Telemetry logging was added to version 4.2 of the Bot Framework SDK. This enables bot applications to send event data to telemetry services such as Application Insights. Telemetry offers insights into your bot by showing which features are used the most, detects unwanted behavior and offers visibility into availability, performance, and usage.

Note: In version 4.6, the standard method for implementing telemetry into a bot has been updated in order to ensure telemetry is logged correctly when using a custom adapter. This article has been updated to show the updated method. The changes are backwards compatible and bots using the previous method will continue to log telemetry correctly.

In this article you will learn how to implement telemetry into your bot using Application Insights. It will cover:

- The code required to wire up telemetry in your bot and connect to Application Insights.
- Enabling telemetry in your bot's Dialogs.
- Enabling telemetry to capture usage data from other services like LUIS and QnA Maker.
- Visualizing your telemetry data in Application Insights.

Prerequisites

C# JavaScript

- The CoreBot sample code
- The Application Insights sample code
- A subscription to Microsoft Azure
- An Application Insights key
- Familiarity with Application Insights
- git

> **Note**
>
> The **Application Insights sample code** was built on top of the **CoreBot sample code**. This article will step you through modifying the CoreBot sample code to incorporate telemetry. If you are following along in Visual Studio you will have the Application Insights sample code by the time you are finished.

Wiring up telemetry in your bot

C# JavaScript

This article starts from the CoreBot sample app and adds the code required to integrate telemetry into any bot. This will enable Application Insights to begin tracking requests.

> **Important**
>
> If you have not setup your **Application Insights** account and created your **Application Insights key**, do that before proceeding.

1. Open the CoreBot sample app in Visual Studio.
2. Add the `Microsoft.Bot.Builder.Integration.ApplicationInsights.Core` NuGet package. For more information on using NuGet, see Install and manage packages in Visual Studio:
3. Include the following statements in `Startup.cs`:

C# Copy

```
using Microsoft.ApplicationInsights.Extensibility;
using Microsoft.Bot.Builder.ApplicationInsights;
using Microsoft.Bot.Builder.Integration.ApplicationInsights.Core;
using Microsoft.Bot.Builder.Integration.AspNet.Core;
```

Note: If you're following along by updating the CoreBot sample code you will notice that the using statement for `Microsoft.Bot.Builder.Integration.AspNet.Core` already exists in the CoreBot sample.

4. Include the following code in the `ConfigureServices()` method in `Startup.cs`. This will make telemetry services available to your bot via dependency injection (DI):

C# Copy

```
// 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>();
    ...
}
```

Note: If you're following along by updating the CoreBot sample code you will notice that `services.AddSingleton<IBotFrameworkHttpAdapter, AdapterWithErrorHandler>();` already exists

5. Instruct the adapter to use the middleware code that was added to the `ConfigureServices()` method. You do this in `AdapterWithErrorHandler.cs` with the parameter TelemetryInitializerMiddleware telemetryInitializerMiddleware in the constructor's parameter list, and the `Use(telemetryInitializerMiddleware);` statement in the contructor as shown here:

C# Copy

```
    public AdapterWithErrorHandler(IConfiguration configuration,
ILogger<BotFrameworkHttpAdapter> logger,
TelemetryInitializerMiddleware telemetryInitializerMiddleware,
ConversationState conversationState = null)
        : base(configuration, logger)
{
    ...
    Use(telemetryInitializerMiddleware);
}
```

6. You will also need to add `Microsoft.Bot.Builder.Integration.ApplicationInsights.Core` to your list of using statements in `AdapterWithErrorHandler.cs`.

7. Add the Application Insights instrumentation key in your `appsettings.json` file. The `appsettings.json` file contains metadata about external services the bot uses while running. For example, CosmosDB, Application Insights and the Language Understanding (LUIS) service connection and metadata is stored there. The addition to your `appsettings.json` file must be in this format:

JSON Copy

```
{
    "MicrosoftAppId": "",
    "MicrosoftAppPassword": "",
    "ApplicationInsights": {
        "InstrumentationKey": "xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx"

    }
}
```

Note: Details on getting the *Application Insights instrumentation key* can be found in the article Application Insights keys.

At this point the preliminary work to enable telemetry using Application Insights is done. You can run your bot locally using the bot emulator and then go into Application Insights to see what is being logged, such as response time, overall app health, and general running information.

Enabling telemetry in your bots Dialogs

When adding a new dialog to any ComponentDialog, it will inherit the Microsoft.Bot.Builder.IBotTelemetryClient of its parent dialog. For example, In the CoreBot sample application all dialogs are added to the MainDialog which is a

CoreBot sample application all dialogs are added to the MainDialog which is a ComponentDialog. Once you set the TelemetryClient property to the MainDialog all dialogs added to it will automatically inherit the telemetryClient from it, so it does not need to be explicitly set when adding dialogs.

Follow the steps below to update your CoreBot example:

1. In `MainDialog.cs`, update the constructor's parameter list to include the `IBotTelemetryClient` parameter, then set the MainDialog's TelemetryClient property to that value as shown in the following code snippet:

```
C#                                                    Copy

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**
>
> If you are following along and updating the CoreBot sample code, you can refer to the **Application Insights sample code** if you run into any problems.

That's all there is to adding telemetry to your bots dialogs, at this point if you ran your bot you should see things being logged in Application Insights, however if you have any integrated technology such as LUIS and QnA Maker you will need to add the `TelemetryClient` to that code as well.

Enabling / disabling activity event and personal information logging

C# JavaScript

Enabling or disabling Activity logging

By default, the `TelemetryInitializerMiddleware` will use the

`TelemetryLoggerMiddleware` to log telemetry when your bot sends / receives activities. Activity logging creates custom event logs in your Application Insights resource. If you wish, you can disable activity event logging by setting `logActivityTelemetry` to false on the `TelemetryInitializerMiddleware` when registering it in **Startup.cs**.

C# Copy

```
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);
            });
    ...
}
```

Enable or disable logging personal information

By default, if activity logging is enabled, some properties on the incoming / outgoing activities are excluded from logging as they are likely to contain personal information, such as user name and the activity text. You can choose to include these properties in your logging by making the following change to **Startup.cs** when registering the `TelemetryLoggerMiddleware`.

C# Copy

```
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);
            });
    ...
}
```

Next we will see what needs to be included to add telemetry functionality to the dialogs. This will enable you to get additional information such as what dialogs run, and statistics about each one.

Enabling telemetry to capture usage data from other services like LUIS and QnA Maker

C# JavaScript

We will next implement telemetry functionality in your LUIS service. The LUIS service has built-in telemetry logging available so there is very little you need to do to start getting telemetry data from LUIS. If you are interested in enabling telemetry in a QnA Maker enabled bot, see Add telemetry to your QnAMaker bot

1. The *`IBotTelemetryClient telemetryClient`* parameter is required in the `FlightBookingRecognizer` constructor in `FlightBookingRecognizer.cs`:

C# Copy

```
public FlightBookingRecognizer(IConfiguration configuration,
IBotTelemetryClient telemetryClient)
```

2. Next you will need to enable the `telemetryClient` when creating your `LuisRecognizer` in the `FlightBookingRecognizer` constructor. You do this by adding the `telemetryClient` as a new *LuisRecognizerOption*:

C# Copy

```
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);
}
```

That's it, you should have a functional bot that logs telemetry data into Application insights. You can use the Bot Framework Emulator to run your bot locally. You shouldn't see any changes in the bot's behavior, but it will be logging information into Application Insights. Interact with the bot by sending multiple messages and in the next section we will review the telemetry results in Application Insights.

For information on testing and debugging your bot, you can refer to the following articles:

- Debug a bot
- Testing and debugging guidelines
- Debug with the emulator

Visualizing your telemetry data in Application Insights

Application Insights monitors the availability, performance, and usage of your bot application whether it's hosted in the cloud or on-premises. It leverages the powerful

data analysis platform in Azure Monitor to provide you with deep insights into your application's operations and diagnose errors without waiting for a user to report them. There are a few ways to see the telemetry data collected by Application Insights, two of the primary ways are through queries and the dashboard.

Querying your telemetry data in Application Insights using Kusto Queries

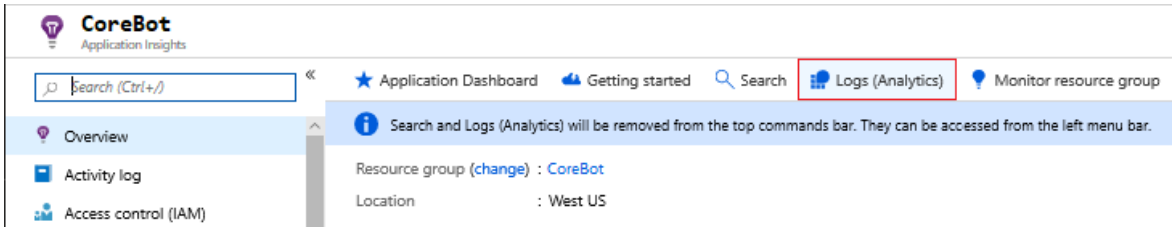
Use this section as a starting point to learn how to use log queries in Application Insights. It demonstrates two useful queries and provides links to other documentation with additional information.

To query your data

1. Go to the Azure portal

2. Navigate to your Application Insights. Easiest way to do so is click on **Monitor > Applications** and find it there.

3. Once in your Application Insights, you can click on *Logs (Analytics)* on the navigation bar.

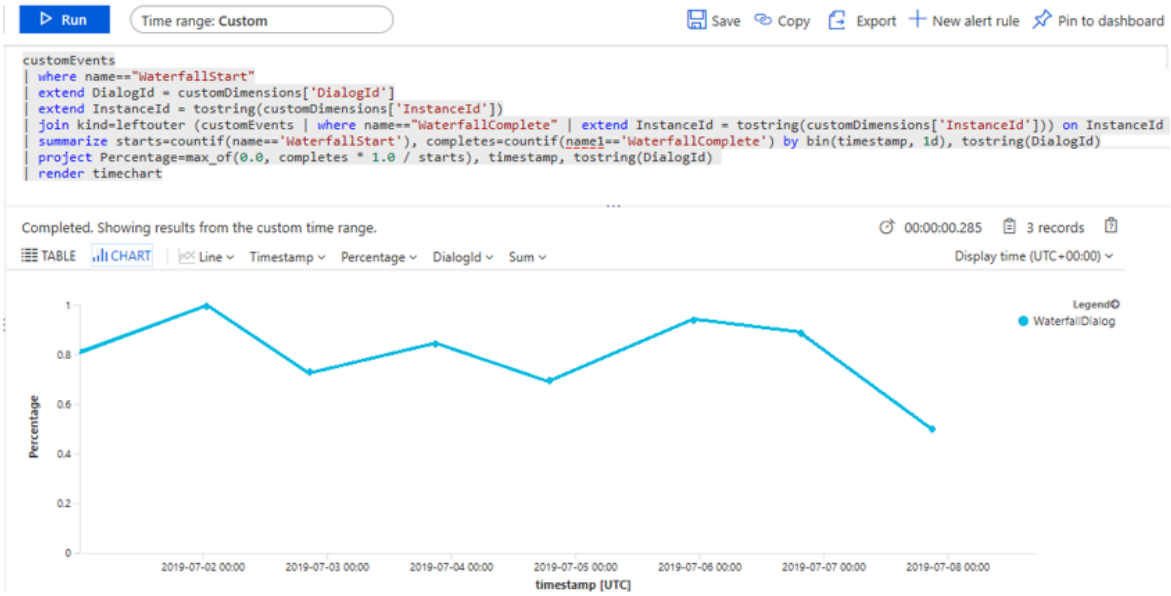


4. This will bring up the Query window. Enter the following query and select *Run*:

SQL Copy

```
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. This will return the percentage of waterfall dialogs that run to completion.

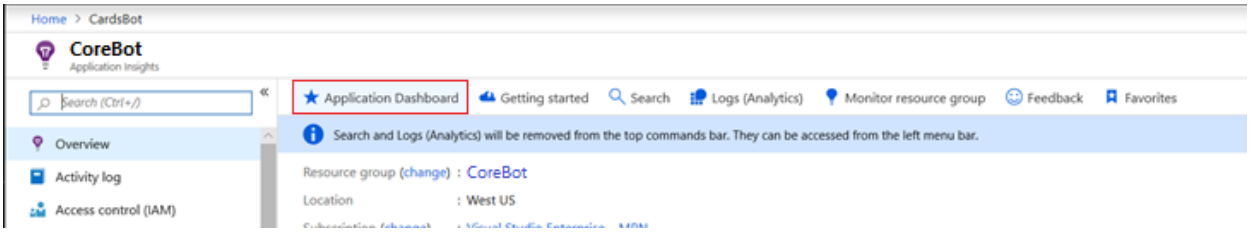


Tip

You can pin any query to your Application Insights dashboard by selecting the button on the top right of the **Logs (Analytics)** blade. Just select the dashboard you want it pinned to, and it will be available next time you visit that dashboard.

The Application Insights dashboard

Anytime you create an Application Insights resource in Azure, a new dashboard will automatically be created and associated with it. You can see that dashboard by selecting the button at the top of your Application Insights blade, labeled **Application Dashboard**.



Alternatively, to view the data, go to the Azure portal. Click **Dashboard** on the left, then select the dashboard you want from the drop-down.

There, you'll see some default information about your bot performance and any additional queries that you've pinned to your dashboard.

Additional Information

- Add telemetry to your QnAMaker bot
- What is Application Insights?
- Using Search in Application Insights
- Create custom KPI dashboards using Azure Application Insights

Is this page helpful?

Yes No