Rollbar.NET Notifier v.1

Documentation

Table of Contents

[Overview 2](#_Toc500332830)

[Conceptual Design of the Notifier 2](#_Toc500332831)

[The SDK Solution Layout 3](#_Toc500332832)

[Key Public Interfaces and Types of the Notifier Library 3](#_Toc500332833)

[IReconfigurable<T> 3](#_Toc500332834)

[RollbarConfig 4](#_Toc500332835)

[ErrorLevel 4](#_Toc500332836)

[ILogger 4](#_Toc500332837)

[IRollbar 5](#_Toc500332838)

[RollbarEventArgs 5](#_Toc500332839)

[RollbarFactory 6](#_Toc500332840)

[RollbarLocator 6](#_Toc500332841)

[RollbarQueueController 6](#_Toc500332842)

[Dependencies and Supported .NET Implementations 6](#_Toc500332843)

[Logging Messages Using the Notifier 7](#_Toc500332844)

[When Using the Singleton-like Instance of the Notifier 7](#_Toc500332845)

[When Using a Scoped Instance of the Notifier 7](#_Toc500332846)

[Monitoring Notifier’s Internal Events 8](#_Toc500332847)

[Integration of the Notifier into .NET Application Hosts 9](#_Toc500332848)

[Asp.Net MVC 9](#_Toc500332849)

[Winforms 10](#_Toc500332850)

[WPF 10](#_Toc500332851)

[Additional Resources 11](#_Toc500332852)

# Overview

The Rollbar.NET Notifier (or simply the Notifier) denotes an SDK library and a component compatible with latest .NET Core and Full Framework implementations. The Notifier component could be hosted by any .NET application built on any of the supported latest .NET Standard implementations.

A Notifier instance allows the hosting application to remotely log its tracing messages using Rollbar’s RESTful API. It completely abstracts out from the client code any need to know anything about the REST, specifics of Rollbar API, or Rollbar data transfer format. However, it could be very helpful to have at least some introductory understanding of these underlaying technologies.

Every explicitly specified message is sent towards Rollbar API as part of larger payload object that also includes some automatically captured metadata about the client as well as arbitrary specified key value pairs of custom data.

For the details regarding types of information that could be included as part of every logged message, please, refer to <https://rollbar.com/docs/api/items_post/>.

# Conceptual Design of the Notifier

The Notifier library support instantiation of multiple instances of the Notifier component as needed.

In general, the lifetime of an instance could be scoped, and the instance should be disposed of at the end of its scope. All the instances implement and obey the *IDisposable* interface.

However, there is one very special instance of the Notifier that does implement IDisposable interface but does not obey its purpose – it does not dispose of itself even if *Dispose()* method is called on it. This special instance has a singleton-like behavior and should be treated as an application-wide service of global scope with its lifetime matching the lifetime of the hosting application.

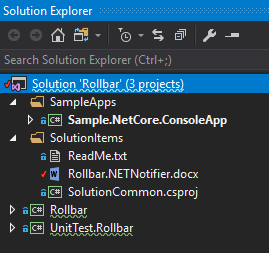
All the logging methods of the Notifier component are implemented as asynchronous fire-and-forget methods returning a reference to the component itself to support more fluent usage scenarios when performing chaining logging of messages. A payload object of every logged message is built and placed into a dedicated queue using a worker thread from the .NET managed thread pool so that the logging methods return very quickly regardless of any complexity involved into building the payload, check-ignoring, transforming or truncating the payload. Some time later, the queued payloads are picked up from the queue(s) and submitted via Rollbar RESTful API by a different dedicated worker thread while obeying assigned submission rate limits and performing submission retries as needed.

Each instance of the Notifier component can be configured independently and differently from any other instance, including even different Rollbar endpoint URL or Rollbar access token. Alternatively, more than one Notifier instance can point to the same endpoint and use the same access token. Internally, the SDK instantiates one payload queue per Notifier instance. The queues are indexed by an access token instance that is used by the corresponding Notifier instance. This way of organizing the queues by an access token allows the dedicated worker thread to process and retry the queues while complying with specified rate limits per access token. All these queues-organizing and queues-processing responsibilities performed by the internal RollbarQueueController singleton type.

The Notifier instances support on-the-fly reconfiguration.

Each Notifier has a built-in instrumentation feature that allows monitoring a hierarchy of different events related to internal operations of each instance: internal operation errors, events related to communication with the Rollbar RESTful API.

# The SDK Solution Layout



*Rollbar* project is the Rollbar.NET Notifier SDK implementation.

*UnitTest.Rollbar* is the MS Built UnitTest project implementing the unit test coverage of the SDK implementation.

*SolutionItems* solution folder contains items applicable to either all or most of the solution projects.

*SampleApps* solution folder contains all the available sample applications demonstrating various aspects of the Rollbar.NET SDK usage.

# Key Public Interfaces and Types of the Notifier Library

All the interfaces and types listed below are defined within the *Rollbar* namespace.

## IReconfigurable<T>

This interface models any generic type T that implements its own reconfiguration based on another provided instance that serves as a prototype for the new configuration of a given object of the type.

/// <summary>

/// Defines generic IReconfigurable interface.

///

/// Any type that supports its own reconfiguration based on a provided original

/// configuration should implement this interface.

/// </summary>

/// <typeparam name="T">A type that supports its reconfiguration.</typeparam>

public interface IReconfigurable<T>

{

/// <summary>

/// Reconfigures this object similar to the specified one.

/// </summary>

/// <param name="likeMe">

/// The pre-configured instance to be cloned in terms of its configuration/settings.

/// </param>

/// <returns>Reconfigured instance.</returns>

T Reconfigure(T likeMe);

/// <summary>

/// Occurs when this instance reconfigured.

/// </summary>

event EventHandler Reconfigured;

}

## RollbarConfig

This is a reconfigurable (*IReconfigurable< RollbarConfig >*) data object holding all the applicable required and optional configurational settings/properties supported by the Notifier.

## ErrorLevel

This is an enumeration of all currently supported log message error levels/categories that can be submitted to the Rollbar API:

public enum ErrorLevel

{

Critical,

Error,

Warning,

Info,

Debug,

}

## ILogger

This interfaces models all the logging methods supported by the Notifier:

public interface ILogger

{

ILogger Log(ErrorLevel level, object obj, IDictionary<string, object> custom = null);

ILogger Log(ErrorLevel level, string msg, IDictionary<string, object> custom = null);

ILogger Critical(string msg, IDictionary<string, object> custom = null);

ILogger Error(string msg, IDictionary<string, object> custom = null);

ILogger Warning(string msg, IDictionary<string, object> custom = null);

ILogger Info(string msg, IDictionary<string, object> custom = null);

ILogger Debug(string msg, IDictionary<string, object> custom = null);

ILogger Critical(Exception error, IDictionary<string, object> custom = null);

ILogger Error(Exception error, IDictionary<string, object> custom = null);

ILogger Warning(Exception error, IDictionary<string, object> custom = null);

ILogger Info(Exception error, IDictionary<string, object> custom = null);

ILogger Debug(Exception error, IDictionary<string, object> custom = null);

ILogger Critical(ITraceable traceableObj, IDictionary<string, object> custom = null);

ILogger Error(ITraceable traceableObj, IDictionary<string, object> custom = null);

ILogger Warning(ITraceable traceableObj, IDictionary<string, object> custom = null);

ILogger Info(ITraceable traceableObj, IDictionary<string, object> custom = null);

ILogger Debug(ITraceable traceableObj, IDictionary<string, object> custom = null);

ILogger Critical(object obj, IDictionary<string, object> custom = null);

ILogger Error(object obj, IDictionary<string, object> custom = null);

ILogger Warning(object obj, IDictionary<string, object> custom = null);

ILogger Info(object obj, IDictionary<string, object> custom = null);

ILogger Debug(object obj, IDictionary<string, object> custom = null);

}

## IRollbar

This interface models complete public interface of a Notifier component:

/// <summary>

/// Defines IRollbar notifier interface.

/// </summary>

/// <seealso cref="Rollbar.ILogger" />

/// <seealso cref="System.IDisposable" />

public interface IRollbar

: ILogger

, IDisposable

{

/// <summary>

/// Configures the using specified settings.

/// </summary>

/// <param name="settings">The settings.</param>

/// <returns></returns>

IRollbar Configure(RollbarConfig settings);

/// <summary>

/// Configures using the specified access token.

/// </summary>

/// <param name="accessToken">The access token.</param>

/// <returns></returns>

IRollbar Configure(string accessToken);

/// <summary>

/// Gets the configuration.

/// </summary>

/// <value>

/// The configuration.

/// </value>

RollbarConfig Config { get; }

/// <summary>

/// Gets the logger.

/// </summary>

/// <value>

/// The logger.

/// </value>

ILogger Logger { get; }

/// <summary>

/// Occurs when a Rollbar internal event happens.

/// </summary>

event EventHandler<RollbarEventArgs> InternalEvent;

}

## RollbarEventArgs

This is an abstract base for implementing concrete derived types representing different internal events happening within the Notifier.

Currently, following concrete events are supported:

InternalErrorEventArgs

RollbarApiErrorEventArgs

CommunicationErrorEventArgs

CommunicationEventArgs

All the events reported as an abstract *RollbarEventArgs* base instances. So, once you subscribe to the event - cast the received instances to the concrete type of interest.

## RollbarFactory

This is a static utility class that is used to create any scoped instance of the Notifier (which is an implementation of *IRollbar*) via its *CreateNew()* method:

/// <summary>

/// RollbarFactory utility class.

/// </summary>

public static class RollbarFactory

{

/// <summary>

/// Creates the new instance of IRollbar.

/// </summary>

/// <returns></returns>

public static IRollbar CreateNew()

{

return RollbarFactory.CreateNew(false);

}

/// <summary>

/// Creates the new instance of IRollbar.

/// </summary>

/// <param name="isSingleton">if set to <c>true</c> [is singleton].</param>

/// <returns></returns>

internal static IRollbar CreateNew(bool isSingleton)

{

return new RollbarLogger(isSingleton);

}

}

Remember, all the scoped instances of the Notifier component are IDisposable, so, use the *using(…){…}* block while dealing with short-lived instances or call *Dispose()* method manually when controlling the instance lifetime manually.

## RollbarLocator

This is a helper singleton-like type used to gain access and instantiate the special singleton-like instance of the Notifier on the first access. Call its *RollbarLocator.RollbarInstance* property to get reference to the special Notifier singleton instance.

## RollbarQueueController

This is a singleton type used to gain access to the SDK-wide internal events at runtime.

# Dependencies and Supported .NET Implementations

The library can be acquired via either

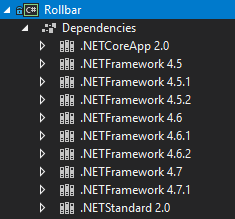
<https://www.nuget.org/packages/Rollbar/>

or

<https://github.com/rollbar/Rollbar.NET>

The library itself depends on the Newtonsoft.Json library and any supported .NET implementation.

Once the SDK’s Rollbar project is included in the hosting application solution, you can see currently supported .NET implementations within the Visual Studio Solution Explorer. For example:



It should be just enough to include the Rollbar project into the hosting application solution and referencing it properly from all the other dependent projects to be able to use the Notifier in your application.

# Logging Messages Using the Notifier

## When Using the Singleton-like Instance of the Notifier

1. Get a reference to the singleton-like instance of the Notifier by calling RollbarLocator.RollbarInstance.
2. Properly configure the instance (before any attempts to use it for logging) by calling its *Configure(…)* method while supplying valid configuration parameters.
3. Call any of the *ILogger*’s methods on the instance in order to log messages towards Rollbar API service.

For example:

const string postServerItemAccessToken = "…65fa5041749b6bf7095a190001…";

RollbarLocator.RollbarInstance

.Configure(new RollbarConfig(postServerItemAccessToken) { Environment = "proxyTest" })

;

RollbarLocator.RollbarInstance

.Info("Basic info log example.")

.Debug("First debug log.")

.Error(new NullReferenceException())

.Error(new Exception("trying out the TraceChain", new NullReferenceException()))

;

## When Using a Scoped Instance of the Notifier

1. Get reference to a newly created instance of the Notifier by calling *RollbarFactory.CreateNew()* helper method.
2. Properly configure the instance (before any attempts to use it for logging) by calling its *Configure(…)* method while supplying valid configuration parameters.
3. Call any of the *ILogger*’s methods on the instance in order to log messages towards Rollbar API service.
4. Dispose of the Notifier instance at the end of its scope by casting it to *IDisposable* and calling *Dispose()* on the cast.

For example, here the scoped instance of the Notifier is disposed of with the help of the *using(…){…}* block:

RollbarConfig loggerConfig = new RollbarConfig(RollbarUnitTestSettings.AccessToken)

{

Environment = RollbarUnitTestSettings.Environment,

};

using (var logger = RollbarFactory.CreateNew().Configure(loggerConfig))

{

logger.Log(ErrorLevel.Error, "test message");

.Info("Basic info log example.")

.Debug("First debug log.")

.Error(new NullReferenceException())

.Error(new Exception("trying out the TraceChain", new NullReferenceException()))

;

}

# Monitoring Notifier’s Internal Events

To monitor any Notifier library internal event regardless of specific instance of the Notifier:

1. Get a reference to the *RollbarQueueController.Instnace* singleton.
2. Subscribe to its *InternalEvent* event.
3. Implement the event handler the way you see it.
4. As the result of this subscription, at runtime, all the Rollbar internal events generated while using **any instance of the Notifier** will be reported into the event handler.

To monitor internal events within any specific instance of the Notifier:

1. Get a reference to a specific instance of the Notifier.
2. Subscribe to its *InternalEvent* event.
3. Implement the event handler the way you see it.
4. As the result of this subscription, at runtime, all the Rollbar internal events generated while using **this specific instance of the Notifier** will be reported into the event handler.

For example, to demonstrate both levels of monitoring at the same time:

static void Main(string[] args)

{

RollbarQueueController.Instance.InternalEvent += OnRollbarInternalEvent;

const string postServerItemAccessToken = "…65fa5041749b6bf7095a190001…";

RollbarLocator.RollbarInstance

.Configure(new RollbarConfig(postServerItemAccessToken) { Environment = "proxyTest" })

.InternalEvent += OnRollbarInternalEvent

;

RollbarLocator.RollbarInstance

.Info("Basic info log example.")

.Debug("First debug log.")

.Error(new NullReferenceException())

.Error(new Exception("trying out the TraceChain", new NullReferenceException()))

;

System.Threading.Thread.Sleep(TimeSpan.FromSeconds(10));

}

private static void OnRollbarInternalEvent(object sender, RollbarEventArgs e)

{

Console.WriteLine(e.TraceAsString());

RollbarApiErrorEventArgs apiErrorEvent = e as RollbarApiErrorEventArgs;

if (apiErrorEvent != null)

{

//TODO: handle/report Rollbar API communication error event...

return;

}

CommunicationEventArgs commEvent = e as CommunicationEventArgs;

if (commEvent != null)

{

//TODO: handle/report Rollbar API communication event...

return;

}

CommunicationErrorEventArgs commErrorEvent = e as CommunicationErrorEventArgs;

if (commErrorEvent != null)

{

//TODO: handle/report basic communication error while attempting to reach

// Rollbar API service...

return;

}

InternalErrorEventArgs internalErrorEvent = e as InternalErrorEventArgs;

if (internalErrorEvent != null)

{

//TODO: handle/report basic internal error while using the Rollbar Notifier...

return;

}

}

# Integration of the Notifier into .NET Application Hosts

In many cases, .NET applications already use a generic logging/error handling infrastructure, like the plain .NET tracing, Microsoft Enterprise Library Logging, log4net, Nlog, Serilog, etc. or a custom solution. Most, if not all, of these libraries support some native ways to attach different third-party tracing/logging data persistence and delivery solutions like Rollbar. Some of the such hooks can be achieved by implementing specific listeners (as in .NET tracing) or sinks (as in Serilog case). So, you can always implement proper custom “hook”/plug-in that internally forwards all the traceable data to Rollbar via the Rollbar.NET Notifier. Please, referrer to the specific library documentation to get directions on proper implementation of such integrations.

When you do not use any kind of third party or custom logging infrastructure, here are a few examples of simple usage of the Notifier within different kinds of .NET applications.

## Asp.Net MVC

To use inside an ASP.Net Application, first in your global.asax.cs and Application\_Start method initialize Rollbar

protected void Application\_Start()

{

...

Rollbar.Init(new RollbarConfig

{

AccessToken = ConfigurationManager.AppSettings["Rollbar.AccessToken"],

Environment = ConfigurationManager.AppSettings["Rollbar.Environment"]

});

...

}

Then create a global action filter

public class RollbarExceptionFilter : IExceptionFilter

{

public void OnException(ExceptionContext filterContext)

{

if (filterContext.ExceptionHandled)

return;

RollbarLocator.RollbarInstance.Error(filterContext.Exception);

}

}

and finally add it to the global filters collection

private static void RegisterGlobalFilters(GlobalFilterCollection filters)

{

...

filters.Add(new RollbarExceptionFilter());

}

## Winforms

To use inside a Winforms Application, do the following inside your main method:

[STAThread]

static void Main()

{

Rollbar.Init(new RollbarConfig

{

AccessToken = "POST\_SERVER\_ACCESS\_TOKEN",

Environment = "production"

});

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.ThreadException += (sender, args) =>

{

RollbarLocator.RollbarInstance.Error(args.Exception);

};

AppDomain.CurrentDomain.UnhandledException += (sender, args) =>

{

RollbarLocator.RollbarInstance.Error(args.ExceptionObject as System.Exception);

};

Application.Run(new Form1());

}

## WPF

Example of using Rollbar.NET inside of a WPF application. It is optional to set the user for Rollbar and can be reset to a different user at any time. This example includes a default user being set with MainWindow.xml loads by calling the SetRollbarReportingUser function.

App.cs:

namespace Sample

{

/// <summary>

/// Interaction logic for App.xaml

/// </summary>

public partial class App : Application

{

protected override void OnStartup(StartupEventArgs e)

{

base.OnStartup(e);

System.Diagnostics.Debug.WriteLine("App Start Up");

//Initialize Rollbar

Rollbar.Init(new RollbarConfig

{

AccessToken = "<your rollbar token>",

Environment = "production"

});

// Setup Exception Handler

AppDomain.CurrentDomain.UnhandledException += (sender, args) =>

{

RollbarLocator.RollbarInstance.Error(args.ExceptionObject as System.Exception);

};

}

}

}

MainWindow.cs:

namespace Sample

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

System.Diagnostics.Debug.Write("Starting MainWindow");

InitializeComponent();

//Set Default User for RollbarReporting

// -- Reset if user logs in or wait to call SetRollbarReportingUser until user logs in

SetRollbarReportingUser("id", "myEmail@example.com", "default");

}

private void SetRollbarReportingUser(string id, string email, string userName)

{

Person person = new Person(id);

person.Email = email;

person.UserName = userName;

Rollbar.PersonData(() => person);

}

}

}

# Additional Resources

1. For the most complete Rollbar documentation, please, follow this link: <https://rollbar.com/docs/>