущность машинного обучения с дочерними элементами

Ниже приведено определение сущности машинного обучения `address` с двумя ролями, `fromAddress` и `toAddress`, а также дочерним элементом.

```

@ list listCity
@ prebuilt number
@ prebuilt geographyV2
@ regex regexZipcode = /[0-9]{5}/
@ ml address hasRoles fromAddress, toAddress
@ address =
  - @ number 'door number'
  - @ ml streetName
  - @ ml location usesFeature geographyV2
    - @ listCity city
    - @ regexZipcode zipcode

```

## Высказывания

[Высказываниями](#) называются входные данные, поступившие от пользователя, которые нужно расшифровать в приложении. Активное обучение, то есть постоянный процесс обучения по новым высказываниям, является важным элементом аналитики на основе машинного обучения, которую предоставляет LUIS. Чтобы научить LUIS извлекать намерения и сущности, нужно собрать разные варианты примеров фраз для каждого намерения.

Соберите фразы, которые вы ожидаете от ваших пользователей. Включите речевые фрагменты с тем же смыслом и множеством конструкций, включая (но не ограничиваясь) следующие.

- Длина фразы — короткие, средние и длинные для клиентского приложения
- Длина слова и фразы
- Размещение слов — сущности в середине, начале и конце фразы.
- Грамматика
- Преобразование во множественную форму
- Морфология
- Выбор существительных и глаголов
- Пунктуация

Вы можете добавить метки для сущностей в речевых фрагментах, используя следующую нотацию.

```
# getUserProfile
- my name is {@userName = vishwac}

@ ml userName
```

Роли также можно добавить метки для ролей непосредственно в речевых фрагментах.

```
# getUserProfile
- my name is {@firstName = vishwac}

@ ml userName hasRoles firstName
```

Для сущностей машинного обучения с дочерними элементами можно также добавить метки.

```
# getUserProfile
- my name is {@userProfile = {@firstName = vishwac}}

@ prebuilt personName

@ ml userProfile
- @ personName firstName
- @ personName lastName
```

Чтобы упростить добавление меток дочерних сущностей как для сущностей машинного обучения, так и для сущностей составного типа, эти метки можно разбить.

```
# getUserProfile
- my name is vishwac and I'm 36 years old
  - my name is {@userProfile = vishwac and I'm 36 years old}
  - my name is {@firstName = vishwac} and I'm 36 years old
  - my name is vishwac and I'm {@userAge = 36} years old
  - i'm {@userProfile = {@firstName = vishwac}}

@ ml userProfile
- @personName firstName
- @personName lastName

@ prebuilt personName
```

## Описание модели

Вы можете добавить в LU-файл сведения о конфигурации для приложения LUIS или базы знаний QnA Maker. Это позволит указать анализатору, как правильно обработать содержимое LU-файла.

Вот как можно определить сведения о конфигурации с помощью >! #:

```
> !# @<property> = <value>
> !# @<scope>-<property> = <value>
> !# @<scope>-<property> = <semicolon-delimited-key-value-pairs>
```

Обратите внимание на то, что любые сведения, явно переданные через аргументы интерфейса командной строки, переопределяют сведения в LU-файле.

```
> Parser instruction - this is optional; unless specified, parser will default to the latest version.
> !# @version = 1.0

> LUIS application description
> !# @app.name = my luis application
> !# @app.desc = description of my luis application
> !# @app.versionId = 0.5
> !# @app.culture = en-us
> !# @app.luis_schema_version = 3.2.0
```

## Внешние ссылки

В следующих разделах подробно описано, как сделать [локальный файл](#) и ссылки [URI](#).

### Ссылки на локальные файлы

Ссылки на LU-файл соответствуют требованиями синтаксиса ссылок Markdown. Поддерживаются следующие ссылки.

- Ссылка на другой LU-файл посредством `[link name](<.lu file name>)`. Ссылка может быть абсолютным путем или относительным путем от содержащего сведения LU-файла.
- Ссылка на папку с другими LU-файлами поддерживается следующим образом.
  - `[link name](<.lu file path*>)` : ищет файлы. lu по указанному абсолютному или относительному пути
  - `[link name](<.lu file path***)` : рекурсивно ищет файлы. lu по указанному абсолютному или относительному пути, включая вложенные папки.
- Можно также добавить ссылки на речевые фрагменты, определенные в заданном файле в разделе intent или в виде пар вопросов и ответов.
  - `[link name](<.lu file path>#<INTENT-NAME>)` : находит все фразы продолжительностью в разделе <намерения-NAME> в файле. lu и добавляет их в список фразы продолжительностью, где указана ссылка.
  - `[link name](<.lu file path>#<INTENT-NAME>*utterances*)` : выполняет поиск всех фразы продолжительностью (не шаблонов) в разделе <имя намерения> в Lu-файле и добавляет их в список фразы продолжительностью, где указана ссылка.
  - `[link name](<.lu file path>#<INTENT-NAME>*patterns*)` : находит все шаблоны (не фразы продолжительностью) в разделе <имя намерения> в файле Lu и добавляет их в список шаблонов, где указана ссылка.
  - `[link name](<.lu file path>#*utterances*)` : находит все фразы продолжительностью в файле. lu и добавляет их в список фразы продолжительностью, где указана ссылка.
  - `[link name](<.lu file path>#*patterns*)` : находит все шаблоны в Lu-файле и добавляет их в список фразы продолжительностью, где указана ссылка.
  - `[link name](<.lu file path>##utterancesAndPatterns*)` — находит все фразы продолжительностью и закономерности в файле. lu и добавляет их в список фразы продолжительностью, где указана ссылка.
  - `[link name](<.qna file path>#$name?)` : находит все изменения из конкретного определения изменения в QnA содержимом и добавляет их в список фразы продолжительностью, где указана

ссылка.

- `[link name](<.qna file path>#*alterations*)` : находит все изменения из QnA содержимого и добавляет их в список фразы продолжительностью, где указана ссылка.
- `[link name](<.qna file path>#?question-to-find?)` : находит все вопросы о вариантах из конкретного вопроса и добавляет их в список фразы продолжительностью, где указана ссылка. Обратите внимание на то, что все пробелы в вопросе необходимо заменить на знаком - .
- `[link name](<.qna file path>#*answers*)` : находит все ответы и добавляет их в список фразы продолжительностью, где указана ссылка.

Ниже приведен пример упомянутых выше ссылок.

```
> You can include references to other .lu files

[All LU files](./all.lu)

> References to other files can have wildcards in them

[en-us](./en-us/*)

> References to other lu files can include subfolders as well.
> /** indicates to the parser to recursively look for .lu files in all subfolders as well.

[all LU files](../**)

> You can include deep references to intents defined in a .lu file in utterances

# None
- [None uttearnces](./all.lu#Help)

> With the above statement, the parser will parse all.lu and extract out all utterances associated with the 'Help' intent and add them under 'None' intent as defined in this file.

> NOTE: This **only** works for utterances as entities that are referenced by the uttearnces in the 'Help' intent will not be brought forward to this .lu file.

# All utterances
> you can use the *utterances* wild card to include all utterances from a lu file. This includes utterances across all intents defined in that .lu file.
- [all.lu](./all.lu#*utterances*)
> you can use the *patterns* wild card to include all patterns from a lu file.
> - [all.lu](./all.lu#*patterns*)
> you can use the *utterancesAndPatterns* wild card to include all utterances and patterns from a lu file.
> - [all.lu](./all.lu#*utterancesAndPatterns*)

> You can include wild cards with deep references to QnA maker questions defined in a .qna file in utterances

# None
- [QnA questions](.//*#?)

> With the above statement, the parser will parse **all** .lu files under ./, extract out all questions from QnA pairs in those files and add them under 'None' intent as defined in this file.

> You can include deep references to QnA maker questions defined in a .qna file in utterances

# None
- [QnA questions](./qna1.qna#?)

> With the above statement, the parser will parse qna1.lu and extract out all questions from QnA pairs in that file and add them under 'None' intent as defined in this file.
```

## Ссылки URI

Ниже приведены примеры создания ссылок URI.

```
> URI to LU resource
[import](http://.../foo.lu)

# intent1
> Ability to pull in specific utterances from an intent
- [import](http://.../foo.lu#None)

# intent2
> Ability to pull in utterances or patterns or both from a specific intent 'None'
- [import](http://.../foo.lu#None*utterances*)
- [import](http://.../bar.lu#None*patterns*)
- [import](http://.../taz.lu#None*utterancesandpatterns*)

# intent3
> Ability to pull in all utterances or patterns or both across all intents
- [import](http://.../foo.lu##utterances*)
- [import](http://.../bar.lu##patterns*)
- [import](http://.../taz.lu##utterancesandpatterns*)
```

## Дополнительные сведения

- Узнайте больше о [формате QNA-файла](#).
- Прочтайте статью [Отладка с помощью адаптивных средств](#), чтобы узнать, как анализировать файлы lu.

# Формат QNA-файла

27.03.2021 • 8 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

QNA-файлы содержат напоминающие Markdown текстовые определения понятий [QnAMaker.ai](#). В этой статье описаны несколько понятий, которые выражаются в формате QNA-файла.

## Добавление комментариев

> позволяет создать комментарий. Ниже приведен пример:

```
> This is a comment and will be ignored
```

## Пары вопросов и ответов

Файл. QnA и средство синтаксического анализа поддерживают определения вопросов и ответов.

Ниже приведен синтаксис простого определения вопроса и ответа:

```
# ? Question
[list of question variations]
```
Answer
```
```

```

Ниже приведены примеры определений вопросов и ответов.

```
> # QnA Definitions
### ? who is the ceo?
```
You can change the default message if you use the QnAMakerDialog.
See [this link](https://docs.botframework.com/en-us/azure-bot-service/templates/qnamaker/#navtitle) for
details.
```

### ? How do I programmatically update my KB?
```
You can use our REST apis to manage your KB.
\#1. See here for details:
https://westus.dev.cognitive.microsoft.com/docs/services/58994a073d9e04097c7ba6fe/operations/58994a073d9e041
ad42d9baa
```
```

```

Обратите внимание на то, что идентификатор типа `markdown` для `answer` является необязательным.

## Несколько вопросов

Можно добавить несколько вопросов для одного ответа, просто добавив варианты вопроса.

```
### ? Aren't you feeling happy today?  
- Feeling cheerful?  
```markdown  
I'm quite happy, thank you.  
```
```

## Фильтры QnA Maker

Фильтры в QnA Maker — это простые пары "ключ-значение", которые можно использовать для сокращения результатов поиска, повышения приоритета ответов и хранения контекста.

Для добавления фильтров используйте следующий синтаксис:

```
**_Filters:_*  
- name = value  
- name = value
```

Ниже приведен пример использования фильтра.

```
### ? Where can I get coffee?  
- I need coffee  
  
-*Filters:  
- location = seattle  
  
```markdown  
You can get coffee in our Seattle store at 1 pike place, Seattle, WA  
```  
  
### ? Where can I get coffee?  
- I need coffee  
  
**Filters:  
- location = portland  
  
```markdown  
You can get coffee in our Portland store at 52 marine drive, Portland, OR  
```
```

## Прием PDF-файлов в QnA Maker

QnA Maker также поддерживает прием PDF-файлов во время создания базы знаний. Вы можете добавить файлы для приема в QnA Maker, используя схему URL-ссылок. Если тип содержимого URI не является текстом или HTML, то средство синтаксического анализа добавит его в коллекцию файлов для приема QnA Maker.

```
[SurfaceManual.pdf](https://download.microsoft.com/download/2/9/B/29B20383-302C-4517-A006-B0186F04BE28/surface-pro-4-user-guide-EN.pdf)
```

## Внешние ссылки

В QNA-файле можно указывать внешние ссылки, используя синтаксис ссылок Markdown.

### Ссылка на другой QNA-файл

Можно указать ссылку на другой QNA-файл с помощью `[link name](<.qna file name>)`. Ссылка может быть абсолютным путем или относительным путем от содержащего сведения QNA-файла.

## Ссылка на папку, содержащую QNA-файлы

Ссылка на папку с другими QNA-файлами поддерживается следующим образом.

- `[link name](<.qna file path>/*)` : ищет файлы. QnA по указанному абсолютному или относительному пути.
- `[link name](<.qna file path>/**)` : рекурсивно ищет файлы. QnA по указанному абсолютному или относительному пути, включая вложенные папки.

## Ссылка на URL-адрес

Можно указать ссылку на URL-адрес для приема файлов в QnAMaker во время создания базы знаний с помощью `[link name](<URL>)`.

## Ссылка из указанного файла

Можно также добавить ссылки на речевые фрагменты, определенные в заданном файле в разделе `intent` или в виде пар вопросов и ответов.

- `[link name](<.lu file path>#<INTENT-NAME>)` : находит все фразы продолжительностью, найденные в разделе <намерения-NAME> в файле Lu и добавляет их в список вопросов, где указана ссылка.
- `[link name](<.lu file path>#*utterances*)` — находит все фразы продолжительностью в файле. Lu и добавляет их в список вопросов, где указана ссылка.
- `[link name](<.qna file path>#?)` : находит вопросы от всех пар QnA, определенных в QnA-файле, и добавляет их в список фразы продолжительностью, где указана эта ссылка.
- `[link name](<.qna folder>/#?)` : находит все вопросы из всех файлов. QnA в указанной папке и добавляет их в список фразы продолжительностью, где указана эта ссылка.

Ниже приведен пример указанных выше ссылок.

```
> QnA URL reference  
[QnaURL](https://docs.microsoft.com/en-in/azure/cognitive-services/qnamaker/faqs)  
  
> Include all content in ./kb1.qna  
[KB1](./kb1.qna)  
  
> Look for all .qna files under a path  
[ChitChat](./chitchat/*)  
  
> Recursively look for .qna files under a path including subfolders.  
[ChitChat](../chitchat/resources/**)
```

## Описание модели

В QNA-файл можно добавить сведения о конфигурации для приложения LUIS или базы знаний QnA Maker, чтобы указать средству синтаксического анализа, как правильно обработать содержимое LU-файла.

Вот как можно добавить сведения о конфигурации `> ! # :`

```
> !# @<property> = <value>  
> !# @<scope>-<property> = <value>  
> !# @<scope>-<property> = <semicolon-delimited-key-value-pairs>
```

Обратите внимание на то, что любые сведения, явно переданные через аргументы интерфейса командной строки, переопределяют сведения в QNA-файле.

```
> Parser instruction - this is optional; unless specified, the parser will default to the latest version.  
> !# @version = 1.0  
  
> QnA Maker KB description  
> !# @kb.name = my qna maker kb name  
  
> Source for a specific QnA pair  
> !# @qna.pair.source = <source value>
```

## Многошаговое содержимое

Для представления многошагового содержимого в формате QNA-файла используется нотация ссылки Markdown. Ссылки указываются следующим образом:

```
- [display text](#<ID or question>)
```

При необходимости можно включить `context-only` любые запросы, которые доступны только в контексте вопроса. Ознакомьтесь с разделом о [добавлении существующей пары "вопрос-ответ"](#) в качестве запроса для дальнейших действий, чтобы узнать больше об использовании `context`.

```
- [tell me a joke](#?joke) `context-only`
```

### Запросы для дальнейших действий

У разработчиков есть два варианта создания запросов для дальнейших действий: использовать вопрос непосредственно в качестве запроса для дальнейших действий или назначить явный идентификатор паре "вопрос-ответ".

### Непосредственное использование вопроса

Первая пара "вопрос-ответ" с текстом ссылки вида `question` будет добавлена в качестве запроса. Если требуется более явное управление, используйте вместо этого [идентификаторы](#).

Если вы напрямую используете вопрос, используйте соглашение Markdown и замените пробелы дефисами (например, используйте `#?when-is-the-portland-store-open` вместо `#?when is the portland store open`). Средство синтаксического анализа приложит все усилия, чтобы найти ссылку.

```
# ?store hours
```
Most our stores are open M-F 9AM-10PM.
```
**Prompts:**  

- [Seattle store](#?seattle)  

- [Portland store](#?when-is-the-portland-store-open)

# ?seattle
```
The Seattle store is open M-F 9AM-10PM.
```
# ?when is the portland store open
- portland store hours
```
The Portland store is open 24/7.
```

```

Обратите внимание на то, что ссылка не будет отображаться как ссылка, которую можно щелкнуть, в

большинстве отрисовщиков Markdown.

### **Назначение явного идентификатора паре "вопрос-ответ"**

Назначьте каждому запросу идентификатор с номером. В приведенном ниже примере запросу для каждого магазина назначено уникальное числовое значение.

```
# ?store hours
```
Most our stores are open M-F 9AM-10PM.
```
**Prompts:** 
- [Seattle store](#1)
- [Portland store](#2)

<a id = "1"></a>

# ?seattle
```
The Seattle store is open M-F 9AM-10PM.
```

<a id = "2"></a>

# ?when is the portland store open
- portland store hours
```
The Portland store is open 24/7.
```
```

```

## Дополнительные ресурсы

- Узнайте больше о [формате LU-файла](#).

# Сущности и типы действий

27.10.2020 • 4 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Сущности являются частью действия и предоставляют дополнительную информацию о действии или общении.

## NOTE

Разные части пакета SDK определяют отдельные классы или элементы сущностей.

- Дополнительные сведения о сущностях сообщений см. в описании [типов сущностей и действий](#).
- Дополнительные сведения о сущностях распознавания LUIS см. в руководстве по [извлечению сущностей](#).

## Сущности

Свойство *entities* сообщения представляет собой массив открытых объектов [schema.org](#), что позволяет осуществлять обмен контекстно-зависимых метаданных между каналом и ботом.

### Сущности Mention

Многие каналы поддерживают способность бота или пользователя "упоминать" кого-то в контексте общения. Чтобы упомянуть пользователя в сообщении, заполните свойство сущностей сообщения *упомянутым* объектом. Объект mention содержит следующие свойства.

СВОЙСТВО	ОПИСАНИЕ
Тип	Тип сущности (Mention).
Mentioned	Объект учетной записи канала, который показывает, какой пользователь был упомянут.
текст	Текст внутри свойства <i>activity.text</i> представляет собой само упоминание (может быть пустым или иметь значение NULL).

Данный пример кода показывает, как добавить сущность mention в коллекцию сущностей.

- [C#](#)
- [JavaScript](#)

```
var entity = new Entity();
entity.SetAs(new Mention()
{
    Text = "@johndoe",
    Mentioned = new ChannelAccount()
    {
        Name = "John Doe",
        Id = "UV341235"
    }
});
entities.Add(entity);
```

**TIP**

При попытке определить намерение пользователя бот может захотеть проигнорировать ту часть сообщения, где о ней упоминается. Назовите метод `GetMentions` и оцените объекты `Mention`, которые возвращены в ответ.

## Объекты Place

Сведения, связанные с местоположением, могут быть переданы в сообщении путем заполнения свойства `entities` сообщения с помощью объекта `Place` или `GeoCoordinates`.

Объект `place` содержит следующие свойства.

СВОЙСТВО	ОПИСАНИЕ
Тип	Тип сущности ( <code>Place</code> ).
Адрес	Описание или почтовый адрес объекта (будущее)
Географические	<code>GeoCoordinates</code>
HasMap	URL-адрес карты или объект <code>map</code> (будущее)
Имя	Название расположения.

Объект `geoCoordinates` содержит следующие свойства.

СВОЙСТВО	ОПИСАНИЕ
Тип	Тип сущности ( <code>GeoCoordinates</code> ).
Имя	Название расположения.
Долгота	Долгота расположения ( <a href="#">WGS 84</a> ).
Широта	Широта расположения ( <a href="#">WGS 84</a> ).
Elevation	Высота расположения ( <a href="#">WGS 84</a> ).

Данный пример кода показывает, как добавить упоминание сущности в коллекцию.

- [C#](#)
- [JavaScript](#)

```
var entity = new Entity();
entity.SetAs(new Place()
{
    Geo = new GeoCoordinates()
    {
        Latitude = 32.4141,
        Longitude = 43.1123123,
    }
});
entities.Add(entity);
```

## Использование сущностей

- [C#](#)
- [JavaScript](#)

Чтобы использовать сущности, используйте `dynamic` ключевое слово или классы со строгой типизацией.

Данный пример кода показывает, как использовать `dynamic` ключевое слово для обработки сущности внутри `Entities` свойства сообщения.

```
if (entity.Type == "Place")
{
    dynamic place = entity.Properties;
    if (place.geo.latitude > 34)
        // do something
}
```

В этом примере кода показано, как использовать строго типизированный класс для обработки объекта в свойстве `Entities` сообщения.

```
if (entity.Type == "Place")
{
    Place place = entity.GetAs<Place>();
    GeoCoordinates geo = place.Geo.ToObject<GeoCoordinates>();
    if (geo.Latitude > 34)
        // do something
}
```

## Типы действий

Действия могут быть различных типов, самыми распространенными являются **сообщение**. Объяснения и дополнительные сведения о разных типах действий см. в [схеме действий Bot Framework](#).

# Встроенные определения политики Azure для службы Azure Bot

27.03.2021 • 2 minutes to read • [Edit Online](#)

Эта страница является индексом определений встроенных политик политики Azure для службы Azure Bot. Дополнительные встроенные компоненты Политики Azure для других служб см. в статье [Встроенные определения Политики Azure](#).

Имя каждого встроенного определения политики связано с определением политики на портале Azure. Перейдите по ссылке в столбце **Версия**, чтобы просмотреть исходный код в [репозитории GitHub для службы "Политика Azure"](#).

## Служба Azure Bot

Имя (ПОРТАЛ AZURE)	ОПИСАНИЕ	ДЕЙСТВИЕ	ВЕРСИЯ (GITHUB)
Конечная точка Службы Bot должна быть допустимым URI HTTPS	<p>Данные могут быть изменены во время передачи. Существуют протоколы, которые обеспечивают шифрование для избежания проблем неправильного использования и незаконного изменения.</p> <p>Чтобы обеспечить взаимодействие ботов только по зашифрованным каналам, задайте в качестве конечной точки допустимый URI HTTPS.</p> <p>Это гарантирует, что протокол HTTPS используется для шифрования передаваемых данных, и часто является требованием для обеспечения соответствия нормативным или отраслевым стандартам.</p> <p>Перейдите по адресу <a href="https://docs.microsoft.com/azure/bot-service/bot-builder-security-guidelines">https://docs.microsoft.com/azure/bot-service/bot-builder-security-guidelines</a>.</p>	Audit, Deny, Disabled	1.0.1

Имя	Описание	Действие	Версия
Служба Bot должна шифроваться с помощью ключа, управляемого клиентом	<p>Служба Azure Bot автоматически шифрует ваш ресурс для обеспечения защиты данных и соблюдения требований к безопасности и соответствию в организации. По умолчанию используются ключи шифрования, управляемые корпорацией Майкрософт. Для большей гибкости управления ключами или управления доступом к подписке выберите ключи, управляемые клиентом, которые также называются собственными ключами (BYOK). Подробнее о шифровании в службе Azure Bot: <a href="https://docs.microsoft.com/azure/bot-service/bot-service-encryption">https://docs.microsoft.com/azure/bot-service/bot-service-encryption</a>.</p>	Audit, Deny, Disabled	1.0.0

## Дальнейшие действия

- Ознакомьтесь со встроенными инициативами в [репозитории GitHub для Политики Azure](#).
- Изучите статью о [структуре определения Политики Azure](#).
- Изучите [сведения о действии политик](#).

# Дополнительные ресурсы Bot Framework

27.10.2020 • 2 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Эти ресурсы предоставляют дополнительные сведения и поддержку для разработки ботов с помощью Bot Framework.

## IMPORTANT

Используйте один из этих ресурсов для поддержки. Не оставляйте комментарии к этой статье. Эта статья не отслеживается для запросов в службу поддержки.

ТИП ПОДДЕРЖКИ	КОНТАКТ
<b>Поддержка сообщества</b>	Вопросы можно публиковать на форуме <a href="#">Stack Overflow</a> с помощью тега <code>botframework</code> . Обратите внимание, что на Stack Overflow существуют правила для воспроизведения проблемы, например обязательное описательное название, полная и краткая формулировка проблемы и достаточное количество сведений. В данном документе не рассматриваются запросы функций или слишком общие вопросы. Для получения дополнительных сведений новым пользователям необходимо посетить <a href="#">центр справки Stack Overflow</a> .
<b>Группа чата сообщества</b>	<a href="#">Gitter:IM</a>
<b>Использование бота</b>	Обратитесь к разработчику бота по электронной почте издателя
<b>Проблемы и рекомендации относительно пакета SDK Bot Framework</b>	Отправляйте проблемы и запросы функций в репозиторий SDK для языка, на котором написан бот ( <a href="#">C#</a> <a href="#">JavaScript</a> или <a href="#">Python</a> ). Вопросы об использовании пакета SDK можно публиковать на форуме <a href="#">Stack Overflow</a> с использованием тега <code>botframework</code> .
<b>Примеры для Bot Framework</b>	Отправляйте проблемы с примерами в репозиторий <a href="#">примеров для Bot Framework</a> .
<b>Справка и поддержка в Azure</b>	<a href="#">Справка и поддержка в Azure</a>
<b>Проблема с документацией</b>	Отправьте <a href="#">запрос</a> в репозиторий GitHub для документации по Bot Framework.
<b>Обновления документации</b>	Щелкните ссылку "Изменить" в статье и отправьте запрос на вытягивание в <a href="#">репозиторий GitHub документации по Bot Framework</a> .
<b>Сообщение о нарушении</b>	Свяжитесь с нами по адресу <a href="mailto:bf-reports@microsoft.com">bf-reports@microsoft.com</a>

# Ключи Application Insights

27.03.2021 • 3 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В Azure Application Insights данные о приложении отображаются в **ресурсе** Microsoft Azure. Чтобы добавить данные телеметрии для бота, требуется подписка Azure и ресурс Application Insights, созданный для вашего бота. Из этого ресурса можно получить три ключа для настройки бота:

1. Ключ инструментирования
2. Идентификатор приложения
3. Ключ API

В этом разделе показано, как создать ключи Application Insights.

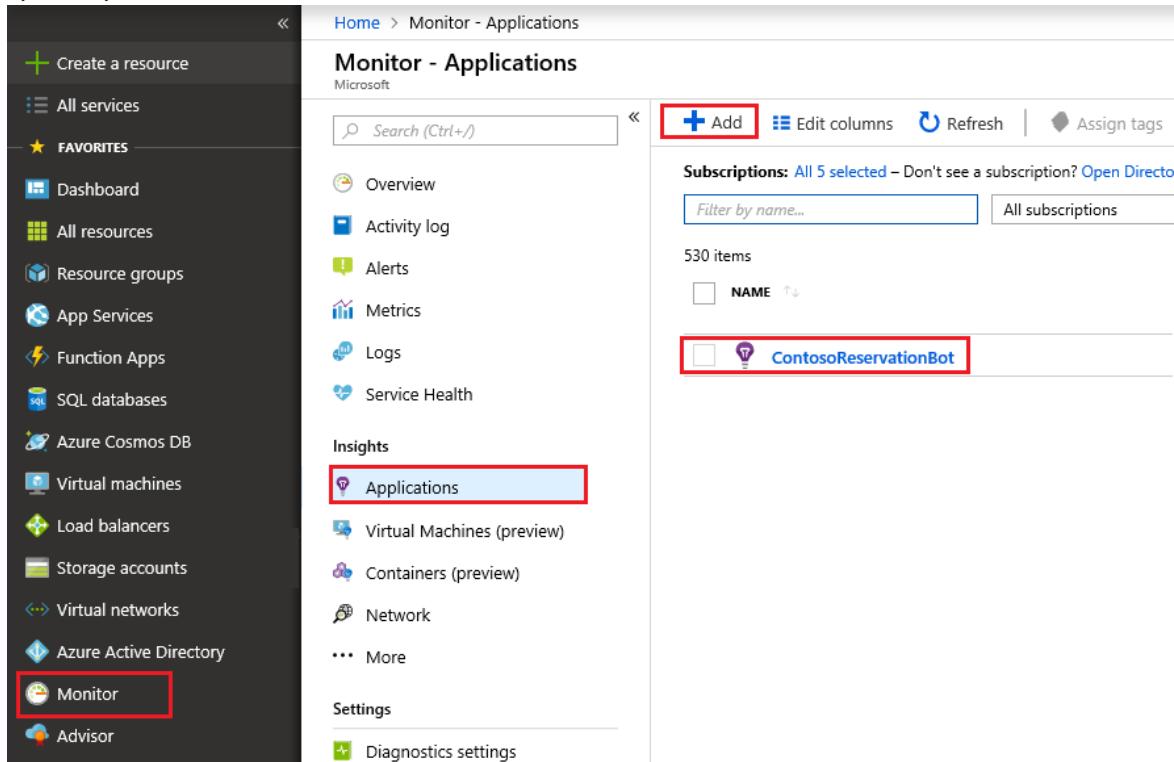
## NOTE

Во время создания или регистрации бота вы могли **включить** или **выключить** Application Insights. Если вы **включили** эту службу, у бота уже есть все необходимые ключи Application Insights. Если же вы **отключили** ее, выполните инструкции в этом разделе для создания этих ключей вручную.

## Ключ инструментирования

Чтобы получить ключ инструментирования, сделайте следующее:

1. На [портале Azure](#) в разделе "Монитор" создайте ресурс Application Insights (или используйте существующий).



2. В списке ресурсов Application Insights щелкните только что созданный ресурс Application Insight.
3. Нажмите **Обзор**.

4. Разверните блок **Основные компоненты** и найдите **ключ инструментирования**.

The first screenshot shows the Application Insights Overview page with various metrics and navigation links. The second screenshot shows the detailed application settings, including the Resource group (ContosoReservationBot), Location (West US 2), Subscription ID, and Tags.

5. Скопируйте **ключ инструментирования** и вставьте его в поле Application Insights Instrumentation Key (Ключ инструментирования Application Insights) в параметрах своего бота.

## Идентификатор приложения

Чтобы получить идентификатор приложения, сделайте следующее:

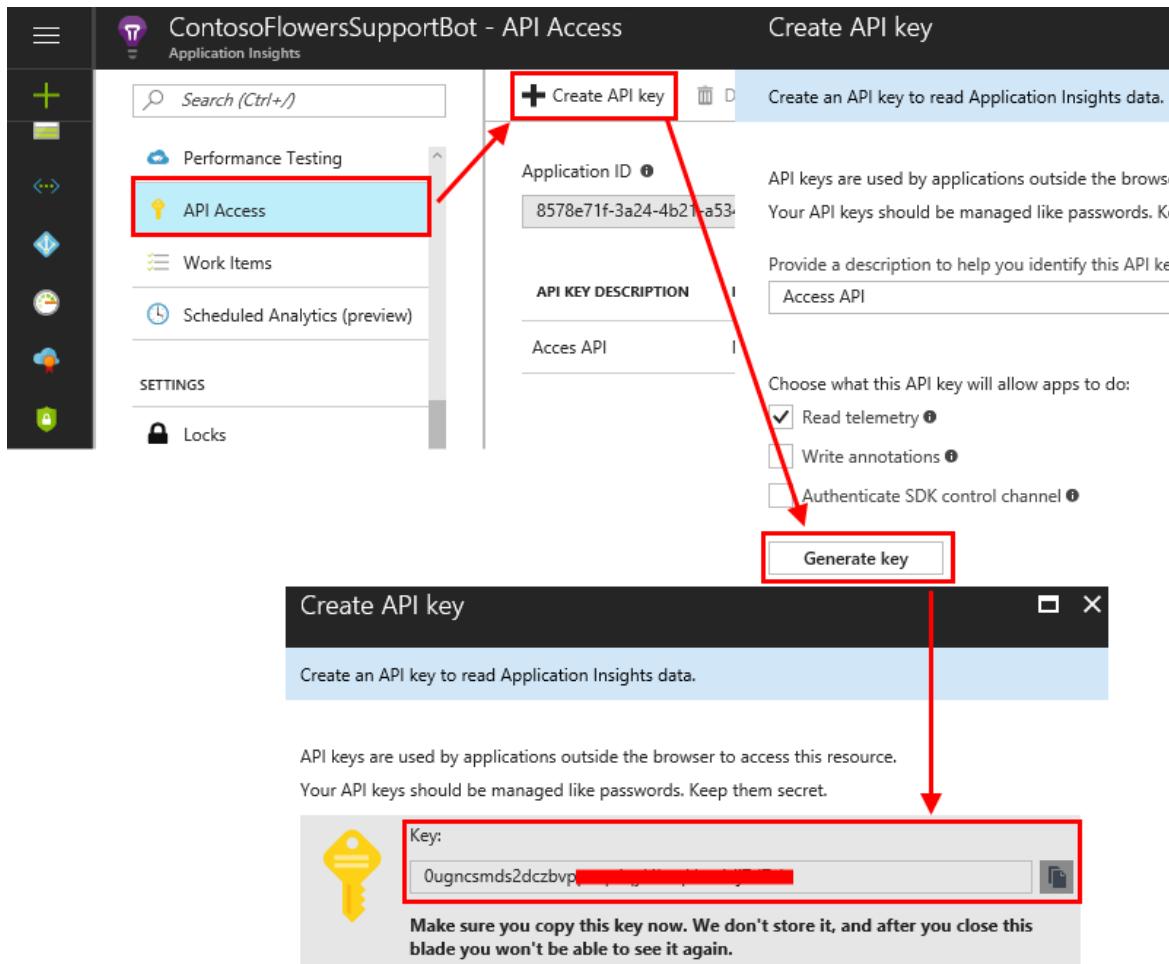
1. В ресурсе Application Insights щелкните **Доступ через API**.
2. Скопируйте **идентификатор приложения** и вставьте его в поле Application Insights Application ID (Идентификатор приложения Application Insights) в параметрах своего бота.

API KEY DESCRIPTION	LAST USED	CREATED ON	PERMIS
Access API	Never	5/15/2017	Read t

## Ключ API

Чтобы получить ключ API, сделайте следующее:

1. Из ресурса Application Insights щелкните **Доступ через API**.
2. Щелкните **Создание ключа API**.
3. Введите краткое описание, проверьте параметр **считывания телеметрии** и нажмите кнопку **создать ключ**.



#### WARNING

Скопируйте этот **ключ API** и сохраните его, так как вы больше не сможете его просмотреть. Если ключ будет утерян, понадобится создать новый.

4. Скопируйте ключ API в поле **Ключ API Application Insights** в параметрах своего бота.

## Дополнительные ресурсы

Дополнительные сведения о подключении этих полей в параметрах бота см. в разделе [Включение аналитики](#).

# Соответствие требованиям в службе Azure Bot

27.10.2020 • 3 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Служба Azure Bot является глобальной службой Azure и поэтому доступна клиентам во всех регионах в облаках, где она развернута, включая:

- Общедоступное облако Azure, которое доступно глобально.
- Azure для государственных организаций доступна в четырех регионах США для государственных учреждений США и их партнеров.

Чтобы помочь клиентам выполнить свои обязательства по соответствию требованиям в регулируемых отраслях и на рынках по всему миру, Azure поддерживает крупнейшее портфолио соответствия требованиям в отрасли как в горизонтальном направлении (общее количество предложений), так и в вертикальном (количество служб для клиентов в области оценки). Предложения для соответствия требованиям Azure сгруппированы в четыре сегмента: применимые глобально, для правительства США, отраслевые, а также зависящие от региона или страны. Предложения для соответствия требованиям основаны на различных типах гарантий, включая официальные процедуры сертификации, аттестации, проверки, авторизации и оценки, проведенные независимыми аудиторскими фирмами, а также поправки к контрактам, самостоятельные оценки и руководства для клиентов, предоставляемые корпорацией Microsoft.

## Сертификации Службы Azure Bot

Служба Azure Bot постоянно раскрывает свое покрытие сертификации. Сейчас служба Azure Bot сертифицирована со следующими сертификатами:

ГЛОБАЛЬНО ПРИМЕНИМО	US (США)	ОТРАСЛЕВЫЕ ОСОБЕННОСТИ	ЗАВИСЯЩИЕ ОТ РЕГИОНА ИЛИ СТРАНЫ
Сертификация CSA STAR	Руководство по требованиям DoD уровень 2	HIPAA BAA	Австралийская IRAP
Аттестация CSA STAR	FedRAMP — средний уровень	HITRUST	Германия C5
ISO 20000-1:2011	GxP (FDA 21 CFR, ч. 11)	PCI DSS, уровень 1	Великобритания: G-Cloud
ISO 22301:2012		WCAG 2.0	
ISO 27001:2013			
ISO 27017:2015			
ISO 27018:2014			
ISO 9001:2015			
SOC 1, 2, 3			

Дополнительные сведения о каждом предложении соответствия и их преимуществах см. в разделе Обзор страницы [соответствия Microsoft Azure](#).

В следующей таблице перечислены сертификаты, поддерживаемые службой Azure Bot в Azure для государственных организаций.

ГЛОБАЛЬНО ПРИМЕНИМО	US (США)	ОТРАСЛЕВЫЕ ОСОБЕННОСТИ
Сертификация CSA STAR	CJIS	HIPAA BAA
Аттестация CSA STAR	Руководство по требованиям DoD уровень 2	PCI DSS
SOC 1, 2, 3	Руководство по требованиям DoD, уровень 4	
	FedRAMP — высокий уровень	
	IRS 1075	
	NIST CSF	
	NIST SP 800-171	

## Дальнейшие действия

Чтобы узнать о последних сертификатах соответствия требованиям для службы Azure Bot, ознакомьтесь [с обзором соответствия требованиям Azure](#).

Дополнительные сведения о сертификации Майкрософт см. в [центре управления безопасностью Azure](#).

# Рекомендации по проверке бота

27.03.2021 • 11 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Мы ждем вас и благодарим вас за инвестиции в наработками и время создания роботов, ботлетам, веб-приложений, надстроек или навыков ("интеграции приложений") для каналов Майкрософт. Ниже приведены минимальные требования, которые должны выполняться при интеграции приложений, прежде чем они могут быть опубликованы в канале Microsoft, таком как Microsoft Teams. Каждый канал может иметь особые требования в дополнение к требованиям, описанным ниже. Если это применимо, вы найдете условия для конкретного канала на странице настройки каждого канала, и вам может потребоваться зарегистрироваться в службе канала, прежде чем можно будет опубликовать робот в этом канале.

## ПОЛИТИКИ ИНТЕГРАЦИИ ПРИЛОЖЕНИЙ

### 1. ЦЕННОСТЬ, ПРЕДСТАВЛЕНИЕ, БЕЗОПАСНОСТЬ И УДОБСТВО ИСПОЛЬЗОВАНИЯ

Интеграция приложения и связанные с ней метаданные должны:

- иметь четкие и информативные метаданные и обеспечивать полезные и качественные впечатления от использования;
- точно и четко отражать источник, функциональность и функции интеграции приложений, а также описывать любые важные ограничения;
- не должны использовать имя или значок, аналогичные именам и значкам других приложений, и не претендовать на представительство компании, государственного органа или другого лица, если нет на это разрешения;
- не подвергать риску и не нарушать безопасность пользователя или безопасность и функциональность канала Microsoft;
- не пытаться изменить или расширить функциональные возможности, описанные в нарушениях этих политик или применимых условиях канала;
- не активировать и не включать вредоносные программы;
- быть доступными;
- продолжать работать и реагировать на ввод пользователей;
- содержать рабочую ссылку на Условия предоставления услуг;
- действовать, как описано в инструкциях, профиле, условиях использования и политике конфиденциальности;
- работать в соответствии с условиями, применимыми к Microsoft Bot Framework (или другими условиями, применимыми к его разработке), и Условиями использования Microsoft (или другими применимыми терминами для канала, на котором публикуется интеграция приложений);
- информировать пользователей, если интеграция приложений включает взаимодействие с пользователем (например, обслуживание клиентов или поддержка реального пользователя);
- быть локализованными для всех поддерживаемых языков. Текст описания интеграции приложения должен быть локализован на каждом объявленном языке.

### 2. Конфиденциальность

- Если интеграция приложений обрабатывает личную информацию пользователей (личная информация включает всю информацию или данные, которые идентифицируют или могут быть использованы для идентификации пользователя или связаны с такой информацией или данными).

Примеры личной информации включают: имя и адрес, номер телефона, биометрические идентификаторы, местоположение, контакты, фотографии, аудио- и видеозаписи, документы, SMS, электронную почту или другие средства общения, скриншоты, а в некоторых случаях и комбинированную историю просмотров). Необходимо предоставить выделенную ссылку интеграции приложений на применяемую политику конфиденциальности, и она должна соответствовать всем применимым законам, правилам и требованиям политики. Эта политика должна охватывать данные, которые вы собираете или передаете, а также сведения о том, что собираетесь делать с этими данными и (если применимо) с кем будете делиться ими. Если у вас нет заявления о конфиденциальности, вот некоторые сторонние ресурсы \*, которые могут оказаться ненужными:

Будущее форума конфиденциальности — [Генератор политики конфиденциальности приложений](#)

Iubenda — [Генератор политики конфиденциальности](#)

*Вы соглашаетесь взять на себя весь риск и ответственность, возникающие в результате использования вами этих сторонних ресурсов, и что корпорация Майкрософт не несет ответственности за какие-либо вопросы, связанные с их использованием.*

- Интеграция приложения не должна собирать, хранить или передавать личные данные, не имеющие отношения к ее основной цели, без предварительного получения согласия пользователя. Необходимо получить все согласия от пользователей на обработку личной информации, если этого требует закон.
- Вы не можете опубликовать интеграцию приложений, которая направляется на детей с возрастом 13 (как определено в интерактивной службе защиты конфиденциальности в Интернете), без явного разрешения корпорации Майкрософт.

### **3. финансовые транзакции**

- Для интеграции приложений с включенной оплатой:
  - интеграция приложений не может передавать информацию финансового инструмента через пользовательский интерфейс;
  - при условии ограничения любого канала или сторонней платформы интеграция приложений может: (а) поддерживать платежи через [Microsoft Seller Center](#) в соответствии с условиями соглашения о центре продавца Microsoft; или (б) передавать ссылки на другие безопасные платежные услуги;
  - если интеграция приложений позволяет использовать вышеперечисленные механизмы оплаты, необходимо упоминать об этом в своих условиях интеграции приложений и политике конфиденциальности (и на всех профилях или веб-сайтах интеграции приложений) до того, как конечный пользователь соглашается использовать интеграцию приложений;
  - Необходимо четко указать, что для интеграции приложения в своем профиле включена оплата, и предоставить конечным пользователям сведения о клиентской службе. перетаскивани
  - вы не можете публиковать интеграцию приложений на канале Microsoft, который включает ссылки или иным образом направляет пользователей на платежные услуги для покупки цифровых товаров без прямого разрешения корпорации Майкрософт.

### **4. содержимое**

- Все содержимое интеграции приложений и связанные с ним метаданные должны быть либо изначально созданы издателем, лицензированным соответствующим образом у стороннего правообладателя, либо использоваться с разрешенными правами владельца, либо быть разрешены по закону.
- Вы не можете публиковать интеграцию приложений или содержимое интеграции приложений, которые:
  - являются незаконными;
  - используют в своих интересах детей, наносят или угрожают причинить им вред;
  - содержат рекламу, спам, нежелательные, незапрашиваемые или массовые средства связи,

публикации или сообщения;

- могут содержать неуместные или оскорбительные материалы (включая, например, обнажение, зоофилию, ненормативную лексику, порнографию или материалы откровенно сексуального характера, сцены беспричинного насилия, пропаганду курения, преступную, опасную или безответственную деятельность, употребление наркотиков, нарушение прав человека, незаконное использование оружия против человека или животного в реальном мире, безответственное использование алкогольной продукции);
  - являются ложными или вводящими в заблуждение (например, запрашивая деньги под ложными предлогами, выдавая себя за другого);
  - несут опасность для вас, канала Microsoft и других или создают риск для безопасности (например, заражение вирусами, преследование, публикация содержимого террористического направления, высказывания, направленные на разжигание ненависти или пропаганду дискриминации, ненависти или насилия по признаку расы, этнической принадлежности, национальному происхождению, языку, полу, возрасту, инвалидности, религии, сексуальной ориентации, статусу ветерана или членства в любой другой социальной группе);
  - нарушают права других пользователей, например несанкционированный обмен авторскими правами на музыку или другие материалы, защищенные авторскими правами;
  - являются клеветническими, неправдивыми, оскорбительными или угрожающими;
  - нарушают конфиденциальность других пользователей;
  - обрабатывает сведения, которые (а) связаны с условиями пациента, лечения или оплатой для обработки. (б) определяет сотрудников как или взаимодействует с пациентов, членами плана работоспособности или бенефициарами; или (с) в противном случае — "защищенная информация о работоспособности" в рамках акта о переносимости и отчетности страхования здоровья, как было изменено ("HIPAA"), или выполнять любые действия, управляемые HIPAA, если вы являетесь "охваченной сущностью" или "бизнес-связью", как определено в HIPAA;
  - являются оскорбительными в любой стране или регионе, для которых приложение является целевым; содержимое может быть воспринято как оскорбительное в определенных странах или регионах в соответствии с местным законодательством или культурными нормами;
  - дают возможность другим нарушать эти правила.
- Если вы вносите какие-либо существенные изменения в интеграцию приложений, необходимо уведомить корпорацию Майкрософт заранее, отправив электронное письмо на канал, на котором вы публикуетесь. Изменения, внесенные в регистрацию робота, могут потребовать повторного рассмотрения программы-робота, чтобы убедиться в том, что он будет продолжать отвечать требованиям, указанным здесь. Корпорация Майкрософт имеет право по своему усмотрению периодически просматривать интеграции приложений на любом канале и удалять их без предварительного уведомления.

# Разделенные на категории действия по каналам

27.03.2021 • 12 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В таблице описано, какие события (действия в сети) могут поступать из определенных каналов.

Пояснение к таблицам:

СИМВОЛ	ЗНАЧЕНИЕ
✓	Следует ожидать, что бот получит это действие
✗	Следует ожидать, что бот <b>никогда</b> не получит это действие
□	Сейчас не определено, может ли бот получать это действие

Действия можно разделить на отдельные категории. Для каждой категории приведена таблица с возможными для нее действиями.

## Диалоговые

\	DIRE CT LINE	DIRE CT LINE <b>(ВЕБ - ЧАТ)</b>	EMAIL	FACE BOOK	GRO UPM E;	KIK	KOM АНД Ы	SLAC K	SKYP E	SKYP E ДЛЯ БИЗН ЕСА	TELE GRA M	TWIL IO
Сооб щен ие	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MessageReacti on	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗

- Все каналы передают действия сообщений.
- Если используется диалог, действия сообщений, как правило, должны передаваться диалогу.
- К действиям MessageReaction это, вероятно, не относится, хотя они являются важной частью общения.
- Существует два логических типа действий MessageReaction: добавленные и удаленные.

### TIP

Реакции на сообщения — это, например, знак одобрения в отношении предыдущего комментария. Они могут возникать без определенного порядка и в этом похожи на кнопки. Это действие сейчас отправляется каналом Teams.

## Экран приветствия

\	DIRECT LINE	DIRECT LINE (WEB-ЧАТ)	EMAIL	FACEBOOK	GROUPME;	KIK	КОМАНДЫ	SLACK	SKYPE	SKYPE ДЛЯ БИЗНЕСА	TELEGRAM	TWILIO
ConversationUpdate	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
ContactRelationUpdate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

- Каналы часто отправляют действия ConversationUpdate.
- Существует два логических типа действий MessageReaction: добавленные и удаленные.
- Идея, что поведение "Приветствие" можно без усилий реализовать с помощью подключения события ConversationUpdate.Added, кажется очень соблазнительной, — и иногда это работает.
- Но это очень поверхностный подход, и чтобы поведение "Приветствие" было достаточно надежным, реализации бота, возможно, также потребуется использовать состояние.

## Расширяемость приложений

\	DIRECT LINE	DIRECT LINE (WEB-ЧАТ)	EMAIL	FACEBOOK	GROUPME;	KIK	КОМАНДЫ	SLACK	SKYPE	SKYPE ДЛЯ БИЗНЕСА	TELEGRAM	TWILIO
Event*	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>									
Event.CreateConversation	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Event.ContinueConversation	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

- Действия событий — это механизм расширяемости в Direct Line (*также называется веб-чатом*).
- Приложение, которому принадлежит клиент и сервер, может выбрать направлять собственные события через службу с помощь этого действия события.

## Microsoft Teams

\	DIRECT LINE	DIRECT LINE (WEB-ЧАТ)	EMAIL	FACEBOOK	GROUPME;	KIK	KOMANDY	SLACK	SKYPE	SKYPE ДЛЯ БИЗНЕСА	TELEGRAM	TWILIO
Invoke.TeamsVerification	X	X	X	X	X	X	✓	X	X	X	X	X
Invoke.ComposeResponse	X	X	X	X	X	X	✓	X	X	X	X	X

- Наряду с некоторыми другими типизированными действиями, Microsoft Teams определяет несколько действий вызова для Teams.
- Действия вызова зависят от приложения и не определяются клиентом.
- Общее понятие об определенных подтипах действий, которые используют вызов, отсутствует.
- Вызов сейчас является единственным действием, которое запускает поведение "запрос — ответ" у бота.

Это очень важно: если для реализации запроса OAuth используются диалоги, действие Invoke.TeamsVerification нужно переадресовать диалогу.

## Обновление сообщений

\	DIRECT LINE	DIRECT LINE (WEB-ЧАТ)	EMAIL	FACEBOOK	GROUPME;	KIK	KOMANDY	SLACK	SKYPE	SKYPE ДЛЯ БИЗНЕСА	TELEGRAM	TWILIO
MessageUpdate	X	X	X	X	X	X	✓	□	X	X	X	X
MessageDelete	X	X	X	X	X	X	✓	□	X	X	X	X

- Обновление сообщений сейчас поддерживается Teams.

## OAuth

	DIRECT LINE	DIRECT LINE (ВЕБ-ЧАТ)	EMAIL	FACEBOOK	GROUPME;	KIK	KOMANDY	SLACK	SKYPE	SKYPE ДЛЯ БИЗНЕСА	TELEGRAM	TWILIO
Event.TokenResponse	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	X	□	□	□	X	□	□	□	□	□

Это очень важно: если для реализации запроса OAuth используются диалоги, действие Event.TokenResponse нужно переадресовать диалогу.

## Без категории

	DIRECT LINE	DIRECT LINE (ВЕБ-ЧАТ)	EMAIL	FACEBOOK	GROUPME;	KIK	KOMANDY	SLACK	SKYPE	SKYPE ДЛЯ БИЗНЕСА	TELEGRAM	TWILIO
EndOfConversation	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	X	X	X	X	X	X	X	X	X	X
InstallationUpdate	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	X	X	X	X	X	X	X	X	X	X
Ввод с клавиатуры	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	X	X	X	X	X	X	X	X	X	X
Handoff	X	X	X	X	X	X	X	X	X	X	X	X

## Не используются (включают вызов для платежа)

- DeleteUserData
- Invoke.PaymentRequest
- Invoke.Address
- Проверка связи

## Сводка действий, поддерживаемых каждым каналом

### Direct Line

- Сообщение
- ConversationUpdate

- Event.TokenResponse
- Event.\*
- *Event.CreateConversation*
- *Event.ContinueConversation*

## Email

- Сообщение

## Facebook

- Сообщение
- *Event.TokenResponse*

## GroupMe;

- Сообщение
- ConversationUpdate
- *Event.TokenResponse*

## Kik

- Сообщение
- ConversationUpdate
- *Event.TokenResponse*

## Команды

- Сообщение
- ConversationUpdate
- MessageReaction
- MessageUpdate
- MessageDelete
- Invoke.TeamsVerification
- Invoke.ComposeResponse

## Slack

- Сообщение
- ConversationUpdate
- *Event.TokenResponse*

## Skyre

- Сообщение
- ContactRelationUpdate
- *Event.TokenResponse*

## Skyre для бизнеса

- Сообщение

- ContactRelationUpdate
  - *Event.TokenResponse*

## Telegram

- Сообщение
  - ConversationUpdate
  - *Event.TokenResponse*

Twilio

- Сообщение

## Сводная таблица всех действий для всех каналов

\	DIRE CT LINE	DIRE CT LINE (ВЕБ - ЧАТ)	EMAIL	FACE BOOK	GRO UPM E;	KIK	KOM АНД Ы	SLAC K	SKYR E	SKYR E ДЛЯ БИЗН ЕСА	TELE GRA M	TWIL IO
Even t.Cont inue Conv ersati on	□	□	□	□	□	□	□	□	□	□	□	□
Invoke.Te ams Verifi cation	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗
Invoke.C omposeR espo nse	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗
MessageU pdat e	✗	✗	✗	✗	✗	✗	✓	□	✗	✗	✗	✗
MessageD elete	✗	✗	✗	✗	✗	✗	✓	□	✗	✗	✗	✗
Even t.Tok enRe spon se	✓	✓	✗	□	□	□	✗	□	□	□	□	□
End OfCo nvers ation	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Instal atio nUp date	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Ввод с клав иатур ы	✓	✓	✗	✓	✗	✗	✓	✓	✗	✗	✓	✗

	DIRECT LINE	DIRECT LINE (WEB-ЧАТ)	EMAIL	FACEBOOK	GROUPE;	KIK	KOMANDY	SLACK	SKYPE	SKYPE ДЛЯ БИЗНЕСА	TELEGRAM	TWILIO
Hand off	X	X	X	X	X	X	X	X	X	X	X	X

## Веб-чат.

Веб-чат отправит:

- message: с параметрами text и (или) attachments.
- event: с параметрами name and value (в виде JSON/string).
- typing: если пользователь задал параметр, а именно sendTypingIndicator, веб-чат не будет отправлять действие contactRelationUpdate. Веб-чат также не поддерживает действие messageReaction, так как мы не получали явных запросов на реализацию поддержки этой функции.

По умолчанию веб-чат выводит следующее:

- message: отображает в режиме карусели или в столбце в зависимости от параметра действия.
- typing: отображает в течение 5 с и скрывает или отображает до поступления следующего действия.
- conversationUpdate: скрывает.
- event: скрывает.
- Другое: отобразит окно предупреждения (не используется в рабочей среде). Вы можете изменить этот конвейер обработки и добавить, удалить или заменить любую пользовательскую обработку.

Вы можете использовать веб-чат для отправки любого типа действий и полезной нагрузки, но мы не документируем и не рекомендуем использовать эту возможность. Вместо этого используйте действие event.

## Поддержка действий каждым из каналов

В следующей таблице указано максимальное число рекомендуемых действий и действий карточки, поддерживаемое в каждом канале. Элемент X указывает, что действие совсем не поддерживается в конкретном канале.

	DIRECT LINE	DIRECT LINE (WEB-ЧАТ)	EMAIL	FACEBOOK	GROUPE;	KIK	ГРАФИК;	KOMANDY	SLACK	SKYPE	SKYPE ДЛЯ БИЗНЕСА	TELEGRAM	TWILIO
Рекомендации для действий	100	100	X	10	X	20	13	X	X	10	X	100	X

\	DIRECT LINE	DIRECT LINE (WEB-ЧАТ)	EMAIL	FAC EBO OK	GRO UPM E;	KIK	ГРА ФИК;	КОМ АНДЫ	SLA CK	SKY PE	SKY PE ДЛЯ БИЗНЕСА	TELEGRAM	TWILIO
ДЕЙСТВИЯ КАРТОЧКИ	100	100	X	3	X	20	99	3	100	3	X	X	X

См. сведения о [числах, указанных в приведенной выше таблице](#).

Дополнительные сведения о рекомендуемых действиях см. в статье [Использование кнопки для ввода данных](#).

Дополнительные сведения о действиях карточки см. в разделе [Отправка карточки для имиджевого баннера](#) в статье [Добавление мультимедиа в сообщения](#).

## Поддержка карт каждым из каналов

CHANNEL	АДАПТИВНАЯ КАРТА	АНИМАЦИОННАЯ КАРТА	КАРТА С АУДИО	ИМИДЖЕВАЯ КАРТА	КАРТОЧКА КВИТАНЦИИ	КАРТОЧКА ВХОДА	КАРТОЧКА С ЭСКИЗОМ	КАРТА С ВИДЕО
Email	◆	🌐	🌐	✓	✓	✓	✓	🌐
Facebook	⚠️◆	✓	X	✓	✓	✓	✓	X
GroupMe ;	◆	🌐	🌐	🌐	🌐	🌐	🌐	🌐
Kik	◆	✓	✓	X	🌐	X	✓	🌐
график;	⚠️◆	✓	🌐	✓	✓	✓	✓	🌐
Microsoft Teams	✓	X	X	✓	✓	✓	✓	X
Skype	X	✓	✓	✓	✓	✓	✓	✓
Slack	◆	✓	🌐	🌐	✓	✓	🌐	🌐
Telegram	⚠️◆	✓	🌐	✓	✓	✓	✓	✓
Twilio	◆	🌐	X	🌐	🌐	🌐	🌐	X
Веб-чат.	✓	✓	✓	✓	✓	✓	✓	✓

Примечание. Канал Direct Line технически поддерживает все карты, но клиент должен их реализовать.

- ✓ : Поддерживается — карта полностью поддерживается, но некоторые каналы поддерживают только ряд действий CardAction и (или) могут ограничивать количество действий, разрешенных для

каждой карты. Зависит от канала.

- : Частично поддерживается — карта может не отображаться, если содержит элементы для ввода данных и (или) кнопки. Зависит от канала.
- : Поддержка отсутствует.
- : Карта преобразована в изображение.
- : Карта преобразована в неформатированный текст — могут не работать ссылки, не отображаться изображения и (или) не воспроизводиться элементы мультимедиа. Зависит от канала.

Эти категории намеренно описаны кратко и не объясняют подробно, как поддерживается каждая карта в каждом канале, из-за огромного разнообразия возможных сочетаний карт, функций и каналов.

Используйте эту таблицу только для справки, но каждую карту в любом случае нужно тестировать во всех нужных каналах.

# Поля идентификаторов в Bot Framework

27.10.2020 • 11 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этом руководстве описываются характеристики полей идентификаторов в Bot Framework.

## Идентификатор канала

У каждого канала Bot Framework есть уникальный идентификатор.

Например, `"channelId": "slack"`.

Идентификаторы канала используются в качестве пространств имен для других идентификаторов. Вызовы среди выполнения в протоколе Bot Framework должны осуществляться в контексте канала; канал предоставляет значение разговора и идентификаторы учетных записей, используемых во время обмена.

По соглашению для идентификаторов канала используются символы нижнего регистра. Каналы обеспечивают постоянный регистр символов в идентификаторах каналов. Поэтому боты могут использовать операции сравнения на равенство с целью выявления соответствия.

### Правила для идентификаторов канала

- В идентификаторах каналов учитывается регистр символов.

## Дескриптор бота

У каждого бота, зарегистрированного в службе Azure Bot, есть дескриптор бота.

Например, `FooBot`.

Дескриптор бота представляет регистрацию бота в подключенной к Интернету службе Azure Bot. Эта регистрация связана с конечной точкой веб-перехватчика HTTP и с регистрацией в каналах.

Служба Azure Bot гарантирует уникальность дескрипторов ботов. Портал Azure выполняет проверку уникальности без учета регистра (это означает, что варианты дескриптора, отличающиеся только регистром символов, рассматриваются как один и тот же дескриптор). Это характеристика портала Azure, и она не обязательно применима к самому дескриптору бота.

### Правила использования дескрипторов ботов

- Дескрипторы ботов являются уникальными (без учета регистра) на платформе Bot Framework.

## Идентификатор приложения API.

У каждого бота, зарегистрированного в службе Azure Bot, есть идентификатор приложения.

### NOTE

Ранее приложения часто назывались "Приложениями MSA" или "Приложениями MSA/AAD". Теперь приложения часто называются просто "приложениями", но в некоторых элементах протокола приложения могут называться "Приложениями MSA".

Например, `"msaAppId": "353826a6-4557-45f8-8d88-6aa0526b8f77"`.

Приложение представляет регистрацию в группе удостоверений портала приложения и используется в качестве механизма идентификации между службами в протоколе среды выполнения Bot Framework. У приложений могут быть другие ассоциации, не связанные с ботами, например веб-сайты или мобильные и классические приложения.

У каждого зарегистрированного бота есть ровно одно приложение. Несмотря на то, что владелец бота не может самостоятельно изменить приложение, связанное с ботом, специалисты службы поддержки Bot Framework могут сделать это в небольшом количестве исключительных случаев.

Боты и каналы могут использовать идентификаторы приложений для уникальной идентификации зарегистрированных ботов.

Идентификаторы приложений гарантированно являются глобально уникальными. Все идентификаторы приложений должны сравниваться без учета регистра.

### Правила для идентификаторов приложений

- Идентификаторы приложений являются уникальными (и должны сравниваться как глобальные уникальные идентификаторы) в рамках платформы приложений Майкрософт.
- У каждого бота есть ровно одно соответствующее приложение.
- Для изменения приложения, с которым связан бот, требуется обращение в службу поддержки Bot Framework.

## Учетная запись канала

У каждого бота и пользователя есть учетная запись в каждом канале. Учетная запись содержит идентификатор (`id`) и другие информативные, но неструктурированные данные, например дополнительное имя.

Например, `"from": { "id": "john.doe@contoso.com", "name": "John Doe" }`.

Эта учетная запись описывает адрес в канале, используемый для отправки и получения сообщений. В некоторых случаях эти операции регистрации выполняются в одной службе (например, Facebook). В других случаях они находятся в нескольких системах (адреса электронной почты, номера телефонов). В более анонимных каналах (например, веб-чатах) регистрация может быть временной.

Учетные записи каналов являются вложенными в каналы. Например, учетная запись Facebook представляет собой просто число. Это число может иметь другое значение в других каналах и может не иметь смысла вне каналов.

Связь между канала учетными записями канала и пользователями (людьми) зависит от соглашений, связанных с каждым каналом. Например, номер для SMS-сообщения обычно связан с одним человеком в течение какого-то времени. По истечении этого периода времени номер может быть передан другому пользователю. И наоборот, учетная запись Facebook обычно навсегда связывается с одним пользователем, хотя одна учетная запись часто может использоваться двумя пользователями.

Для большинства каналов рекомендуется считать учетную запись канала своего рода почтовым ящиком, в который доставляются сообщения. Обычно большинство каналов позволяют сопоставить несколько адресов с одним адресом электронной почты, например, `"jdoe@contoso.com"` и `"john.doe@service.contoso.com"` могут представлять один и тот же адрес электронной почты. Некоторые каналы изменяют адрес учетной записи в зависимости от того, какие боты обращаются к этому каналу. Например, Facebook изменяет идентификаторы пользователей так, чтобы у каждого бота был собственный адрес для приема и отправки сообщений, отличающийся от других адресов.

Хотя иногда можно установить соответствие адресов, для установки соответствия адресов электронной почты и пользователей необходимо знание соглашений, действующих внутри канала, и во многих случаях это невозможно.

Бот получает адрес учетной записи канала с помощью поля `recipient` в действиях, отправляемых боту.

### Правила для учетных записей каналов

- Учетные записи канала имеют значение только в соответствующих каналах.
- Несколько идентификаторов могут представлять одну и ту же учетную запись.
- Для проверки идентификаторов на равенство можно использовать порядковое сравнение.
- Выполнить сравнение, с помощью которого можно определить, что два разных идентификатора представляют одну и ту же учетную запись, один и тот же бот или одного и того же человека, обычно невозможно.
- Устойчивость ассоциаций между идентификаторами, учетными записями, адресами электронной почты и пользователями зависит от канала.

## Идентификатор разговора

Отправка и получение сообщений происходит в контексте разговора, который определяется по идентификатору.

Например, `"conversation": { "id": "1234" }`.

Разговор включает обмен сообщениями и другие действия. Каждый разговор может содержать одно или несколько действий, и каждое действие появляется только в одном разговоре. Разговоры могут быть бессрочными или иметь четкое начало и конец. Процесс создания, изменения и завершения диалога происходит в канале (т. е. разговор происходит только тогда, когда канал знает об этом), и характеристики этих процессов устанавливаются каналом.

Действия в беседе отправляются пользователями и ботами. Определение "участников" разговора различается в зависимости от канала. К участникам разговора могут относиться пользователи, присутствующие в канале, пользователи, которые когда-либо получали сообщение, и пользователи, которые отправили сообщение.

Некоторые каналы (например, SMS и некоторые другие) отличаются тем, что идентификатор канала, который назначается для разговора тет-а-тет, является идентификатором учетной записи удаленного канала. У этой особенности есть два побочных эффекта:

1. Идентификатор разговора является субъективным и зависит от того, кто просматривает его. В разговоре участников А и В участник А видит идентификатор разговора "В", а участник В — идентификатор разговора "А".
2. Если у бота есть несколько учетных записей канала в этом канале (например, у бота есть два номера для SMS-сообщений), то идентификатора разговора недостаточно для уникальной идентификации разговора с точки зрения бота.

Таким образом, идентификатор разговора необязательно идентифицирует конкретный разговор в канале даже для одного бота.

### Правила для идентификаторов разговора

- Разговоры имеют значение только в соответствующих каналах.
- Несколько идентификаторов могут представлять один и тот же разговор.
- Порядковое сравнение на равенство не гарантирует, что два идентификатора разговора принадлежат одному и тому же разговору, хотя в большинстве случаев это так.

## Идентификатор действия

Отправка и получение действий осуществляется по протоколу Bot Framework, и часто действия можно определить.

Например, `"id": "5678"`.

Идентификаторы действий являются необязательными и используются в каналах для того, чтобы бот мог сослаться на идентификатор в последующих вызовах API, если они доступны:

- Ответ на определенное действие
- Запрос списка участников на уровне действия

Поскольку другие варианты использования не предусмотрены, дополнительные правила обращения с идентификаторами действий отсутствуют.

# Реализация навыка для использования в Power Virtual Agents

27.10.2020 • 6 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Навык — это бот, который может использоваться другим ботом. Таким образом можно создать однопользовательского бота и расширить его возможности одним или несколькими навыками. Чтобы узнать больше о навыках в общем, ознакомьтесь с разделом [Обзор навыков](#). Чтобы узнать об их создании, ознакомьтесь с разделом [Реализация навыка](#). Кроме того, шаблоны виртуального помощника содержат набор [готовых навыков](#), которые можно настроить и развернуть, а не создавать их с нуля.

Если вы предполагаете, что ваш навык будет использоваться на роботе с [виртуальными агентами](#), необходимо учитывать некоторые дополнительные ограничения на ваши навыки.

## Ограничения манифеста

Служба Power Virtual Agents накладывает ограничения на то, что можно объявить в [манифесте навыка](#).

- Вы можете объявить не более 25 действий.
- Каждое действие может использовать не более 25 входных или выходных данных.
- Невозможно использовать тип массива для входных или выходных данных.

## Ограничение арендаторов

Чтобы обеспечить соответствие требованиям и адекватную систему управления пользовательскими навыками, регистрируемыми для использования в Power Virtual Agents, ваш бот навыка должен быть зарегистрированным приложением в Azure Active Directory. После добавления навыка мы проверяем, что его идентификатор приложения относится к арендатору клиента, выполнившего вход, а конечная точка навыка соответствует [Home Page URL](#) зарегистрированного приложения.

Прежде чем можно будет зарегистрировать бот в качестве навыка в Power Virtual Agents, необходимо убедиться, что в качестве [начальной страницы на портале Azure](#) для бота задан URL-адрес его манифеста навыка.

## Проверка, выполненная при регистрации навыка

Чтобы подключиться к вашему навыку из бота Power Virtual Agents, пользователю сначала потребуется [импортировать этот навык в Power Virtual Agents](#). Ваш навык пройдет ряд проверок. Сбой при одной из этих проверок может привести к появлению сообщения об ошибке, как описано в этой таблице.

ЭТАП ПРОВЕРКИ	КОД ОШИБКИ	СООБЩЕНИЕ ОБ ОШИБКЕ	ОПИСАНИЕ ИЛИ УСТРАНЕНИЕ РИСКОВ
URL-адрес манифеста является допустимым	<a href="#">URL_MALFORMED</a> , <a href="#">URL_NOT_HTTPS</a>	Недопустимая ссылка. Ссылка должна начинаться с https://	Повторно введите ссылку в формате защищенного URL-адреса.

ЭТАП ПРОВЕРКИ	КОД ОШИБКИ	СООБЩЕНИЕ ОБ ОШИБКЕ	ОПИСАНИЕ ИЛИ УСТРАНЕНИЕ РИСКОВ
Манифест доступен для извлечения	MANIFEST_FETCH_FAILED	При получении манифеста навыка возникли проблемы.	Убедитесь, что URL-адрес манифеста является ссылкой на манифест. Попробуйте открыть URL-адрес манифеста в веб-браузере. Если URL-адрес отрисовывает страницу в течение 10 секунд, необходимо повторно зарегистрировать свой навык.
Манифест доступен для чтения	MANIFEST_TOO_LARGE	Манифест слишком большой.	Размер манифеста должен быть не менее 500 КБ.
Манифест доступен для чтения	MANIFEST_MALFORMED	Манифест несовместим.	Проверьте, является ли манифест допустимым JSON-файлом. Проверьте, содержит ли манифест обязательные свойства, такие как <code>name</code> , <code>msaAppId</code> и т. д. Дополнительные сведения см. в разделе <a href="#">ограничения манифеста</a> .
Навык еще не зарегистрирован	MANIFEST_ALREADY_IMPORTED	Этот навык уже добавлен в бот.	Удалите навык и зарегистрируйте его снова.
Конечная точка манифеста и домены домашней страницы совпадают	MANIFEST_ENDPOINT_ORIGIN_MISMATCH	Несоответствие в конечных точках навыка.	Домен URL-адреса домашней страницы и домен URL-адреса манифеста приложения Azure AD должен совпадать. Ознакомьтесь с <a href="#">ограничением арендаторов</a> .
Навык размещается в клиенте пользователя, выполнившего вход	APPID_NOT_IN_TENANT	Чтобы добавить навык, его необходимо сначала зарегистрировать.	Глобальный администратор должен зарегистрировать навык в организации пользователя, выполнившего вход.
Действия ограничены	LIMITS_TOO_MANY_ACTIONS	Навык может содержать до 25 действий.	В манифесте навыка определено слишком много действий навыка. Удалите действия и повторите попытку.

ЭТАП ПРОВЕРКИ	КОД ОШИБКИ	СООБЩЕНИЕ ОБ ОШИБКЕ	ОПИСАНИЕ ИЛИ УСТРАНЕНИЕ РИСКОВ
Входные параметры действия ограничены	LIMITS_TOO_MANY_INPUTS	Действия ограничены 25 входными параметрами.	Слишком много входных параметров действия навыка. Удалите несколько параметров и повторите попытку.
Выходные параметры действия ограничены	LIMITS_TOO_MANY_OUTPUTS	Действия ограничены 25 выходными параметрами.	Слишком много выходных параметров действия навыка. Удалите параметр и повторите попытку.
Число навыков ограничено	LIMITS_TOO_MANY_SKILLS	У бота может быть до 25 навыков.	В бот добавлено слишком много навыков. Удалите навык и повторите попытку.
Токен безопасности допустим	AADERROR_OTHER	Похоже, что произошла ошибка.	Могла возникнуть временная ошибка при получении маркера безопасности для активации навыка. Повторите импорт навыка.
Навык Работоспособен	ENDPOINT_HEALTHCHECK_FAILED , HEALTH_PING_FAILED	При проверке навыка произошла ошибка.	При отправке EndOfConversation действия для вашего навыка виртуальные агенты питания получили неизвестный ответ. Убедитесь, что ваш навык работает и отвечает правильно.
Навык разрешен	ENDPOINT_HEALTHCHECK_UNAUTHED	Этот навык не допускает, так как ваш робот не указан в списке.	Проверьте, добавлен ли ваш робот в список разрешений для навыка. Дополнительные сведения см. в статье <a href="#">Настройка уровня подготовки</a> для виртуальных агентов питания.

# Запросы Агента пользователя Bot Framework

27.03.2021 • 2 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Если вы читаете это сообщение, возможно, вы получили запрос от службы Microsoft Bot Framework. Это руководство поможет понять природу запросов и предпринять шаги по их прекращению (если это необходимо).

При получении запроса, скорее всего, у него был заголовок User-Agent и выглядел он следующим образом.

```
User-Agent: BF-DirectLine/3.0 (Microsoft-BotFramework/3.0 +https://botframework.com/ua)
```

Наиболее важной частью приведенной строки является идентификатор **Microsoft-BotFramework**, который используется Microsoft Bot Framework, — набор средств и служб, которые позволяют независимым разработчикам программного обеспечения создавать и управлять своими ботами.

Если вы являетесь Bot-разработчиком, возможно, вы уже знаете, почему эти запросы направляются в службу. Если нет, продолжайте читать, чтобы узнать больше.

## Причины, по которым Майкрософт связывается со службами пользователя.

Bot Framework подключает пользователей в службах разговора, таких как Facebook Messenger, к ботам, которые являются веб-серверами с REST API, работающими на конечных точках с доступом в Интернет. HTTP-вызовы ботов (также называемые вызовы веб-перехватчика) отправляются только на те URL-адреса, которые были указаны разработчиком ботов, зарегистрированным на портале разработчиков Bot Framework.

Если вы получаете незапрошенные запросы от служб Bot Framework к веб-службе, скорее всего, это связано с тем, что разработчик случайно или преднамеренно указал ваш URL-адрес в качестве обратного вызова перехватчика для своего робота.

## Отключение получения запросов

Чтобы получить помощь по отключению нежелательных запросов от веб-службы, обратитесь в [bf-reports@microsoft.com](mailto:bf-reports@microsoft.com).

# Вопросы и ответы о платформе Bot

27.03.2021 • 5 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Ниже приведены некоторые распространенные вопросы, которые могут возникнуть. Если вы не нашли нужный ответ, вы можете разместить свои вопросы на [Stack overflow](#) с помощью `botframework` тега. Если вы являетесь новым пользователем, сначала посетите [Центр справки Stack overflow](#).

## Общие сведения и доступность

- [Почему корпорация Майкрософт разработала Bot Framework?](#)
- [Как запустить Bot в автономном режиме?](#)
- [Как перенести службу Azure Bot из одного региона в другой?](#)
- [Что такое пакет SDK версии 4?](#)
- [Поддержка жизненного цикла SDK Bot Framework версии 3 и уведомление о прекращении использования](#)

## Общие сведения о платформе Bot

- [Почему действие ввода с клавиатуры не дает никаких результатов?](#)
- [Чем отличается библиотека соединителя от библиотеки построителя в пакете SDK?](#)
- [Каким образом сообщения пользователя связаны с вызовами методов HTTPS?](#)
- [Каким образом можно перехватывать все сообщения между пользователем и моим ботом?](#)
- [Что собой представляет метод IDialogStack.Forward в пакете SDK Bot Framework для .NET?](#)
- [Какова разница между понятиями "упреждающий" и "реагирующий"?](#)
- [Как отправить упреждающее сообщение пользователю?](#)
- [Каким образом можно ссылаться на несерIALIZУЕМЫЕ службы из диалогов C# в пакете SDK версии 3?](#)
- [Что такое ETag? Как он связан с хранилищем контейнера данных Bot?](#)
- [Что является собой ограничения частоты?](#)
- [Как возникает ограничение частоты?](#)
- [Каковы ограничения частоты?](#)
- [Как узнать, влияет ли ограничение скорости?](#)
- [Как реализовать человеческий прием?](#)
- [Каков предельный размер файла, переданного с помощью канала?](#)

## Экосистема

- [Разделы справки разрешить эмулятору подключаться к localhost, не находясь за корпоративным прокси-сервером?](#)
- [Когда добавятся дополнительные возможности общения для Bot Framework?](#)
- [У меня есть канал связи, который можно настроить с помощью платформы Bot. Можно ли работать с корпорацией Майкрософт для этого?](#)
- [Если требуется создать бот для Microsoft Teams, какие инструменты и службы следует использовать?](#)
- [Как создать бот, использующий центр обработки данных для US Gov организаций?](#)
- [Что такое канал Direct Line?](#)

- Как Bot Framework связан с Cognitive Services?
- Каковы действия по настройке веб-разговора и прямой линии для Azure для государственных организаций?
- Какие возможные обрабатываемые компьютером решения существуют для встроенных сущностей даты, времени, длительности и установки в службе LUIS?
- Что можно сделать, если требуемое количество намерений LUIS превышает максимально допустимое значение?
- Каким образом можно использовать несколько моделей LUIS?
- Где можно получить дополнительную помощь по LUIS?

## Безопасность и конфиденциальность

- Собираются ли программы-роботы, зарегистрированные в среде Bot, собраны персональные данные? Если да, как обеспечить безопасность и безопасность данных? Как насчет конфиденциальности?
- Могу ли я разместить бот на своих серверах?
- Как вы блокируете или удаляете боты из службы?
- Как с помощью платформы Bot управлять удостоверениями и доступом?
- Разделы справки запретить использование программы Bot пользователям, принадлежащим только к моему клиенту?
- Какие конкретные URL-адреса нужно разрешить в моем корпоративном брандмауэре для доступа к службам платформы Bot?
- Можно ли заблокировать весь входящий трафик бота, кроме трафика от Bot Framework Service?
- Какая роль RBAC нужна для создания и развертывания бота?
- Какие есть способы защиты от клиентов, олицетворяющих Bot Framework Service?
- Какова цель волшебного кода во время проверки подлинности?

## Azure

- Как это сделать?
- Какие файлы необходимо заархивировать для развертывания?
- Какую версию Azure CLI следует использовать для развертывания Bot?
- Что делать при получении Azure CLI ошибок устаревания?
- Что представляют собой Нерекомендуемые команды CLI, связанные с `az deployment`?
- Разделы справки определить, являются ли команды Azure CLI устаревшими?











# Устранение неполадок индекса

27.03.2021 • 3 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Ниже приведены некоторые распространенные вопросы, связанные с проблемами, которые могут возникнуть. Если вы не нашли нужный ответ, вы можете разместить свои вопросы на [Stack overflow](#) с помощью `botframework` тега. Если вы являетесь новым пользователем, посетите [Центр справки Stack overflow](#).

## Общие сведения об устранении неполадок

- [Как можно устранить неполадки с ботом?](#)
- [Как устранить неполадки, связанные с проверкой подлинности?](#)
- [Я использую пакет SDK для платформы Bot для .NET. Как устранить неполадки с моим Bot?](#)
- [Разделы справки проверить сетевое подключение между программы-роботы и каналом?](#)
- [Почему возникает ошибка с кодом состояния HTTP 429 "Слишком много запросов"?](#)
- [Почему сообщения моего бота не попадают к пользователю?](#)
- [Как выполнять фоновые задачи в ASP.NET?](#)
- [Мой Bot медленнее реагирует на первое полученное сообщение. Как сделать это быстрее?](#)
- [Как гарантировать определенный порядок доставки сообщений?](#)
- [Почему отбрасываются части текста сообщения?](#)
- [Каким образом можно поддерживать несколько ботов в одной конечной точке службы ботов?](#)
- [Как работают идентификаторы в Bot Framework?](#)
- [Как получить доступ к идентификатору пользователя?](#)
- [Почему имена пользователей Facebook больше не отображаются?](#)
- [Почему бот Kik отвечает: "К сожалению, я не могу сейчас говорить"?](#)
- [Как использовать прошедшие проверку подлинности службы из бота?](#)
- [Как можно предоставить доступ к боту только предопределенному списку пользователей?](#)
- [Почему диалог Direct Line 1.1 начинается заново после каждого сообщения?](#)
- [По какой причине служба Direct Line 3.0 отвечает с кодом состояния HTTP 502 "Недопустимый шлюз"?](#)
- [Почему при создании бота возникает исключение Authorization\\_RequestDenied?](#)
- [Почему не удается перенести бот?](#)
- [Где можно получить дополнительную помощь?](#)

## Конфигурация

- [Тестирование в веб-чате](#)
- [Бот не работает в веб-чате.](#)
- [Бот работает в веб-чате, но не работает в других каналах.](#)
  - [Проблемы с конфигурацией канала](#)
  - [Особое поведение для канала](#)
  - [Сбой канала](#)

## Ошибки HTTP 500

- Включение Application Insights в ASP.NET
- Включение Application Insights для Node.js
- Запрос исключений
- Отсутствие событий Application Insights

## Аутентификация

- ИДЕНТИФИКАТОР и пароль приложения
- Шаг 1. Отключение системы безопасности и тестирование на локальном компьютере
- Шаг 2. Проверка идентификатора и пароля приложения Bot
- Шаг 3. Включение безопасности и тестирование на localhost
- Шаг 4. Тестирование ленты в облаке

# Общие сведения об устранении неполадок

27.03.2021 • 22 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Ответы на эти часто задаваемые вопросы помогут вам устранить распространенные проблемы, связанные с разработкой или эксплуатацией ботов.

## Как можно устранить неполадки с ботом?

1. Выполните отладку исходного кода бота в [Visual Studio Code](#) или [Visual Studio](#).
2. Протестируйте Bot с помощью [эмулятора](#) перед его развертыванием в облаке.
3. Разверните бот на облачной платформе размещения, например Azure, а затем проверьте возможность подключения к боту с помощью встроенного элемента управления веб-чата на панели мониторинга бота на [портале Azure](#). Если после развертывания в Azure с ботом возникнут проблемы, просмотрите следующую запись блога: [Understanding Azure troubleshooting and support](#) (Общие сведения об устранении неполадок и включении поддержки в Azure).
4. Исключите [аутентификацию](#) как возможную проблему.
5. Протестируйте работу бота в Web Chat, Teams или любом другом канале, который вы будете использовать с этим ботом. Это поможет проверить полное взаимодействие с пользователем.
6. Протестируйте бот в каналах с требованиями дополнительной проверки подлинности, таких как Direct Line или веб-чат.
7. Воспользуйтесь инструкциями по [отладке бота](#) и прочитайте другие статьи об отладке в этом разделе.

## Как устранить неполадки, связанные с проверкой подлинности?

Дополнительные сведения об устранении неполадок бота, связанных с аутентификацией, см. в статье [Устранение неполадок проверки подлинности Bot Framework](#).

## Разделы справки проверить сетевое подключение между программой-роботы и каналом?

Вы можете использовать IP-адреса, созданные в следующих шагах, чтобы проверить, блокирует ли правило подключение с этими адресами. См. раздел [Проверка трассировки брандмауэра на неудачных подключениях](#).

### Проверка подключения из Bot к каналу

1. В браузере перейдите на [портал Azure](#).
2. Выберите службу приложений Bot, подключение которой нужно протестировать.
3. В левой области в разделе **средства разработки** выберите **Дополнительные инструменты**.
4. На правой панели щелкните Go ( [Переход](#)). Отобразится страница сведений о KUDU.
5. В верхней строке меню щелкните **консоль отладки**. Затем в раскрывающемся меню выберите команду cmd. Откроется консоль веб-приложения KUDU Bot. Дополнительные сведения см. в разделе [KUDU](#).

/ + | 5 items |

Name	
	ASP.NET
	data
	LogFiles
	site
	SiteExtensions

Kudu Remote Execution Console  
Type 'exit' then hit 'enter' to get a new CMD process.  
Type 'cls' to clear the console

Microsoft Windows [Version 10.0.14393]  
(c) 2016 Microsoft Corporation. All rights reserved.

D:\home>

6. Запустите nslookup directline.botframework.com и проверьте, работает ли разрешение DNS.

Обратите внимание, что `nslookup` (поиск сервера имен) — это программа командной строки для сетевого администрирования, которая выполняет запрос к службе доменных имен (DNS) для получения сопоставления имен доменов или IP-адресов или других записей DNS. Если разрешение DNS работает, ответ на эту команду будет содержать соответствующую информацию.

```
Non-authoritative answer:  
Server: Unknown  
Address: 168.63.129.16  
  
Name: waws-prod-bay-119.cloudapp.net  
Address: 104.40.11.192  
Aliases: directline.botframework.com  
         bc-directline.trafficmanager.net  
         bc-directline-westus2.azurewebsites.net  
         waws-prod-bay-119.sip.azurewebsites.windows.net
```

Средство просмотра протокола IP-адресов Whois полезно для получения сведений об IP-адресах.

7. Выполните команду `curl -I directline.botframework.com`. (Параметр `-I` используется для получения ответа, содержащего только заголовок.) Убедитесь, что возвращено состояние HTTP 301. Это проверка наличия подключения.

```
D:\home>curl -I directline.botframework.com
% Total    % Received % Xferd  Average Speed   Time     Time     Time  Current
          Dload  Upload   Total   Spent    Left  Speed
 0      0  0  0  0  0  0  --::-- --::-- --::--  0HTTP/1.1 301 Moved Permanently
Content-Length: 159
Content-Type: text/html; charset=UTF-8
Location: https://directline.botframework.com/
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Strict-Transport-Security: max-age=31536000; includeSubDomains
Date: Mon, 14 Dec 2020 00:36:15 GMT
```

## Проверка подключения канала к Bot

Так как оно не имеет доступа к рабочему сайту и `directline.botframework.com` находится в общедоступном Интернете, необходимо использовать в режиме имитации фигурную скобку. Выполните описанные ниже действия за пределами виртуальной частной сети (VNET), например с помощью *гиперобъекта* сотового телефона. См. также [виртуальную сеть Azure](#).

- Выполните команду `nslookup ivr-sr-bot.botapps.amat.com`. Разрешение DNS работает, если ответ на эту команду содержит релевантную информацию.

```
Non-authoritative answer:  
Server: UnKnown  
Address: 168.63.129.16  
  
Name: ivr-sr-bot.botapps.amat.com  
Address: 152.135.164.117
```

2. Запустите `curl -I https://ivr-sr-bot.botapps.amat.com/api/messages` И проверьте, возвращен ли соответствующий код состояния HTTP (например, метод 405 не разрешен). Метод, указанный в запросе, не разрешен для ресурса, идентифицируемого по указанному универсальному коду ресурса (URI). Это лишь способ проверить наличие подключения.

```
D:\home>curl -I https://ivr-sr-bot.botapps.amat.com/api/messages  
% Total    % Received % Xferd  Average Speed   Time     Time      Time  Current  
          Dload  Upload Total   Spent    Left Speed  
0       0       0       0       0       0  --::-- --::-- --::--  @HTTP/2 405  
allow: POST  
content-length: 0  
server: restify  
x-powered-by: ASP.NET  
set-cookie: ARRAffinity=5c9ddcbc4658c5cee63f1b3a0667a796a95cae019fb612774ebb204c4bfdd373;Path=/;HttpOnly;Secure;Domain=ivr-sr-bo  
t.botapps.amat.com  
set-cookie: ARRAffinitySameSite=5c9ddcbc4658c5cee63f1b3a0667a796a95cae019fb612774ebb204c4bfdd373;Path=/;HttpOnly;SameSite=None;Se  
cure;Domain=ivr-sr-bot.botapps.amat.com  
date: Mon, 14 Dec 2020 00:38:08 GMT
```

3. Если вы не получаете ответ от Bot, запишите IP-адрес клиента.

#### Проверка трассировки брандмауэра на неудачных подключениях

Используйте IP-адреса из `nslookup ivr-sr-bot.botapps.amat.com` и `nslookup directline.botframework.com`. Проверьте, блокирует ли правило подключение с этими адресами в любом направлении.

## Я использую пакет SDK Bot Framework для .NET. Как можно устранить неполадки с ботом?

**Просмотрите исключения.** В Visual Studio 2019 щелкните **Отладка > Windows > Параметры исключений**. В окне **Параметры исключений** установите флажок **Break When Thrown** (Прерывать при возникновении) рядом с полем **Исключения среды CLR**. При наличии возникших или необработанных исключений в окне выходных данных могут отображаться результаты диагностики.

**Просмотрите стек вызовов.** В Visual Studio можно выбрать вариант отладки **Только мой код**. После изучения всего стека вызовов вы сможете глубже понять любую проблему.

**Убедитесь, что все методы диалога заканчиваются планом по обработке следующего сообщения.** Все шаги диалога должны передавать управление следующему шагу каскада или завершать текущий диалог для выхода из стека. Если шаг обрабатывается неправильно, беседа будет проходить не так, как вы ожидаете. Прочтайте статью о [концепциях диалогов](#), чтобы лучше изучить эту тему.

## Почему возникает ошибка с кодом состояния HTTP 429 "Слишком много запросов"?

Сообщение об ошибке с кодом состояния HTTP 429 указывает, что в течение заданного промежутка времени было отправлено слишком много запросов. Текст ответа должен содержать описание проблемы. В нем также может указываться минимальный обязательный интервал между запросами. Среди прочего, эта ошибка может возникать из-за средства [ngrok](#). Если вы используете бесплатный план и достигли пределов ngrok, перейдите на веб-страницу с ценами и ограничениями, чтобы изучить дополнительные [варианты](#).

## Почему сообщения моего бота не попадают к пользователю?

Действие сообщения, созданное в ответе, должно быть правильно устранено, в противном случае оно не будет доставлено на предполагаемое место назначения. В подавляющем большинстве случаев это не нужно обрабатывать явным образом; пакет SDK берет на себя адресацию действий сообщения.

Правильная адресация действия подразумевает наличие правильного *идентификатора беседы* и сведений об отправителе. В большинстве случаев действие сообщения отправляется в ответ на поступившее сообщение. Это означает, что все данные для направления можно получить из входящего действия.

Изучив трассировки или журналы аудита, можно убедиться в правильности направления сообщений. Если что-то работает неправильно, создайте в боте точку останова и проверьте, где для сообщения задаются идентификаторы.

## Как выполнять фоновые задачи в ASP.NET?

В некоторых случаях может потребоваться запустить асинхронную задачу, которая в течение нескольких секунд находится в режиме ожидания, а затем выполняет код для очистки профиля пользователя или сброса состояния диалога или беседы. Дополнительные сведения о том, как это сделать, см. в статье о [выполнении фоновых задач в ASP.NET](#). В частности, рассмотрите возможность использования `HostingEnvironment.QueueBackgroundWorkItem`.

## Мой бот медленно реагирует на первое получаемое сообщение. Как повысить его быстродействие?

Боты представляют собой веб-службы, и некоторые платформы размещения, включая Azure, автоматически вводят службу в спящий режим, если она не получает трафик в течение определенного периода времени. Если это происходит с вашим ботом, его необходимо перезапустить с нуля при очередном получении сообщения, значительно замедляющего его ответ по сравнению с тем, как если бы он был уже запущен.

Некоторые платформы для размещения позволяют настроить службу таким образом, чтобы ее было невозможно перевести в спящий режим. Если бот размещается в службе веб-приложений Azure Bot, перейдите к службе бота на [портале Azure](#), щелкните **Параметры приложения**, а затем — **Всегда включено**. Этот параметр доступен в большинстве, но не во всех планах обслуживания.

## Как гарантировать определенный порядок доставки сообщений?

По возможности Bot Framework будет поддерживать порядок доставки сообщений. Например, если вы отправили сообщение А и ожидаете завершения этой операции HTTP, прежде чем инициировать новую операцию HTTP для отправки сообщения В. Некоторые каналы, такие как SMS и электронная почта, не гарантируют порядок доставки сообщений на устройство пользователя.

## Почему отбрасываются части текста сообщения?

Bot Framework и многие каналы интерпретируют текст так, как если бы он был отформатирован с помощью [Markdown](#). Проверьте, содержит ли текст символы, которые могут быть интерпретированы как синтаксис Markdown.

## Каким образом можно поддерживать несколько ботов в одной конечной точке службы ботов?

В этом [примере](#) показана настройка `Conversation.Container` с правильным классом `MicrosoftAppCredentials` и использование простого `MultiCredentialProvider` для проверки подлинности нескольких идентификаторов приложений и паролей.

## Идентификаторы

### Как работают идентификаторы в Bot Framework?

Дополнительные сведения об идентификаторах в Bot Framework см. в [этом руководстве](#).

### Как получить доступ к идентификатору пользователя?

Каналы Bot Framework предоставляют идентификатор пользователя в поле `from.Id` для любого действия, отправленного пользователем. SMS-сообщения и сообщения электронной почты предоставляют в этом свойстве необработанный идентификатор пользователя. В других каналах свойство `from.Id` маскируется и содержит уникальный идентификатор пользователя, который отличается от идентификатора пользователя в канале. Чтобы подключиться к существующей учетной записи, можно использовать карточку для входа и реализовать собственный поток OAuth для подключения идентификатора пользователя к своему ИД пользователя службы.

### Почему имена пользователей Facebook больше не отображаются?

Вы меняли пароль для Facebook? Если да, маркер доступа будет недействителен, и вам потребуется обновить параметры конфигурации бота для канала Facebook Messenger на [портале Azure](#).

### Почему бот Kik отвечает: "К сожалению, я не могу сейчас говорить"?

Боты для Kik поддерживаются для 50 подписчиков. После того как с ботом пообщалось 50 уникальных пользователей, любой новый пользователь, который пытается завести разговор с ботом, получит сообщение "К сожалению, я не могу сейчас говорить". Дополнительные сведения см. в [документации по Kik](#).

### Как использовать прошедшие проверку подлинности службы из бота?

Сведения о проверке подлинности Azure Active Directory см. в руководстве по [добавлению проверки подлинности в Bot](#).

#### NOTE

При добавлении в бот функций проверки подлинности и безопасности следует убедиться, что шаблоны, реализуемые в коде, соответствуют стандартам безопасности, которые подходят для вашего приложения.

### Как можно предоставить доступ к боту только предопределенному списку пользователей?

Некоторые каналы, такие как SMS и электронная почта, предоставляют неограниченный диапазон адресов. В этих случаях сообщения от пользователя будут содержать неформатированный идентификатор пользователя в свойстве `from.Id`.

Другие каналы, такие как Facebook и временной резерв, предоставляют адреса с указанием областей или клиентов таким образом, чтобы программа-робот не могла предсказать идентификатор пользователя заранее. В этих случаях будет необходимо проверить подлинность пользователя через ссылку для входа

или общий секрет, чтобы определить, авторизованы ли они для использования бота.

## Почему диалог Direct Line 1.1 начинается заново после каждого сообщения?

### NOTE

Этот раздел не применим к последней версии протокола Direct Line 3.0

Если диалог Direct Line начинается заново после каждого сообщения, скорее всего, свойство `from` отсутствует или имеет значение `null` в сообщениях, которые клиент Direct Line отправил боту. Когда клиент Direct Line отправляет сообщение с отсутствующим или имеющим значение `null` свойством `from`, служба Direct Line автоматически выделяет идентификатор, поэтому каждое отправляемое клиентом сообщение будет считаться сообщением от нового пользователя.

Чтобы устранить эту проблему, задайте свойству `from` в каждом сообщении, отправленном клиентом Direct Line, постоянное значение, однозначно представляющее пользователя, который отправляет сообщение. Например, если пользователь уже выполнил вход на веб-страницу или в приложение, этот существующий идентификатор можно использовать в качестве значения свойства `from` в сообщениях, отправляемых пользователем. Кроме того, можно создать произвольный идентификатор пользователя при загрузке страницы или загрузке приложения, сохранить этот идентификатор в файле cookie или состоянии устройства и использовать его в качестве значения свойства `from` в сообщениях, отправляемых пользователем.

## По какой причине служба Direct Line 3.0 отвечает с кодом состояния HTTP 502 "Недопустимый шлюз"?

Direct Line 3.0 возвращает код состояния HTTP 502 при попытке связаться с ботом, но запрос завершается ошибкой. Эта ошибка означает, что либо бот вернул ошибку, либо истекло время ожидания запроса. Чтобы получить дополнительные сведения об ошибках, возникающих в боте, перейдите к панели мониторинга бота на [портале Azure](#) и щелкните ссылку "Проблемы" для затронутых каналов. Если для бота настроена служба Application Insights, в ней вы также сможете найти подробные сведения об ошибке.

## Почему при создании бота возникает исключение Authorization\_RequestDenied?

Управление разрешениями на создание ботов службы ботов Azure осуществляется на портале Azure Active Directory (AAD). Если на [портале AAD](#) неправильно настроены разрешения, при попытке создать службу бота пользователи будут получать исключение `Authorization_RequestDenied`.

Сначала проверьте, являетесь ли вы гостевым пользователем каталога.

1. Войдите на [портал Azure](#).
2. Выберите **Все службы** и выполните поиск по слову *активный*.
3. Выберите Azure Active Directory.
4. Выберите раздел **Пользователи**.
5. В списке найдите пользователя и убедитесь, что в колонке **Тип пользователя** не указано значение **Гость**.

Users - All users  
microsoft - Azure Active Directory

NAME	USER NAME	USER TYPE
C	[REDACTED]	Member
&	[REDACTED]	Owner
🌐	[REDACTED]	Guest
q	[REDACTED]	Member
🌐	[REDACTED]	Guest

После проверки, подтверждающей, что вы не являетесь **гостевым пользователем**, нужно убедиться, что пользователи в Active Directory могут создавать службу ботов. Для этого администратору каталога необходимо настроить следующие параметры.

1. Войдите на [портал AAD](#). В колонке **Пользователи и группы** выберите **Параметры пользователей**.
2. В разделе **Регистрация приложения** задайте параметру **Пользователи могут регистрировать приложения** значение **Да**. После этого пользователи в вашем каталоге смогут создавать службу ботов.
3. В разделе **Внешние пользователи** задайте параметру **Разрешения для гостей ограничены** значение **Нет**. После этого гостевые пользователи в вашем каталоге смогут создавать службу ботов.

The screenshot shows the Azure Active Directory admin center interface. On the left, there's a sidebar with various service links. The 'Users and groups' link is highlighted with a red box and has a red arrow pointing to it from below. Below this, the 'User settings' link is also highlighted with a red box. The main content area is titled 'Users and groups - User settings'. It contains several sections: 'Enterprise applications', 'App registrations', 'External users', and 'Administration portal'. In each of these sections, there are configuration options with 'Yes' and 'No' buttons. Three specific options are highlighted with red boxes: 'Users can register applications' (under App registrations), 'Guest users permissions are limited' (under External users), and 'Restrict access to Azure AD administration portal' (under Administration portal).

## Почему не удается перенести бот?

Если бот зарегистрирован на dev.botframework.com и вы хотите перенести его в Azure, проблемы с миграцией бота могут возникать в том случае, если бот входит в каталог, отличный от каталога по умолчанию. Попробуйте выполнить следующие действия.

1. Из целевого каталога добавьте нового пользователя (с помощью адреса электронной почты), который не входит в каталог по умолчанию, назначьте пользователю роль участника для подписок, являющихся целью миграции.
2. На портале разработки добавьте адрес электронной почты пользователя в качестве совладельца робота, который необходимо перенести. Затем выполните выход.
3. Войдите на портал разработчика в качестве нового пользователя и начните процесс миграции бота.

## Где можно получить дополнительную помощь?

- Используйте сведения из ранее данных ответов на вопросы на сайте [Stack Overflow](#) или разместите свои вопросы с помощью тега `botframework`. Обратите внимание, что на Stack Overflow существуют правила для воспроизведения проблемы, например обязательное описательное название, полная и краткая формулировка проблемы и достаточное количество сведений. В данном документе не рассматриваются запросы функций или слишком общие вопросы. Для получения дополнительных сведений новым пользователям необходимо посетить [центр справки Stack Overflow](#).

- Чтобы найти информацию по известным проблемам с пакетом SDK Bot Framework или сообщить о новой проблеме, перейдите на соответствующую страницу для [BotBuilder](#) на сайте GitHub.
- Используйте сведения в обсуждениях сообщества BotBuilder на сайте [Gitter](#).

# Устранение неполадок, связанных с конфигурацией бота

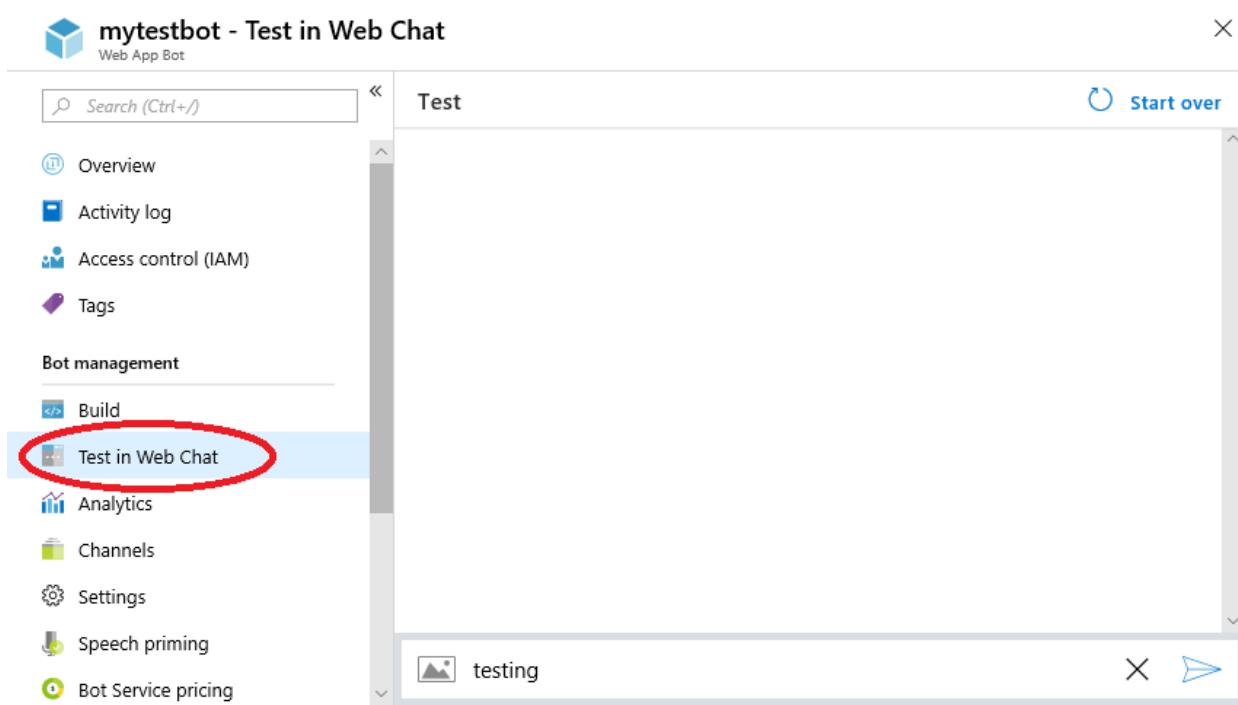
27.03.2021 • 6 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

При выполнении бота могут возникать ошибки разных типов, например он может не отвечать на запросы, создавать исключения или работать не во всех подключенных каналах. При устранении неполадок с ботом прежде всего проверьте его работу в веб-чате. Это позволит понять, с чем связана проблема: с самим ботом (если он не работает ни в одном канале) или только с определенным каналом (если бот успешно работает в некоторых каналах и не работает в других).

## Тестирование в веб-чате

1. Откройте ресурс бота на [портале Azure](#).
2. Откройте панель **Test in Web Chat** (Тестирование в веб-чате).
3. Отправьте боту сообщение.



Если бот не возвращает ожидаемые выходные данные, перейдите к разделу [Бот не работает в веб-чате](#). В противном случае перейдите к разделу [Бот работает в веб-чате, но не работает в других каналах](#).

## Бот не работает в веб-чате

Неработоспособность бота может быть вызвана разными причинами. Наиболее вероятно, что приложение бота не работает и не может принимать сообщения. Также возможно, что бот получает сообщения но не может ответить. Причины могут быть следующими:

- бот не работает или недоступен;
- бот аварийно завершает работу;
- неправильно указана конечная точка бота;

- бот успешно получает сообщения, но не может на них ответить.

Чтобы определить, работает ли бот, сделайте следующее.

1. Откройте панель **Обзор**.
2. Скопируйте значение **Messaging endpoint** (Конечная точка обмена сообщениями) и вставьте его в адресную строку браузера.

Если эта конечная точка возвращает ошибку HTTP 404 или 405, значит бот доступен и может отвечать на сообщения. Чтобы выяснить причину длительного ожидания, см. сведения о [превышении времени ожидания и завершении работы с ошибкой HTTP 5xx](#).

Если конечная точка возвращает сообщение об ошибке "This site can't be reached" (Этот сайт недоступен) или "Не удается открыть эту страницу", значит бот не работает и его следует развернуть повторно.

## Бот работает в веб-чате, но не работает в других каналах

Если бот успешно работает веб-чате, но не может работать в каких-либо других каналах, проверьте следующие возможные причины.

- Устранение неполадок, связанных с конфигурацией бота
  - Тестирование в веб-чате
  - Бот не работает в веб-чате.
  - Бот работает в веб-чате, но не работает в других каналах.
    - Проблемы с конфигурацией канала
    - Особое поведение для канала
    - Сбой канала
  - Дополнительные ресурсы

### Проблемы с конфигурацией канала

Вполне возможно, что параметры конфигурации канала (например, имя пользователя и пароль бота) заданы неправильно или изменились во внешней среде. Например, бот настроен на взаимодействие с определенной страницей Facebook, которая позднее была удалена. В этом случае проще всего удалить такой канал и заново настроить конфигурацию для него.

Ниже приведены ссылки на инструкции по настройке каналов, поддерживаемых платформой Bot Framework.

- [Alexa](#)
- [Direct Line](#)
- [Электронная почта](#)
- [Facebook](#)
- [GroupMe](#)
- [Kik](#)
- [Microsoft Teams](#)
- [Skype](#)
- [Skype для бизнеса](#)
- [Slack](#)
- [Telegram](#)
- [Twilio](#)

### Особое поведение для канала

Возможно, в реализации разных каналов различаются некоторые функции. Например, сейчас не все

каналы поддерживают адаптивные карточки. Большинство каналов поддерживают действия (кнопки), но отображают их по-разному. Заметив различия в поведении определенных типов сообщений в разных каналах, ознакомьтесь со сведениями в статье [Разделенные на категории действия по каналам](#).

Ниже приведены некоторые ссылки с дополнительной информацией по отдельным каналам.

- [Add bots to Microsoft Teams apps](#) (Добавление ботов в приложения Microsoft Teams)
- [Facebook: Introduction to the Messenger Platform](#) (Facebook: общие сведения о платформе Messenger)
- [Сведения о Skype для разработчиков](#)
- [Slack: Enabling interactions with bots](#) (Slack: организация взаимодействия с ботами)

### Сбой канала

В некоторых случаях может прерываться обслуживание отдельных каналов. Обычно такие простоя не продолжаются долго. Но если вы считаете, что произошел сбой канала, проверьте сведения на веб-сайте этого канала или в социальных сетях.

Есть еще один способ быстро проверить наличие сбоев в канале: создайте тестовый бот (например, простейший повторитель сообщений) и добавьте в него проблемный канал. Если тестовый бот успешно работает с частью каналов, но не работает с другими каналами, проблема не в рабочем боте.

## Дополнительные ресурсы

См. инструкции по [отладке бота](#) и другие статьи об отладке в этом разделе.

# Устранение неполадок при ошибках HTTP 500

27.03.2021 • 6 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Если вы столкнулись с ошибками HTTP 500, первым шагом для устранения неполадок будет включение Application Insights.

Примеры AppInsights см. в примере примеров кода на [C# Luis-with-AppInsights](#) и [JS](#).

См. подробнее о добавлении Application Insights к существующему боту в инструкциях по [использованию телеметрии аналитики бесед](#).

## Включение Application Insights в ASP.NET

Базовая поддержка Application Insights описана в статье [Настройка Application Insights для веб-сайта ASP.NET](#). Платформа Bot Framework (начиная с версии 4.2) предоставляет дополнительный уровень телеметрии для Application Insights, но без него можно обойтись при диагностике ошибок HTTP 500.

## Включение Application Insights для Node.js

Базовая поддержка Application Insights описана в статье [о мониторинге служб и приложений Node.js с помощью Application Insights](#). Платформа Bot Framework (начиная с версии 4.2) предоставляет дополнительный уровень телеметрии для Application Insights, но без него можно обойтись при диагностике ошибок HTTP 500.

## Запрос исключений

Анализ ошибок с кодом состояния HTTP 500 проще всего начать с исключений.

Следующие запросы возвращают последние исключения:

```
exceptions
| order by timestamp desc
| project timestamp, operation_Id, appName
```

Выберите несколько идентификаторов операций из результатов первого запроса и получите по ним дополнительные сведения:

```
let my_operation_id = "d298f1385197fd438b520e617d58f4fb";
let union_all = () {
    union
        (traces | where operation_Id == my_operation_id),
        (customEvents | where operation_Id == my_operation_id),
        (requests | where operation_Id == my_operation_id),
        (dependencies | where operation_Id == my_operation_id),
        (exceptions | where operation_Id == my_operation_id)
};

union_all
| order by timestamp desc
```

Если в списке есть только `exceptions`, проанализируйте подробные сведения и сопоставьте эти данные

со строками вашего кода. Если все исключения поступают только от соединителя канала (`Microsoft.Bot.ChannelConnector`), выполните анализ отсутствия событий Application Insights и убедитесь, что служба Application Insights настроена правильно и в коде правильно регистрируются события.

## Отсутствие событий Application Insights

Если возникают ошибки с кодом 500, но в Application Insights не поступает больше никаких событий от бота, проверьте следующее.

### Проверка локального выполнения бота

Убедитесь, что программа-робот сначала выполняется локально с эмулятором.

### Проверка копирования файлов конфигурации (только для .NET)

Убедитесь, что файл `appsettings.json` и любые другие файлы конфигурации правильно упаковываются в процессе развертывания.

#### Сборки приложения

Убедитесь, что сборки Application Insights правильно пакуются в процессе развертывания.

- Microsoft.ApplicationInsights
- Microsoft.ApplicationInsights.TraceListener
- Microsoft.AI.Web
- Microsoft.AI.WebServer
- Microsoft.AI.ServeTelemetryChannel
- Microsoft.AI.PerfCounterCollector
- Microsoft.AI.DependencyCollector
- Microsoft.AI.Agent.Intercept

Убедитесь, что файл `appsettings.json` и любые другие файлы конфигурации правильно упаковываются в процессе развертывания.

#### appsettings.json

Убедитесь, что в файле `appsettings.json` задан ключ инструментирования.

- [ASP.NET Web API](#)
- [ASP.NET Core](#)

```
{  
    "Logging": {  
        "IncludeScopes": false,  
        "LogLevel": {  
            "Default": "Debug",  
            "System": "Information",  
            "Microsoft": "Information"  
        },  
        "Console": {  
            "IncludeScopes": "true"  
        }  
    }  
}
```

### Проверка файла конфигурации

Убедитесь, что в файл конфигурации включен ключ Application Insights.

```
{  
    "ApplicationInsights": {  
        "type": "appInsights",  
        "tenantId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",  
        "subscriptionId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",  
        "resourceGroup": "my resource group",  
        "name": "my appinsights name",  
        "serviceName": "my service name",  
        "instrumentationKey": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",  
        "applicationId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",  
        "apiKeys": {},  
        "id": ""  
    }  
},
```

## Проверка журналов

Bot ASP.NET и node будут создавать журналы на уровне сервера, который можно проверить.

### Настройка просмотра журналов в браузере

1. Откройте бот на [портале Azure](#).
2. Откройте страницу **Параметры службы приложений** > All App service settings (Все параметры службы приложений), чтобы проверить все параметры службы.
3. Откройте страницу **мониторинга и журналов диагностики** для службы приложений.
  - Убедитесь, что включен параметр **Вход в приложения (файловая система)**. Не забудьте щелкнуть **Сохранить**, если придется изменить этот параметр.
4. Перейдите на страницу **мониторинга и потока журналов**.
  - Выберите **Журналы веб-сервера** и убедитесь, что появляется сообщение об успешном подключении. Это выглядит примерно так:

```
Connecting...  
2018-11-14T17:24:51 Welcome, you are now connected to log-streaming service.
```

Оставьте это окно открытым.

### Настройка перезапуска службы бота в браузере

1. Откройте другое окно браузера и перейдите в нем к странице бота на портале Azure.
2. Откройте страницу **Параметры службы приложений** > All App service settings (Все параметры службы приложений), чтобы проверить все параметры службы.
3. Перейдите на страницу **Обзор** для службы приложений и щелкните **Перезапуск**.
  - Если появится запрос подтверждения, выберите ответ **Да**.
4. Вернитесь в первое окно браузера и проверьте записи в журналах.
5. Убедитесь, что новые записи журналов успешно поступают.
  - Если здесь нет никаких сообщений о действиях, разверните бот повторно.
  - Теперь перейдите на страницу **Журналы приложений** и проверьте наличие ошибок.

# Устранение неполадок проверки подлинности Bot Framework

27.03.2021 • 16 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Это руководство поможет при устранении неполадок проверки подлинности с помощью бота, оценивая ряд сценариев, чтобы определить, где существует проблема.

## NOTE

Чтобы выполнить все действия, описанные в этом руководстве, необходимо загрузить и использовать [Bot Framework Emulator](#) и иметь доступ к параметрам регистрации бота на [портале Azure](#).

## Идентификатор приложения и пароль

Безопасность бота настраивается с помощью **идентификатора приложения Microsoft** и **пароля приложения Microsoft**, которые вы получаете при регистрации бота с помощью Bot Framework. Эти значения обычно указаны в файле конфигурации бота и используются для получения маркеров доступа из службы учетной записи Microsoft.

Если это не сделано, [разверните бот в Azure](#), чтобы получить **идентификатор и пароль приложения Майкрософт**, которые можно использовать для проверки подлинности.

## NOTE

Сведения о том, как найти значения AppID и AppPassword для развернутого бота, см. в разделе [MicrosoftAppID](#) и [MicrosoftAppPassword](#).

## Шаг 1. Отключение системы безопасности и тестирование на локальном компьютере

На этом шаге бот доступен и функционирует на локальном компьютере, когда отключена система безопасности.

## WARNING

Отключение системы защиты бота может позволить неизвестным злоумышленникам выдавать себя за пользователей. Если вы работаете в защищенной среде отладки, используйте только следующую процедуру.

### Отключение системы безопасности

Чтобы отключить систему защиты бота, измените его параметры конфигурации и удалите значения идентификатора и пароля приложения.

- [C#](#)
- [JavaScript](#)
- [Python](#)

Если вы используете пакет SDK для Bot Framework для .NET, добавьте или измените параметры `appsettings.js` в файле:

```
"MicrosoftAppId": "",  
"MicrosoftAppPassword": ""
```

## Тестирование бота на локальном компьютере

Протестируйте бот на локальном компьютере с помощью Bot Framework Emulator.

1. Запустите бот на локальном компьютере.
2. Установите Bot Framework Emulator.
3. Подключитесь к Bot с помощью эмулятора.
  - Введите `http://localhost:port-number/api/messages` в адресную строку эмулятора, где **номер порта** соответствует номеру порта, указанному в браузере, где выполняется приложение.
  - Убедитесь, что поля **идентификатора приложения Microsoft** и **пароля приложения Microsoft** пусты.
  - Нажмите кнопку **Соединить**.
4. Чтобы проверить возможность подключения к Bot, введите какой-либо текст в эмулятор и нажмите клавишу ВВОД.

Если бот реагирует на входные данные и в окне чата нет ошибок, значит бот доступен и функционирует на локальном узле при выключеной системе безопасности. Перейдите к [Шагу 2](#).

Если в окне чата указаны одна или несколько ошибок, щелкните по ошибке, чтобы узнать подробности. Часто возникающие проблемы.

- Параметры эмулятора указывают неправильную конечную точку для Bot. Убедитесь, что указан правильный номер порта в URL-адресе и правильный путь в конце URL-адреса (например, `/api/messages`).
- Параметры эмулятора указывают конечную точку Bot, которая начинается с `https`. На локальном узле конечная точка должна начинаться с `http`.
- Параметры эмулятора указывают значение поля **идентификатора приложения Microsoft** и/или поля **пароля Microsoft App**. Оба поля должны быть пустыми.
- Система безопасности бота не была отключена. [Проверьте](#), чтобы не было значений идентификатора или пароля приложения в настройках бота.

## Шаг 2. Проверка идентификатора и пароля приложения бота

На этом шаге проверяется допустимость идентификатора и пароля приложения бота, которые используются для проверки подлинности. (Если вы не знаете этих значений, [получите их](#) сейчас.)

### WARNING

Приведенные ниже инструкции отключают проверку SSL для `login.microsoftonline.com`. Выполните эту процедуру только в защищенной сети и после этого измените пароль приложения.

### Выдача HTTP-запроса к службе входа в систему Microsoft

Эти инструкции описывают, как использовать `cURL` для HTTP-запроса к службе входа в систему Microsoft. Можно использовать альтернативный инструмент, такой как Postman, просто убедитесь, что запрос соответствует [протоколу проверки подлинности](#) Bot Framework.

Чтобы убедиться, что идентификатор и пароль приложения бота действительны, выполните следующий

запрос, используя cURL, заменив `APP_ID` и `APP_PASSWORD` идентификатором и паролем бота.

#### TIP

Пароль может содержать специальные символы, и тогда описанный ниже вызов будет недопустимым. В этом случае попробуйте преобразовать пароль в формат URL-адреса.

```
curl -k -X POST https://login.microsoftonline.com/botframework.com/oauth2/v2.0/token -d  
"grant_type=client_credentials&client_id=APP_ID&client_secret=APP_PASSWORD&scope=https%3A%2F%2Fapi.botframew  
ork.com%2F.default"
```

Этот запрос пытается обменять идентификатор и пароль бота на маркер доступа. Если запрос будет успешным, вы получите полезные данные JSON, которые среди прочих содержат свойство `access_token`.

```
{"token_type": "Bearer", "expires_in": 3599, "ext_expires_in": 0, "access_token": "eyJ0eXAiKQ1Q..."}
```

Если запрос выполняется успешно, нужно подтвердить, что идентификатор приложения и пароль, указанный в запросе, являются допустимыми. Перейдите к [Шагу 3](#).

Если пришло сообщение об ошибке в ответ на запрос, проверьте его, чтобы определить причину. Если ответ указывает, что идентификатор приложения или пароль недопустимы, [получите правильные значения](#) с портала Bot Framework и повторно отправьте запрос с новыми значениями, чтобы подтвердить, что они действительны.

## Шаг 3. Включение системы безопасности и тестирования на локальном компьютере

На этом шаге проверяется доступность и функциональность бота на локальном компьютере, когда система безопасности отключена, и подтверждается, что идентификатор приложения и пароль, которые бот будет использовать для аутентификации, действительны. На этом шаге убедитесь, что бот доступен и работает на локальном компьютере, когда включена система защиты.

### Включение системы безопасности

Безопасность бота зависит от служб Microsoft, даже если бот работает только на локальном компьютере. Чтобы включить систему защиты для бота, отредактируйте его параметры конфигурации, чтобы заполнить идентификатор и пароль приложения с помощью значений, которые были подтверждены на [Шаге 2](#). Кроме того, проверьте актуальность пакетов, особенно `System.IdentityModel.Tokens.Jwt` и `Microsoft.IdentityModel.Tokens`.

Если вы используете пакет SDK Bot Framework для .NET, заполните эти параметры в `appsettings.config` или аналогичные значения в файле `appsettings.json`:

```
<appSettings>  
  <add key="MicrosoftAppId" value="APP_ID" />  
  <add key="MicrosoftAppPassword" value="PASSWORD" />  
</appSettings>
```

Если используется пакет SDK Bot Framework для Node.js, заполните эти параметры (или обновите соответствующие переменные среды).

```
var connector = new builder.ChatConnector({  
    appId: 'APP_ID',  
    appPassword: 'PASSWORD'  
});
```

#### NOTE

Сведения о том, как найти значения AppID и AppPassword для бота, см. в разделе [MicrosoftAppID](#) и [MicrosoftAppPassword](#).

### Тестирование бота на локальном компьютере

Протестируйте бот на локальном компьютере с помощью Bot Framework Emulator.

1. Запустите бот на локальном компьютере.
2. Установите Bot Framework Emulator.
3. Подключитесь к Bot с помощью эмулятора.
  - Введите `http://localhost:port-number/api/messages` в адресную строку эмулятора, где **номер порта** соответствует номеру порта, указанному в браузере, где выполняется приложение.
  - Введите идентификатор приложения бота в поле **Идентификатор приложения Microsoft**.
  - Введите пароль приложения бота в поле **Пароль приложения Microsoft**.
  - Нажмите кнопку **Соединить**.
4. Чтобы проверить возможность подключения к Bot, введите какой-либо текст в эмулятор и нажмите клавишу ВВОД.

Если бот реагирует на входные данные и в окне чата нет ошибок, значит бот доступен и работает на локальном компьютере, когда включена система безопасности. Перейдите к [Шагу 4](#).

Если в окне чата указаны одна или несколько ошибок, щелкните по ошибке, чтобы узнать подробности. Часто возникающие проблемы.

- Параметры эмулятора указывают неправильную конечную точку для Bot. Убедитесь, что указан правильный номер порта в URL-адресе и правильный путь в конце URL-адреса (например, `/api/messages`).
- Параметры эмулятора указывают конечную точку Bot, которая начинается с `https`. На локальном узле конечная точка должна начинаться с `http`.
- В параметрах эмулятора поле **идентификатор приложения Microsoft** и/или **пароль приложения Microsoft** не содержат допустимые значения. Оба поля должны быть заполнены и содержать соответствующие значения, которые были проверены на [Шаге 2](#).
- Для бота система безопасности не включена. Убедитесь, что настройки конфигурации бота задают значения как для идентификатора приложения, так и для пароля.

### Шаг 4. Тестирование бота в облаке

На этом шаге вы подтвердили, что бот доступен и функционирует на локальном узле, когда система безопасности отключена. Функционирование бота на локальном узле подтверждает, что идентификатор и пароль приложения бота являются допустимыми, когда включена система безопасности. На этом шаге разверните бот в облаке и убедитесь, что он доступен и работает там, где включена защита.

#### Развертывание бота в облаке

Bot Framework требует, чтобы боты были доступны из Интернета, поэтому нужно развернуть бот на платформе облачного размещения, такой как Azure. Не забудьте включить систему безопасности для бота до развертывания, как описано на [Шаге 3](#).

## NOTE

Если у вас еще нет поставщика услуг размещения в облаке, вы можете зарегистрироваться для доступа к [бесплатной учетной записи](#).

Если разворачивать бот в Azure, протокол SSL будет автоматически настроен для приложения, тем самым позволяя Bot Framework требовать конечную точку HTTPS. Если используется другой провайдер облачного размещения, убедитесь, что приложение настроено для протокола SSL, чтобы у бота была конечная точка HTTPS.

### Тестирование бота

Чтобы протестировать бот в облаке с включенной системой безопасности, выполните следующие действия.

1. Убедитесь, что бот успешно развернут и запущен.
2. Войдите на [портал Azure](#).
3. На портале перейдите к колонке **Регистрация каналов ботов**.
4. На панели слева **Управление ботами** щелкните элемент **Тестирование в веб-чате**.
5. Чтобы проверить подключение к боту, введите текст в элемент управления "Веб-чат" и нажмите ВВОД.

Если в окне "Веб-чат" появится ошибка, используйте сообщение об ошибке, чтобы определить ее причину. Часто возникающие проблемы.

- Некорректная **Конечная точка обмена сообщениями**, указанная на странице **Параметры** для бота на портале Bot Framework. Убедитесь, что указан правильный путь в конце URL-адреса (например, `/api/messages`).
- **Конечная точка обмена сообщениями**, указанная на странице **Параметры** для бота на портале Bot Framework, не начинается с `https` или не поддерживается платформой Bot Framework. Бот должен иметь допустимую цепочку сертификатов системы безопасности.
- Бот имеет отсутствующие или неверные значения для идентификатора или пароля приложения. [Проверьте](#), чтобы настройки конфигурации бота определяли допустимые значения идентификатора и пароля приложения.

Если бот соответствующим образом реагирует на входные данные, нужно подтвердить, что бот доступен и успешно работает в облаке с включенной системой безопасности. На этом шаге бот готов безопасно [подключиться к каналу](#), такому как Facebook Messenger, Direct Line и др.

## Дополнительные ресурсы

Если проблема не исчезла после выполнения действий, описанных выше, можно сделать следующее.

- Воспользуйтесь инструкциями по [отладке бота](#) и другими статьями об отладке в этом разделе.
- [Выполните отладку бота в облаке](#) с помощью Bot Framework Emulator и ПО для туннелирования [ngrok](#). *ngrok не является продуктом корпорации Майкрософт*.
- Используйте инструменты прокси, например [Fiddler](#), для проверки трафика HTTPS через бот. *Fiddler не является продуктом корпорации Майкрософт*.
- См. сведения о [технологиях аутентификации, которые использует Bot Framework](#).
- Получите помощь от других пользователей, используя [вспомогательные](#) ресурсы Bot Framework.

# Обзор виртуального помощника

27.03.2021 • 2 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Продолжая подход с открытым кодом корпорации Майкрософт к пакету SDK для Bot Framework, решение Virtual-Source с открытым кодом предоставляет набор основных базовых возможностей и полный контроль над процессом и данными пользователей.

В его основе виртуальный помощник (доступный в C# и TypeScript) — это шаблон проекта с рекомендациями по разработке программы-робота на Microsoft Azureной платформе.

- Шаблон виртуального помощника включает в себя множество практических рекомендаций, описанных в статье Создание диалоговых и автоматизирующих интеграцию компонентов, которые очень полезны для разработчиков-роботов.
- Навыки работы с Bot-инфраструктурой — это повторно используемые строительные блоки навыков, охватывающие случаи использования в беседах, позволяющие вам добавлять широкие возможности в программу-робот в течение нескольких минут. Виртуальный помощник предоставляет ряд навыков.

Ознакомьтесь с дополнительными сведениями о виртуальном помощнике в [документации по решениям для Bot Framework](#).

# Общие сведения о веб-чате

27.03.2021 • 10 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье содержатся сведения о компоненте [веб-чата Bot Framework](#). Компонент веб-чата Bot Framework представляет собой веб-клиент с широкими возможностями настройки для пакета SDK для Bot Framework версии 4. Пакет SDK для Bot Framework версии 4 позволяет разработчикам моделировать беседы и создавать сложные приложения-боты.

Если вы хотите перейти с использования веб-чата версии 3 на использование версии 4, перейдите к [этому разделу](#).

## Использование

### NOTE

Сведения о предыдущих версиях веб-чата (версия 3) см. [на сайте GitHub](#).

Сначала создайте бот с помощью службы [Azure Bot](#). После этого вам необходимо [получить секрет веб-чата бота](#) на портале Azure. Затем с помощью секрета [создайте маркер](#) и передайте его в свой веб-чат.

В следующих примерах показано, как добавить элемент управления "веб-чат" на веб-сайт.

```
<!DOCTYPE html>
<html>
  <body>
    <div id="webchat" role="main"></div>
    <script src="https://cdn.botframework.com/botframework-webchat/latest/webchat.js"></script>
    <script>

      // Set style options.
      const styleOptions = {
        botAvatarInitials: 'BF',
        userAvatarInitials: 'WC'
      };

      window.WebChat.renderWebChat(
        {
          directLine: window.WebChat.createDirectLine({
            token: 'YOUR_DIRECT_LINE_TOKEN'
          }),
          userID: 'YOUR_USER_ID',
          username: 'Web Chat User',
          locale: 'en-US',
          styleOptions
        },
        document.getElementById('webchat')
      );
    </script>
  </body>
</html>
```

Передавать параметры `userID`, `username`, `locale`, `botAvatarInitials` и `userAvatarInitials` в метод `renderWebChat` необязательно. Дополнительные сведения о стиле см. в разделе [Зачем стилемоптионс?](#)

## Интеграция с помощью JavaScript

Веб-чат разработан для интеграции с имеющимся веб-сайтом с помощью JavaScript или React. Интеграция с JavaScript предоставляет поддержку стилей и настроек. Дополнительные сведения см. в статье [об интеграции WebChat с веб-сайтом](#).

Вы можете воспользоваться полным стандартным пакетом веб-чата, содержащим часто используемые компоненты.

```
<!DOCTYPE html>
<html>
  <body>
    <div id="webchat" role="main"></div>
    <script src="https://cdn.botframework.com/botframework-webchat/latest/webchat.js"></script>
    <script>
      window.WebChat.renderWebChat(
        {
          directLine: window.WebChat.createDirectLine({
            token: 'YOUR_DIRECT_LINE_TOKEN'
          }),
          userID: 'YOUR_USER_ID'
        },
        document.getElementById('webchat')
      );
    </script>
  </body>
</html>
```

Рабочий пример полного пакета веб-чата можно найти [на сайте GitHub](#).

## Интеграция с помощью React

Для полноценной настройки вы можете использовать [React](#), чтобы реконструировать компоненты веб-чата.

Чтобы установить производственную сборку из прм, выполните команду

```
npm install botframework-webchat .
```

```
import { DirectLine } from 'botframework-directlinejs';
import React from 'react';
import ReactWebChat from 'botframework-webchat';

export default class extends React.Component {
  constructor(props) {
    super(props);

    this.directLine = new DirectLine({ token: 'YOUR_DIRECT_LINE_TOKEN' });
  }

  render() {
    return (
      <ReactWebChat directLine={ this.directLine } userID='YOUR_USER_ID' />
      element
    );
  }
}
```

Кроме того, с помощью команды `npm install botframework-webchat@master` можно установить сборку разработки, синхронизированную с веткой `master` веб-чата на сайте GitHub.

Ознакомьтесь с рабочим примером [веб-чата, созданного с помощью React](#), на сайте GitHub.

#### TIP

Если вы не знакомы с React и JSX, воспользуйтесь обучающими материалами на странице [о начале работы с React](#).

## Настройка пользовательского интерфейса веб-чата

Разработка веб-чата предполагает настройку без разветвления исходного кода. В приведенной ниже таблице указано, какие настройки можно выполнять, импортируя веб-чат различными способами. Этот список не является исчерпывающим.

НАСТРОЙКА	ПАКЕТ CDN	REACT
Изменение цвета	✓	✓
Изменение размеров	✓	✓
Обновление и замена стилей CSS	✓	✓
Прослушивание событий	✓	✓
Взаимодействие с веб-страницей размещения	✓	✓
Настраиваемые действия для преобразования для просмотра		✓
Настраиваемые вложения для преобразования для просмотра		✓
Добавление компонентов пользовательского интерфейса		✓
Реконструирование всего пользовательского интерфейса		✓

Ознакомьтесь с дополнительными сведениями [о настройке веб-чата на сайте GitHub](#).

#### NOTE

Сведения о сетях доставки содержимого (CDN) см. в [этой статье](#).

## Переход с использования версии 3 веб-чата на использование версии 4

Существует три возможных способа перехода с использования версии 3 на использование версии 4.

Сначала Сравните свой начальный сценарий с указанным ниже.

- Чтобы обновить элемент управления веб-чат, внедренный в `<iframe>`, см. документацию в репозитории веб-чата для [пакета внедрения](#).
- Для обновления элемента управления "веб-чат", который использует небольшие настройки, изучите

образец "Web Chat 00. Migration/a. v3-to-v4 ", описывающий этот процесс.

- Чтобы обновить разветвленную версию Web Chat v3 с большим количеством настроек, см. [руководство по миграции](#) в Web Chat.

## Справочник по API веб-чата

Существует несколько свойств, которые можно передать в компонент React веб-чата (`<ReactWebChat>`)

или метод `renderWebChat()`. Краткое описание доступных свойств см. в [справочнике по API веб-чата](#).

Кроме того, вы можете изучить исходный код, начинающийся с [packages/component/src/Composer.js](#) .

## Совместимость с браузерами

Веб-чат поддерживает две последние версии современных браузеров, таких как Chrome, Microsoft Edge и FireFox. Если вам требуется веб-чат в Internet Explorer 11, ознакомьтесь с [пакетом ES5](#) и [демонстрацией](#).

Но обратите внимание на следующее:

- Веб-чат не поддерживает Internet Explorer старше версии 11.
- Настройка, описанная в примерах, не относящихся к ES5, не поддерживается для Internet Explorer. Так как IE11 — устаревший браузер, он не поддерживает ES6, и многие примеры, использующие функции стрелок и современные возможности, необходимо будет вручную преобразовать в ES5. Если предполагается полноценная настройка приложения, настоятельно рекомендуем разработать приложение для современного браузера, такого как Google Chrome или Edge.
- Реализация поддержки примеров IE11 (ES5) для веб-чата не планируется.
  - Для клиентов, которые хотят вручную перезаписать другие примеры для работы в IE11, рекомендуем рассмотреть возможность преобразования кода из ES6+ в ES5 с использованием полизаполнений и компиляторов, таких как `babel` .

## Выполнение тестирования с помощью последних фрагментов кода веб-чата

*В настоящее время тестирование невыпущенных компонентов доступно только с помощью упаковки MyGet.*

Если вы хотите протестировать компонент или исправление ошибки, которые еще не были выпущены, укажите свой пакет веб-чата в веб-канале чата с ежедневным обновлением, а не в официальном веб-канале npmjs.

В настоящее время вы можете получить доступ к веб-каналу чата с ежедневным обновлением, подписавшись на наш веб-канал MyGet. Для этого вам нужно будет обновить реестр в своем проекте.

**Это изменение обратимо. В наших инструкциях описан процесс возвращения к подписке на официальный выпуск.**

[Подписка на получение сведений о новых фрагментах кода на сайте myget.org](#)

Для этого вы можете добавить свои пакеты, а затем изменить реестр проекта.

1. Добавьте зависимости своего проекта, кроме веб-чата.
2. В корневом каталоге проекта создайте файл `.npmrc`.
3. Добавьте в файл следующую строку: `registry=https://botbuilder.myget.org/F/botframework-webchat/npm/` .
4. Добавьте веб-чат в зависимости проекта `npm i botframework-webchat --save` .
5. Обратите внимание, что теперь в `package-lock.json` реестры указывают на MyGet. В проекте веб-чата включен вышеупомянутый прокси-сервер, который перенаправляет пакеты, отличные от MyGet, в `npmjs.com` .

Повторная подпись на официальный выпуск на сайте [npmjs.com](https://npmjs.com)

Повторная подпись требует сброса реестра.

1. Удалите `.npmrc` file.
2. Удалите корневой элемент `package-lock.json`.
3. Удалите каталог `node_modules`.
4. Переустановите пакеты с помощью `npm i`.
5. Обратите внимание, что в `package-lock.json` реестры снова указывают на <https://npmjs.com/>.

## Участие

Подробные сведения о создании проекта и рекомендации репозитория по запросам на вытягивание см. на [этой странице](#).

В рамках этого проекта действуют [правила поведения в отношении продуктов с открытым исходным кодом Майкрософт](#). Дополнительные сведения: [вопросы и ответы по правилам поведения](#). С любыми другими вопросами или комментариями обращайтесь по адресу [opencode@microsoft.com](mailto:opencode@microsoft.com).

## Сообщение о проблемах с безопасностью

О проблемах с безопасностью и ошибках следует сообщать в частном порядке по электронной почте в Центр реагирования на вопросы безопасности (MSRC) по адресу [secure@microsoft.com](mailto:secure@microsoft.com). Вы должны получить ответ в течение 24 часов. Если по какой-либо причине вы не получили ответ, свяжитесь с нами по электронной почте, чтобы убедиться, что мы получили исходное сообщение. Дополнительные сведения, включая ключ MSRC PGP, можно найти в [техническом центре безопасности](#).

(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены Все права защищены.

# Настройка веб-чата

27.03.2021 • 8 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

В этой статье подробно описано, как настроить примеры веб-чата в соответствии с требованиями бота.

## Интеграция веб-чата в веб-сайт

Следуйте инструкциям на [обзорной странице](#) для интеграции элемента управления "Веб-чат" в веб-сайт.

## Настройка стилей

Последняя версия элемента управления "Веб-чат" предоставляет широкие возможности настройки: вы можете изменять цвета, размеры, размещение элементов, добавлять пользовательские элементы и взаимодействовать с веб-страницей для размещения. Ниже приведены несколько примеров того, как настраивать эти элементы пользовательского интерфейса веб-чата.

Полный список всех параметров, которые можно легко изменять в веб-чате, см. в файле

`defaultStyleOptions.js`.

Эти параметры создадут *набор стилей* — набор правил CSS, дополненный `glamor`. Вы можете найти полный список стилей CSS, созданных в наборе стилей в `createStyleSet.js` файле.

## Установка размера контейнера веб-чата

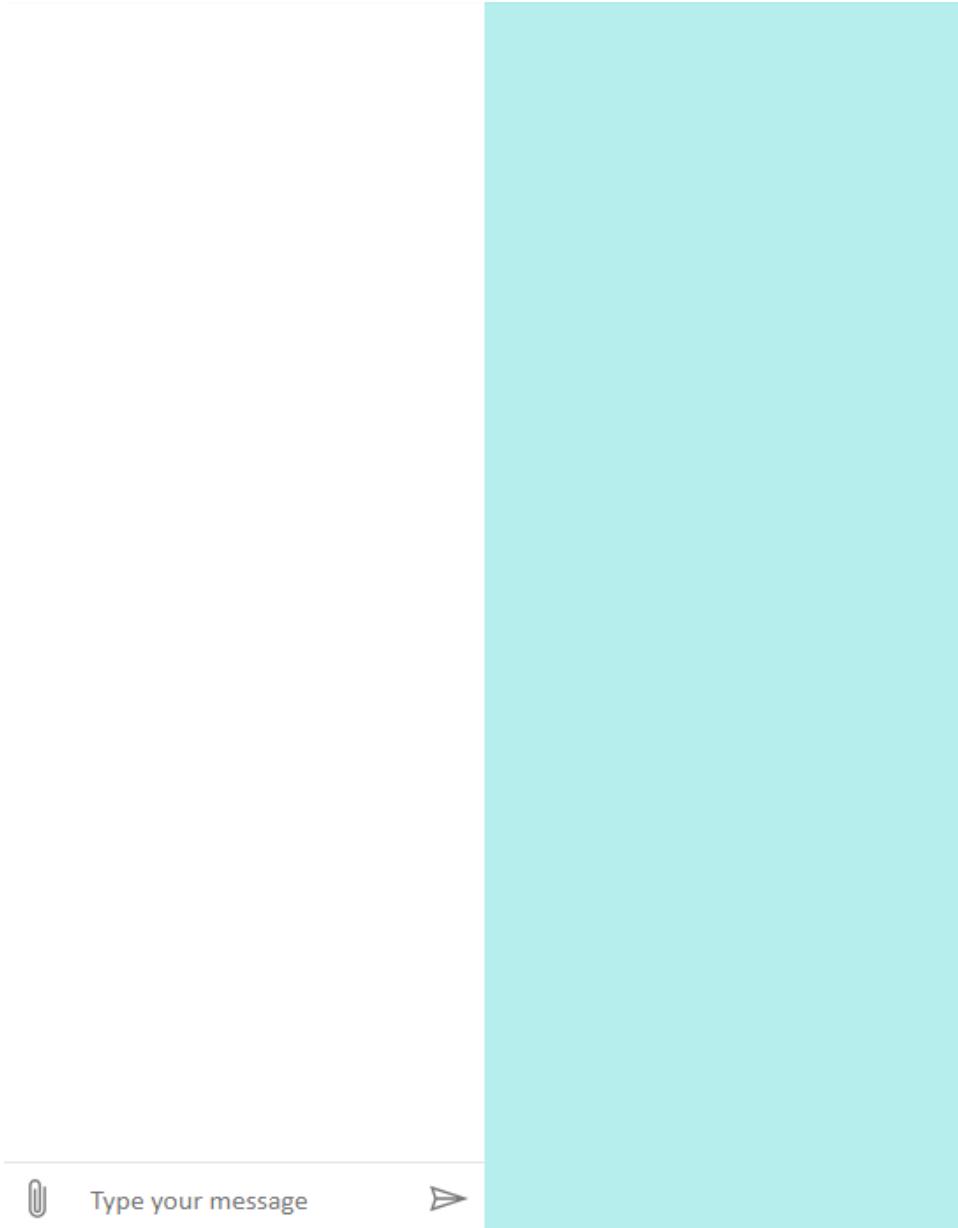
Теперь можно настроить размер контейнера веб-чата с помощью `styleSetOptions`. В следующем примере есть цвет фона `body` — `paleturquoise` — для отображения контейнера веб-чата (раздел на белом фоне).

```
...
<head>
  <style>
    html, body { height: 100% }
    body {
      margin: 0;
      background-color: paleturquoise;
    }

    #webchat {
      height: 100%;
      width: 100%;
    }
  </style>
</head>
<body>
  <div id="webchat" role="main"></div>
  <script>
    (async function () {
      window.WebChat.renderWebChat({
        directLine: window.WebChat.createDirectLine({ token }),
        styleOptions: {
          rootHeight: '100%',
          rootWidth: '50%'
        }
      }, document.getElementById('webchat'));
    })()
  </script>
...

```

Результат:



## Изменение шрифта или цвета

Вместо использования цвета фона по умолчанию и шрифтов, используемых внутри пузырьков чата, вы можете настроить их в соответствии со стилем целевой веб-страницы. Приведенный ниже фрагмент кода позволяет изменить цвет фона сообщений от пользователя и от бота.

The screenshot shows a Microsoft Teams-style chat interface. A message from a bot named "Clippy" is displayed in a light green box with the text "What can you do?". Below it, a message from a user is shown in a light purple box with the text "I can do many things, but don't call me Clippy.". Both messages have a timestamp "A minute ago" below them.

Если вам нужно задать простые параметры стиля, сделайте это с помощью `styleOptions`. В параметрах стиля приводится набор предварительно определенных стилей, которые можно изменять напрямую. На их основе веб-чат будет вычислять всю таблицу стилей.

```

<!DOCTYPE html>
<html>
  <body>
    <div id="webchat" role="main"></div>
    <script src="https://cdn.botframework.com/botframework-webchat/latest/webchat.js"></script>
    <script>
      const styleOptions = {
        bubbleBackground: 'rgba(0, 0, 255, .1)',
        bubbleFromUserBackground: 'rgba(0, 255, 0, .1)'
      };

      window.WebChat.renderWebChat(
        {
          directLine: window.WebChat.createDirectLine({
            secret: 'YOUR_BOT_SECRET'
          }),

          // Passing 'styleOptions' when rendering Web Chat
          styleOptions
        },
        document.getElementById('webchat')
      );
    </script>
  </body>
</html>

```

## Изменение CSS вручную

Помимо цвета вы можете изменить шрифты, используемые для отображения сообщений:

*Can you tell me about your siblings?*

4 minutes ago

*Do you know one of my siblings is called  
"Microsoft Comic Chat"? We are buddy.*

4 minutes ago

Для более глубокой настройки стилей также можно изменить стиль, заданный вручную, напрямую задав правила CSS.

Так как правила CSS тесно связаны со структурой дерева DOM, вероятно, эти правила необходимо обновить для работы с более новой версией веб-чата.

```

<!DOCTYPE html>
<html>
  <body>
    <div id="webchat" role="main"></div>
    <script src="https://cdn.botframework.com/botframework-webchat/latest/webchat.js"></script>
    <script>
      // "styleSet" is a set of CSS rules which are generated from "styleOptions"
      const styleSet = window.WebChat.createStyleSet({
        bubbleBackground: 'rgba(0, 0, 255, .1)',
        bubbleFromUserBackground: 'rgba(0, 255, 0, .1)'
      });

      // After generated, you can modify the CSS rules
      styleSet.textContent = {
        ...styleSet.textContent,
        fontFamily: "'Comic Sans MS', 'Arial', sans-serif",
        fontWeight: 'bold'
      };

      window.WebChat.renderWebChat(
        {
          directLine: window.WebChat.createDirectLine({
            secret: 'YOUR_BOT_SECRET'
          }),

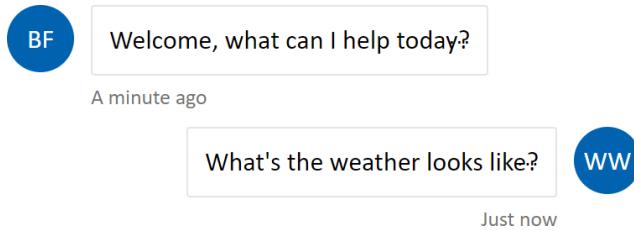
          // Passing 'styleSet' when rendering Web Chat
          styleSet
        },
        document.getElementById('webchat')
      );
    </script>
  </body>
</html>

```

## Изменение аватара бота в диалоговом окне

В последней версии Web Chat поддерживаются аватары, которые можно настроить с помощью

`botAvatarInitials` И `userAvatarInitials` В СВОЙСТВЕ `styleOptions`.



```

<!DOCTYPE html>
<html>
  <body>
    <div id="webchat" role="main"></div>
    <script src="https://cdn.botframework.com/botframework-webchat/latest/webchat.js"></script>
    <script>
      const styleOptions = {
        botAvatarInitials: 'BF',
        userAvatarInitials: 'WC'
      };

      window.WebChat.renderWebChat(
        {
          directLine: window.WebChat.createDirectLine({
            secret: 'YOUR_BOT_SECRET'
          }),
          styleOptions
        },
        document.getElementById('webchat')
      );
    </script>
  </body>
</html>

```

В свойство `styleOptions` Web Chat мы добавили `botAvatarInitials` И `userAvatarInitials`.

```

botAvatarInitials: 'BF',
userAvatarInitials: 'WC'

```

`botAvatarInitials` задаст текст внутри аватара в левой части окна. Если присвоено ложное значение, аватар на стороне бота будет скрыт. Напротив, `userAvatarInitials` задаст текст аватара в правой части окна.

## Пользовательские действия или вложения преобразования для просмотра

С помощью последней версии веб-чата можно также преобразовать для просмотра действия или вложения, которые веб-чат не поддерживает. Преобразование для просмотра действий и вложений отправляется через настраиваемый конвейер, смоделированный после ПО промежуточного слоя [Redux](#). Конвейер достаточно гибкий, чтобы вы могли выполнить следующие задачи:

- Дополнить имеющиеся действия или вложения.
- Добавить новые действия и вложения.
- Заменить имеющиеся действия и вложения (или удалить их).
- Поместить ПО промежуточного слоя в цепочку репликации.

### Отображение репозитория GitHub как вложения

Если вы хотите отобразить колоду карт репозитория GitHub, можно создать компонент React для репозитория GitHub и добавить его в качестве ПО промежуточного слоя для вложения.

Show me related repositories.

A minute ago

Here are the list of GitHub repository:



[Microsoft/  
BotFramework-WebChat](#)



[Mic  
BotFrame](#)

A minute ago

```
import ReactWebChat from 'botframework-webchat';
import ReactDOM from 'react-dom';

// Create a new React component that accept render a GitHub repository attachment
const GitHubRepositoryAttachment = props => (
  <div
    style={{
      fontFamily: "'Calibri', 'Helvetica Neue', Arial, sans-serif",
      margin: 20,
      textAlign: 'center'
    }}
  >
    <svg
      height="64"
      viewBox="0 0 16 16"
      version="1.1"
      width="64"
      aria-hidden="true"
    >
      <path
        fillRule="evenodd"
        d="M8 0C3.58 0 0 3.58 0 8c0 3.54 2.29 6.53 5.47 7.59.4.07.55-.17.55-.38 0-.19-.01-.82-.01-1.49-2.01.37-2.53-.49-2.69-.94-.09-.23-.48-.94-.82-1.13-.28-.15-.68-.52-.01-.53.63-.01 1.08.58 1.23.82.72 1.21 1.87.87 2.33.66.07-.52.28-.87.51-1.07-1.78-.2-3.64-.89-3.64-3.95 0-.87.31-1.59.82-2.15-.08-.2-.36-1.02.08-2.12 0 0 .67-.21 2.2.82.64-.18 1.32-.27 2-.27.68 0 1.36.09 2 .27 1.53-1.04 2.2-.82 2.2-.82.44 1.1.16 1.92.08 2.12.51.56.82 1.27.82 2.15 0 3.07-1.87 3.75-3.65 3.95.29.25.54.73.54 1.48 0 1.07-.01 1.93-.01 2.2 0 .21.15.46.55.38A8.013 8.013 0 0 0 16 8c0-4.42-3.58-8-8-8z"
      />
    </svg>
    <p>
      <a
        href={`https://github.com/${encodeURI(props.owner)}/${encodeURI(
          props.repo
        )}`}
        target="_blank"
      >
        {props.owner}<br />
        {props.repo}
      </a>
    </p>
  </div>
);

// Creating a new middleware pipeline that will render <GitHubRepositoryAttachment> for specific type of attachment
const attachmentMiddleware = () => next => card => {
  switch (card.attachment.contentType) {
    case 'application/vnd.microsoft.botframework.samples.github-repository':
      return (
        <GitHubRepositoryAttachment
```

```

        owner={card.attachment.content.owner}
        repo={card.attachment.content.repo}
    />
);

default:
    return next(card);
}

};

ReactDOM.render(
    <ReactWebChat
        // Prepending the new middleware pipeline
        attachmentMiddleware={attachmentMiddleware}
        directLine={window.WebChat.createDirectLine({ token })}>,
    document.getElementById('webchat')
);

```

Полный пример можно найти в [примере customization-card-components](#).

В этом примере мы добавляем новый компонент React, называемый `GitHubRepositoryAttachment`:

```

const GitHubRepositoryAttachment = props => (
    <div
        style={{
            fontFamily: "'Calibri', 'Helvetica Neue', Arial, sans-serif",
            margin: 20,
            textAlign: 'center'
        }}
    >
        <svg
            height="64"
            viewBox="0 0 16 16"
            version="1.1"
            width="64"
            aria-hidden="true"
        >
            <path
                fillRule="evenodd"
                d="M8 0C3.58 0 0 3.58 0 8c0 3.54 2.29 6.53 5.47 7.59 4.07 5.55-1.17 5.55-3.38 0-.19-.01-.82-.01-1.49-2.01.37-2.53-.49-2.69-.94-.09-.23-.48-.94-.82-1.13-.28-.15-.68-.52-.01-.53.63-.01 1.08.58 1.23.82.72 1.211.87.87 2.33.66.07-.52.28-.87.51-1.07-1.78-.2-3.64-.89-3.64-3.95 0-.87.31-1.59.82-2.15-.08-.2-.36-1.02.08-2.12 0 0 .67-.21 2.2.82.64-.18 1.32-.27 2-.27.68 0 1.36.09 2 .27 1.53-1.04 2.2-.82 2.2-.82.44 1.1.16 1.92.082.12.51.56.82 1.27.82 2.15 0 3.07-1.87 3.75-3.65 3.95.29.25.54.73.54 1.48 0 1.07-.01 1.93-.01 2.2 0.21.15.46.55.38A8.013 0 0 0 16 8c0-4.42-3.58-8-8-8z"
            />
        </svg>
        <p>
            <a
                href={`https://github.com/${encodeURI(props.owner)}/${encodeURI(
                    props.repo
                )}`}
                target="_blank"
            >
                {props.owner}/<br />
                {props.repo}
            </a>
        </p>
    </div>
);

```

Затем создаем ПО промежуточного слоя, которое преобразует для просмотра новый компонент React, когда бот отправляет вложение с типом содержимого

`application/vnd.microsoft.botframework.samples.github-repository`. В противном случае он будет по-

прежнему работать на ПО промежуточного слоя путем вызова `next(card)`.

```
const attachmentMiddleware = () => next => card => {
  switch (card.attachment.contentType) {
    case 'application/vnd.microsoft.botframework.samples.github-repository':
      return (
        <GitHubRepositoryAttachment
          owner={card.attachment.content.owner}
          repo={card.attachment.content.repo}
        />
      );
    default:
      return next(card);
  }
};
```

Действие, отправленное из бота, выглядит следующим образом:

```
{
  "type": "message",
  "from": {
    "role": "bot"
  },
  "attachmentLayout": "carousel",
  "attachments": [
    {
      "contentType": "application/vnd.microsoft.botframework.samples.github-repository",
      "content": {
        "owner": "Microsoft",
        "repo": "BotFramework-WebChat"
      }
    },
    {
      "contentType": "application/vnd.microsoft.botframework.samples.github-repository",
      "content": {
        "owner": "Microsoft",
        "repo": "BotFramework-Emulator"
      }
    },
    {
      "contentType": "application/vnd.microsoft.botframework.samples.github-repository",
      "content": {
        "owner": "Microsoft",
        "repo": "BotFramework-DirectLineJS"
      }
    }
  ]
}
```

# Добавление функции единого входа в Web Chat

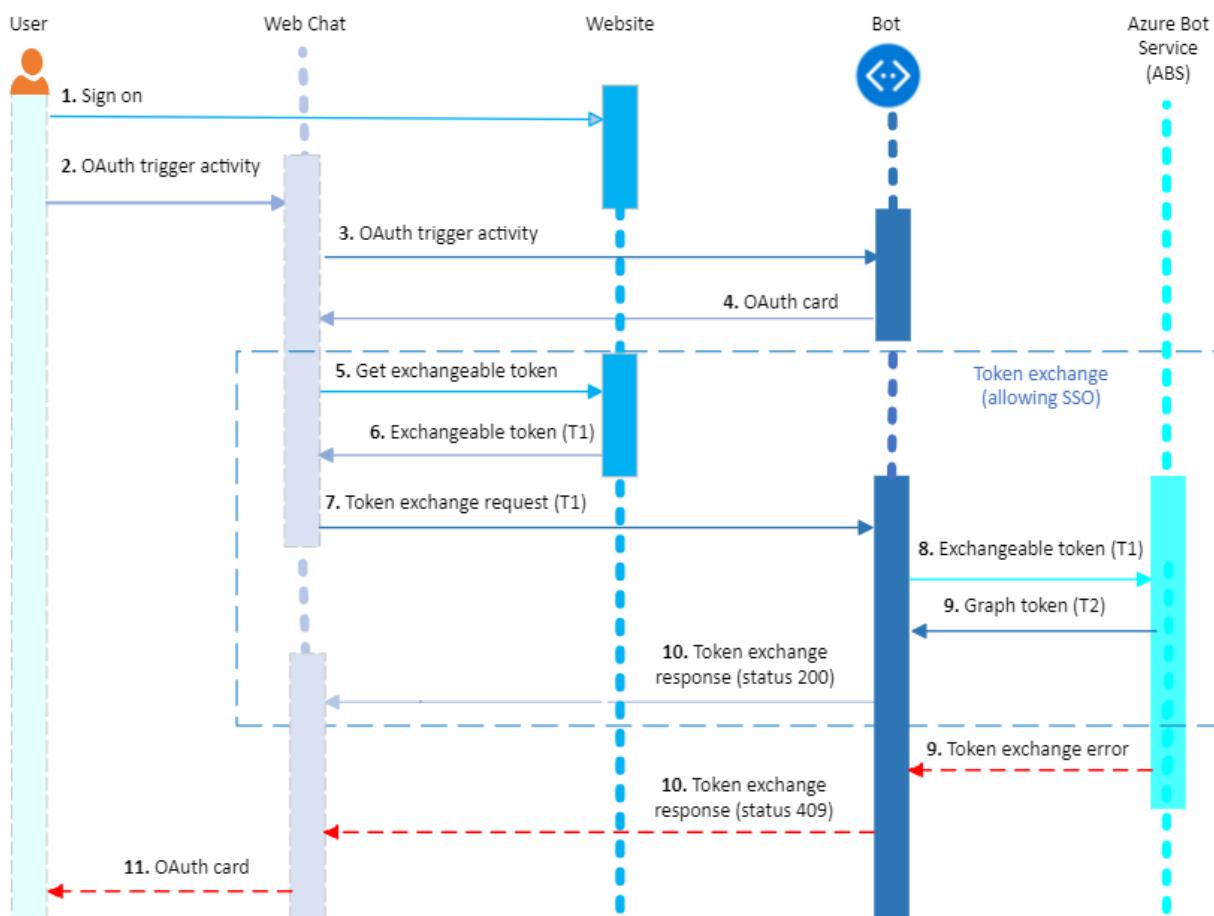
27.10.2020 • 3 minutes to read • [Edit Online](#)

**применимо к:** Пакет SDK v4

Единый вход (SSO) позволяет клиенту, например элементу управления для разговора, взаимодействовать с программой-роботом от имени пользователя. Сейчас поддерживается только поставщик удостоверений [Azure AD версии 2](#).

Как правило, веб-чат внедряется на страницу веб-сайта. При входе пользователя на веб-сайт веб-чат вызывает робота от имени пользователя. Маркер клиента веб-сайта, основанный на учетных данных пользователя, передается на другой сервер для доступа к Bot. Таким образом, пользователю не нужно выполнять вход дважды: в первый раз на веб-сайте, а второй раз на Bot — термин единий вход.

На следующей схеме показан поток единого входа при использовании клиента веб-чата.



В случае сбоя SSO возвращается к существующему поведению при отображении карты OAuth . Ошибка может быть связана, например, с необходимостью получить согласие пользователя или со сбоем при обмене токенами.

Давайте проанализируем этот поток.

1. Вход пользователя на веб-сайт.
2. Действие триггера OAuth получается через Интернет-чат.
3. Веб-чат запускает диалог с Bot через действие триггера OAuth.

4. Bot отправляет карточку OAuth в Интернет-чат.
5. Веб-чат перехватывает карточку OAuth перед ее отображением пользователю и проверяет, содержит ли она `TokenExchangeResource` свойство.
6. Если свойство существует, веб-чат должен получить для пользователя маркер с обменом данными, который должен быть токеном Azure AD v2.
7. Веб-чат отправляет действие `Invoke` на робот с текстом, показанным ниже.

```
{
  "type": "Invoke",
  "name": "signin/tokenExchange",
  "value": {
    "id": "<any unique Id>",
    "connectionName": "<connection name on the bot (from the OAuth Card)>",
    "token": "<exchangeable token>"
  }
}
```

8. Программа-робот обрабатывает, `TokenExchangeInvokeRequest` выполняя запрос к службе Azure Bot для получения токена, поддающегося Exchange.
9. Служба Azure Bot отправляет маркер в Bot.
10. Bot возвращает обратно в `TokenExchangeInvokeResponse` веб-чат. Веб-чат ждет, пока он получит `TokenExchangeInvokeResponse`.

```
{
  "status": "<response code>",
  "body": {
    "id": "<unique Id>",
    "connectionName": "<connection Name on the bot (from the OAuth Card)>",
    "failureDetail": "<failure reason if status code is not 200, null otherwise>"
  }
}
```

11. Если параметр `TokenExchangeInvokeResponse` имеет значение `status 200`, то в веб-чате не отображается карта OAuth. Для любого другого `status` или, если `TokenExchangeInvokeResponse` не получен, веб-чат отображает карточку OAuth для пользователя. Это гарантирует, что поток единого входа применит обычный режим карты OAuth при любых ошибках или несоответствии зависимостей, таких как согласие пользователя.

Пример реализации см. в этом [примере единого входа](#).