100 XP

Instrument the application

7 minutes

The `TelemetryClient` object is used to record information about application-specific events and metrics.

In the video app example, stakeholders want information about how users are experiencing videos on the site. They'd like data about the number of videos started and never finished and the number of videos added to the site every day. This data is specific to the application and isn't included in the telemetry provided by Application Insights. You can add it by using the functionality provided in the SDK.

In this unit, you'll learn about events and metrics and about how to write code to track them in your apps.

Events and metrics

Events and *metrics* are the two primary types of custom data that can be tracked by Application Insights.

Events represent any kind of event that occurs in your application that you're interested in tracking. For the video sharing application, you might want to track an event each time a video is added to or deleted from the site, or every time a comment is made on a video. Each kind of event that occurs in your application needs a unique name. You might have events named `VideoAdded`, `VideoDeleted`, and `CommentAdded`. Tracked events can have arbitrary data attached to them that can be used to filter and group the data. For example, you might want to track the length of each video or comment added. You can find every tracked event with all its data in the Diagnostic Search view in the Azure portal. You can create aggregate visualizations in Metrics Explorer.

Metrics are numeric measurements taken independently of any event, typically on an interval. For example, you might want to track the number of videos being viewed simultaneously every 10 seconds. Like events, each metric must have a unique name.

Metrics are *pre-aggregated*. Application Insights stores and transmits a statistical summary of measurements taken over time, not the exact value of each measurement. This technique greatly reduces the amount of data sent to Application Insights, reducing costs, and makes metric data available for viewing and querying more quickly than if each individual measurement were tracked.

Metrics can also be multidimensional. Multidimensional metrics record multiple values into a single metric. These properties can be used to segment, filter, and group the data in visualizations. Depending on your use case, using a multidimensional metric instead of multiple individual metrics can make it easier to explore your telemetry data and find meaning in it. For example, suppose your user interface includes **Like** and **Love** buttons that users can select to rate videos. If you wanted to track the total number of Likes and Loves in the system, instead of tracking two distinct metrics, you could create a single metric called TotalUserResponses. This single metric could include a dimension called Type and use it to track the numbers of both Likes and Loves. These values could then be aggregated together or displayed separately on one chart.

Add instrumentation to your code

Get a TelemetryClient object

The `TelemetryClient` class provides access to all the properties and methods that you use to send information to Application Insights.

When you initialize the SDK with `UseApplicationInsights()`, `TelemetryClient` is registered for dependency injection. This means you can efficiently access `TelemetryClient` by adding it to the constructor of any controller or other component where you want to use it. For example, if you want to add custom telemetry to the `HomeController` for your MVC web app, modify the constructor to accept a `TelemetryClient` parameter and store the value for use in the controller's methods:

C# Copy

```
public class HomeController : Controller
{
    private TelemetryClient aiClient;

    public HomeController(TelemetryClient aiClient)
    {
        this.aiClient = aiClient;
    }
}
```

In this example, `aiClient` can be used in your controller methods to track events and metrics.

Tracking events

To track an event that occurs within your application, call `TrackEvent` on a `TelemetryClient` with the name of the event. `TrackEvent` supports two optional `IDictionary` arguments for tracking named properties and numeric metrics related to the event.

C# Copy

```
// Track an event
this.aiClient.TrackEvent("CommentSubmitted");

// Track an event with properties
this.aiClient.TrackEvent("VideoUploaded", new Dictionary<string, string>
{{"Category", "Sports"}, {"Format", "mp4"}});
```

Notice that the names of events and properties are ordinary strings. Events and properties can be given any names you like. It's up to you to appropriately name your events and make sure the names are used consistently throughout your code.

Tracking metrics

Metrics are recorded a little differently: call `GetMetric` with the name of the metric to get a `Metric` object, and then call `TrackValue` on it to record the measurement data. The different shape of this API reflects the underlying architecture used to pre-aggregate metrics values before sending them to the Application Insights service.

C# Copy

```
this.aiClient.GetMetric("SimultaneousPlays").TrackValue(5);
```

As with events, you can assign a metric any name you'd like.

For multidimensional metrics, use the `GetMetric` method with two or more values, and then set both values with a call to `TrackValue`:

C# Copy

```
Metric userResponse = this.aiClient.GetMetric("UserResponses", "Kind");

userResponse.TrackValue(24, "Likes");
userResponse.TrackValue(5, "Loves");
```

Next unit: Exercise - Instrument the application

Continue >