Timer trigger for Azure Functions

09/08/2018 9 minutes to read Contributors all

In this article

This article explains how to work with timer triggers in Azure Functions. A timer trigger lets you run a function on a schedule.

This is reference information for Azure Functions developers. If you're new to Azure Functions, start with the following resources:

- Create your first function
- Azure Functions developer reference
- C#, F#, Node, or Java developer reference
- Azure Functions triggers and bindings concepts

Packages - Functions 1.x

The timer trigger is provided in the Microsoft.Azure.WebJobs.Extensions NuGet package, version 2.x. Source code for the package is in the azure-webjobs-sdk-extensions GitHub repository.

Support for this binding is automatically provided in all development environments. You don't have to manually install the package or register the extension.

Packages - Functions 2.x

The timer trigger is provided in the Microsoft.Azure.WebJobs.Extensions NuGet package, version 3.x. Source code for the package is in the azure-webjobs-sdk-extensions GitHub repository.

Support for this binding is automatically provided in all development environments. You don't have to manually install the package or register the extension.

Example

See the language-specific example:

- C#
- C# script (.csx)
- F#
- JavaScript
- Java

C# example

The following example shows a C# function that runs every five minutes:

C# Copy

```
[FunctionName("TimerTriggerCSharp")]
public static void Run([TimerTrigger("0 */5 * * * *")]TimerInfo myTimer, ILogger log)
{
    if(myTimer.IsPastDue)
    {
        log.LogInformation("Timer is running late!");
    }
    log.LogInformation($"C# Timer trigger function executed at: {DateTime.Now}");
}
```

C# script example

The following example shows a timer trigger binding in a *function.json* file and a C# script function that uses the binding. The function writes a log indicating whether this function invocation is due to a missed schedule occurrence.

Here's the binding data in the *function.json* file:

JSON Copy

```
{
    "schedule": "0 */5 * * * *",
    "name": "myTimer",
    "type": "timerTrigger",
    "direction": "in"
}
```

Here's the C# script code:

C# Copy

```
public static void Run(TimerInfo myTimer, ILogger log)
{
    if(myTimer.IsPastDue)
    {
        log.LogInformation("Timer is running late!");
    }
    log.LogInformation($"C# Timer trigger function executed at: {DateTime.Now}" );
}
```

F# example

The following example shows a timer trigger binding in a *function.json* file and an F# script function that uses the binding. The function writes a log indicating whether this function invocation is due to a missed schedule occurrence.

Here's the binding data in the *function.json* file:

JSON Copy

```
{
    "schedule": "0 */5 * * * *",
    "name": "myTimer",
    "type": "timerTrigger",
    "direction": "in"
}
```

Here's the F# script code:

F# Copy

```
let Run(myTimer: TimerInfo, log: ILogger ) =
    if (myTimer.IsPastDue) then
        log.LogInformation("F# function is running late.")
    let now = DateTime.Now.ToLongTimeString()
    log.LogInformation(sprintf "F# function executed at %s!" now)
```

JavaScript example

The following example shows a timer trigger binding in a *function.json* file and a JavaScript function that uses the binding. The function writes a log indicating whether this function invocation is due to a missed schedule occurrence.

Here's the binding data in the *function.json* file:

JSON Copy

```
{
    "schedule": "0 */5 * * * *",
    "name": "myTimer",
    "type": "timerTrigger",
    "direction": "in"
}
```

Here's the JavaScript code:

JavaScript Copy

```
module.exports = function (context, myTimer) {
    var timeStamp = new Date().toISOString();

    if(myTimer.isPastDue)
    {
        context.log('Node is running late!');
    }
    context.log('Node timer trigger function ran!', timeStamp);

    context.done();
};
```

Java example

The following example function triggers and executes every five minutes. The `@TimerTrigger` annotation on the function defines the schedule using the same string format as CRON expressions.

Java Copy

```
@FunctionName("keepAlive")
public void keepAlive(
  @TimerTrigger(name = "keepAliveTrigger", schedule = "0 *&#47;5 * * * *") String
timerInfo,
      ExecutionContext context
 ) {
     // timeInfo is a JSON string, you can deserialize it to an object using your favorite
JSON library
     context.getLogger().info("Timer is triggered: " + timerInfo);
}
```

Attributes

In C# class libraries, use the TimerTriggerAttribute.

The attribute's constructor takes a CRON expression or a `TimeSpan`. You can use `TimeSpan` only if the function app is running on an App Service plan. The following example shows a CRON expression:

C# Copy

```
[FunctionName("TimerTriggerCSharp")]
public static void Run([TimerTrigger("0 */5 * * * *")]TimerInfo myTimer, ILogger log)
{
    if (myTimer.IsPastDue)
    {
        log.LogInformation("Timer is running late!");
    }
    log.LogInformation($"C# Timer trigger function executed at: {DateTime.Now}");
}
```

Configuration

The following table explains the binding configuration properties that you set in the *function.json* file and the `TimerTrigger` attribute.

function.json property	Attribute property	Description
type	n/a	Must be set to "timerTrigger". This property is set automatically when you create the trigger in the Azure portal.
direction	n/a	Must be set to "in". This property is set automatically when you create the trigger in the Azure portal.
name	n/a	The name of the variable that represents the timer object in function code.
schedule	**ScheduleExpression**	A CRON expression or a TimeSpan value. A `TimeSpan` can be used only for a function app that runs on an App Service Plan. You can put the schedule expression in an app setting and set this property to the app setting name wrapped in % signs, as in this example: "%ScheduleAppSetting%".
runOnStartup	**RunOnStartup**	If `true`, the function is invoked when the runtime starts. For example, the runtime starts when the function app wakes up after going idle due to inactivity. when the function app restarts due to function changes, and when the function app scales out. So **runOnStartup** should rarely if ever be set to `true`, especially in production.

function.json property	Attribute property	Description
useMonitor	UseMonitor	Set to `true` or `false` to indicate whether the schedule should be monitored. Schedule monitoring persists schedule occurrences to aid in ensuring the schedule is maintained correctly even when function app instances restart. If not set explicitly, the default is `true` for schedules that have a recurrence interval greater than 1 minute. For schedules that trigger more than once per minute, the default is `false`.

When you're developing locally, app settings go into the local.settings.json file.

> ⊗ Caution
>
> We recommend against setting **runOnStartup** to `true` in production. Using this setting makes code execute at highly unpredictable times. In certain production settings, these extra executions can result in significantly higher costs for apps hosted in Consumption plans. For example, with **runOnStartup** enabled the trigger in invoked whenever your function app is scaled. Make sure you fully understand the production behavior of your functions before enabling **runOnStartup** in production.

Usage

When a timer trigger function is invoked, the timer object is passed into the function. The following JSON is an example representation of the timer object.

JSON Copy

```
{
    "Schedule":{
    },
    "ScheduleStatus": {
        "Last":"2016-10-04T10:15:00+00:00",
        "LastUpdated":"2016-10-04T10:16:00+00:00",
        "Next":"2016-10-04T10:20:00+00:00"
    },
    "IsPastDue":false
}
```

The `IsPastDue` property is `true` when the current function invocation is later than scheduled. For example, a function app restart might cause an invocation to be missed.

CRON expressions

Azure Functions uses the NCronTab library to interpret CRON expressions. A CRON expression includes six fields:

```
{second} {minute} {hour} {day} {month} {day-of-week}
```

Each field can have one of the following types of values:

Type	Example	When triggered
A specific value	"0 5 * * * *"	at hh:05:00 where hh is every hour (once an hour)
All values (`*`)	"0 * 5 * * *"	at 5:mm:00 every day, where mm is every minute of the hour (60 times a day)
A range (`-` operator)	"5-7 * * * * *"	at hh:mm:05,hh:mm:06, and hh:mm:07 where hh:mm is every minute of every hour (3 times a minute)
A set of values (`,` operator)	"5,8,10 * * * * *"	at hh:mm:05,hh:mm:08, and hh:mm:10 where hh:mm is every minute of every hour (3 times a minute)
An interval value (`/` operator)	"0 */5 * * * *"	at hh:05:00, hh:10:00, hh:15:00, and so on through hh:55:00 where hh is every hour (12 times an hour)

To specify months or days you can use numeric values, names, or abbreviations of names:

- For days, the numeric values are 0 to 6 where 0 starts with Sunday.
- Names are in English. For example: `Monday` , `January` .
- Names are case-insensitive.
- Names can be abbreviated. Three letters is the recommended abbreviation length. For example: `Mon` , `Jan` .

CRON examples

Here are some examples of CRON expressions you can use for the timer trigger in Azure Functions.

Example	When triggered
`"0 */5 * * * *"`	once every five minutes
`"0 0 * * * *"`	once at the top of every hour
`"0 0 */2 * * *"`	once every two hours
`"0 0 9-17 * * *"`	once every hour from 9 AM to 5 PM

Example	When triggered
`"0 30 9 * * *"`	at 9:30 AM every day
`"0 30 9 * * 1-5"`	at 9:30 AM every weekday
`"0 30 9 * Jan Mon"`	at 9:30 AM every Monday in January

> **Note**
>
> You can find CRON expression examples online, but many of them omit the `{second}` field. If you copy from one of them, add the missing `{second}` field. Usually you'll want a zero in that field, not an asterisk.

CRON time zones

The numbers in a CRON expression refer to a time and date, not a time span. For example, a 5 in the `hour` field refers to 5:00 AM, not every 5 hours.

The default time zone used with the CRON expressions is Coordinated Universal Time (UTC). To have your CRON expression based on another time zone, create an app setting for your function app named `WEBSITE_TIME_ZONE`. Set the value to the name of the desired time zone as shown in the Microsoft Time Zone Index.

For example, *Eastern Standard Time* is UTC-05:00. To have your timer trigger fire at 10:00 AM EST every day, use the following CRON expression that accounts for UTC time zone:

JSON Copy

```
"schedule": "0 0 15 * * *"
```

Or create an app setting for your function app named `WEBSITE_TIME_ZONE` and set the value to **Eastern Standard Time**. Then uses the following CRON expression:

JSON Copy

```
"schedule": "0 0 10 * * *"
```

When you use `WEBSITE_TIME_ZONE`, the time is adjusted for time changes in the specific timezone, such as daylight savings time.

TimeSpan

A `TimeSpan` can be used only for a function app that runs on an App Service Plan.

Unlike a CRON expression, a `TimeSpan` value specifies the time interval between each function invocation. When a function completes after running longer than the specified interval, the timer immediately invokes the function again.

Expressed as a string, the `TimeSpan` format is `hh:mm:ss` when `hh` is less than 24. When the first two digits are 24 or greater, the format is `dd:hh:mm`. Here are some examples:

Example	When triggered
"01:00:00"	every hour
"00:01:00"	every minute
"24:00:00"	everyday

Scale-out

If a function app scales out to multiple instances, only a single instance of a timer-triggered function is run across all instances.

Function apps sharing Storage

If you share a Storage account across multiple function apps, make sure that each function app has a different `id` in *host.json*. You can omit the `id` property or manually set each function app's `id` to a different value. The timer trigger uses a storage lock to ensure that there will be only one timer instance when a function app scales out to multiple instances. If two function apps share the same `id` and each uses a timer trigger, only one timer will run.

Retry behavior

Unlike the queue trigger, the timer trigger doesn't retry after a function fails. When a function fails, it isn't called again until the next time on the schedule.

Troubleshooting

For information about what to do when the timer trigger doesn't work as expected, see Investigating and reporting issues with timer triggered functions not firing.