v5 Documentation

Rollbar.NET SDK

[**The Big Picture**](#_56x0h296wmm6) **2**

[Primary Motivation Behind v5 of the SDK](#_30ocsbqf1d3z) 2

[Overview of Main Changes in v5 as Compare to v4](#_4it17feun5e5) 2

[Core Abstractions of the SDK](#_1vvb626l9y1) 3

[Rollbar Configuration](#_s6wuuk9fe81p) 3

[Key Configuration Related Interfaces](#_q6z5wbhgt9zt) 3

[Key Configuration Related Classes](#_t1eyhqyenvj0) 5

[Rollbar Infrastructure](#_y5qzonbkumx0) 6

[Rollbar Loggers](#_btfkdpf2fnj5) 8

[Key Interfaces](#_h7przdhyfnik) 8

[Key Classes](#_k0ur54h7s2yw) 8

[Rollbar Assistant](#_ltqcbryu5ung) 9

[Rollbar Packages and Package Decorators](#_k0mypwyj2smj) 10

[Rollbar Internal Events](#_xf2rak6w2r4j) 13

[Rollbar Exceptions](#_xzrc604vylyo) 14

[**How-To by Example**](#_w56g1mifao47) **15**

[Initialize the Infrastructure](#_kpcd8t4dr3ve) 15

[Use Shared Logger Singleton](#_jz365lb2o04d) 16

[When the Infrastructure is Initialized](#_za7un6vr42ly) 16

[When the Infrastructure is Not Used](#_ca4xqnme8azs) 16

[Use Scoped (Short-Lived) Logger Instances](#_vwlw5z2ibukv) 17

[When the Infrastructure is Initialized](#_6hr60rmyz3w3) 17

[When the Infrastructure is Not Used](#_h1i718vkjfc1) 17

[Use Packages and Package Decorators](#_6kn1owqh4hg5) 18

[Monitor SDK Internal Events](#_wum6o1lf4cp8) 18

[Capture Extra State When Reporting Errors](#_mkxsbu3hryr2) 19

[Capture Telemetry Events/Data](#_pgduft1kt5bj) 20

[**SDK Usage Working Samples**](#_bm6xlp3yku6y) **21**

# The Big Picture

## Primary Motivation Behind v5 of the SDK

The primary motivation for making another major version of the SDK is the addition of Blazor Client-Side support.

With the relatively recent introduction of Blazor, its development stack for the server side worked fine with our SDK requiring no SDK changes to support the new technology on the server side. However, in-browser support of Blazor was not possible due to the current limitations of the Blazor implementation on the client-side. As of now, Blazor client-side runtime environment does not allow creation of background threads (at least not yet).

Pre-v5 SDK was fundamentally designed around its own internal payload processing infrastructure that tries to offload as mauch payload composition and processing on a dedicated background thread “running” the infrastructure.

## Overview of Main Changes in v5 as Compare to v4

So, in order to add Blazor client-side support to the SDK we had to make that dependency on the infrastructure optional. Hence, we had to make the infrastructure initialization as an explicit and optional step within v5 usage pattern.

To properly support both SDK usage scenarios (infrastructure based vs. infrastructure independent) we had to change the layout of what used to be known as `RollbarConfig` object in pre-v5 incarnations of the SDK. In v5, it was replaced by the `RollbarInfrastructureConfig` and `RollbarLoggerConfig` where each of the configuration objects is now designed in a more organized and structured manner. Each configuration is composed of what we call configuration options that are atomic custom typed configuration objects representing specific aspects/areas of the SDK configuration. For example, destination options, developer options, data security options, telemetry options, etc.

It is critical to understand that some of the options (i.e. corresponding SDK features) are only available within infrastructure based usage scenarios. For example, today, offline store options or telemetry options are only available if the infrastructure is used.

So, a new abstraction of RollbarInfrastructure was introduced with v5 of the SDK. It is designed as a singleton based application wide service. If a particular SDK usage scenario relies on the SDK the singleton must be initialized with a valid `RollbarInfrastructureConfig` object first - before any attempts to get access to any of the RollbarLogger instances. From there on, the `RollbarLocator` or/and `RollbarFactory` will provide you with proper logger reference to a proper `IRollbar` implementation depending on whether the infrastructure was actually initialized or not.

Past that point, all the usage patterns of the `IRollbar` implementations should look identical to the pre-v5 usage patterns.

Most (if not all) plug-in modules were not affected by these changes at all - from the client code/usage point of view.

## Core Abstractions of the SDK

### Rollbar Configuration

Here we describe core concepts and patterns used to configure the available SDK options.

#### Key Configuration Related Interfaces

* `IRollbarInfrastructureConfig`
* `IRollbarLoggerConfig`
* `I...Options`

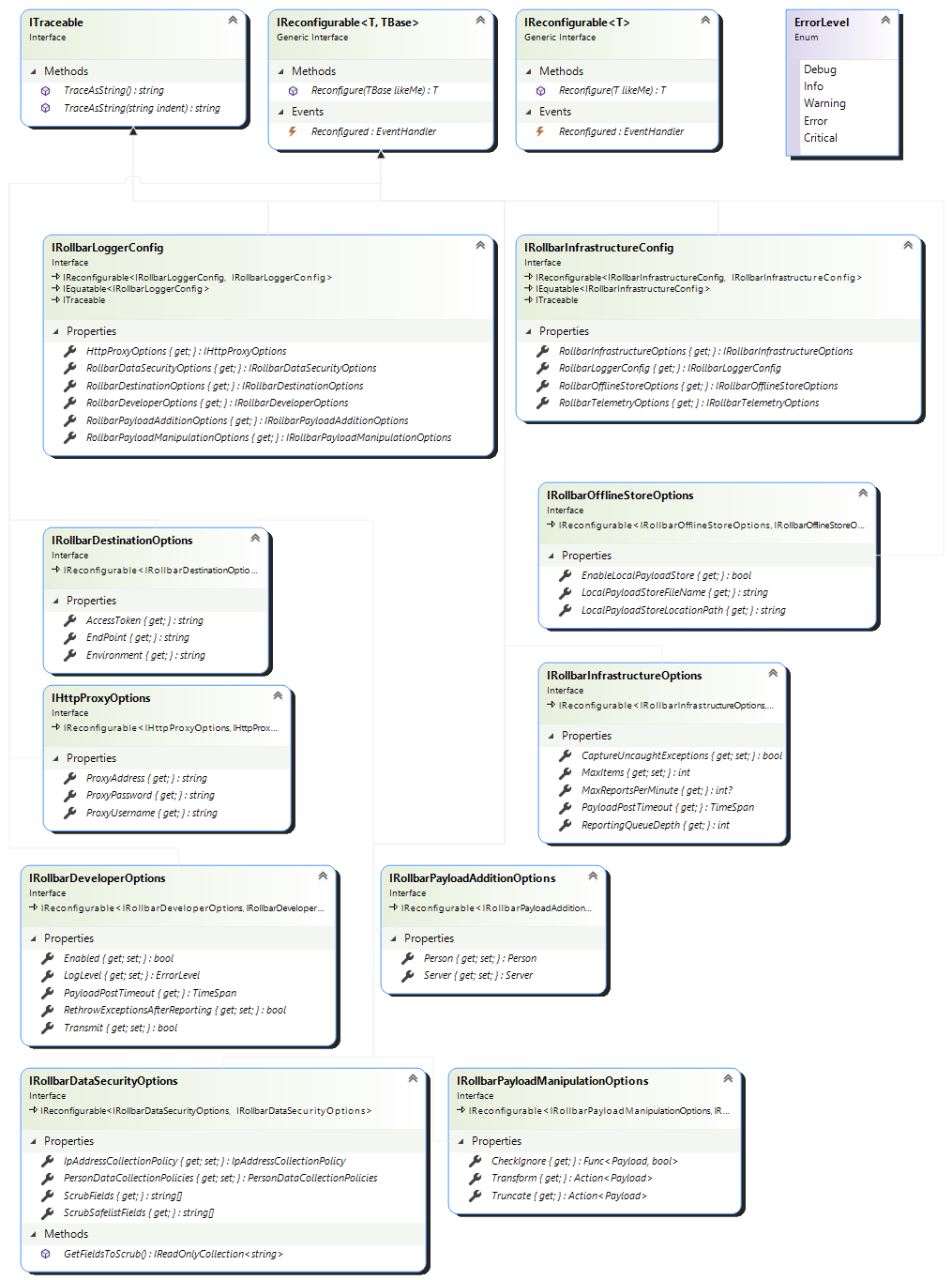
The Configs are composed of Options., i.e. have Options as their properties.

`IRollbarInfrastructureConfig` also includes the RollbarLoggerConfig property of `IRollbarLoggerConfig` type.

All of these interfaces (both the Configs and Options) are `IReconfigurable` and `ITracable`.

Most of the Options’ properties are Read-Only.

The primary way to modify either Config or Options objects exposed via the interfaces is by reconfiguring the objects based on provided similar prototype objects. Just call the `Reconfigure(prototype)` method on a Config or Options interface at hand.



#### Key Configuration Related Classes

Every (pimarilly Read-Only) `I...Config` or `I...Options` has a Read-Write capable matching class implementing each interface (just remove `I` prefix from an interface name to get the corresponding implementing class name).

Normally, while using Configs and Options, you instantiate needed class and either manually set all the desired properties on that instance or load it from an appropriate configuration file using proper `IRollbarConfigurationLoader` implementation (check out Rollbar.App.Config or/and RollbarAppSettings.Json modules).

Once the instance is properly configured/loaded as needed it can be used to reconfigure any existing configuration object exposed via the similar configuration interface by calling the `Reconfigure(prototype)` method on target configuration/options to be reconfigured based on the provided prototype configuration/options.

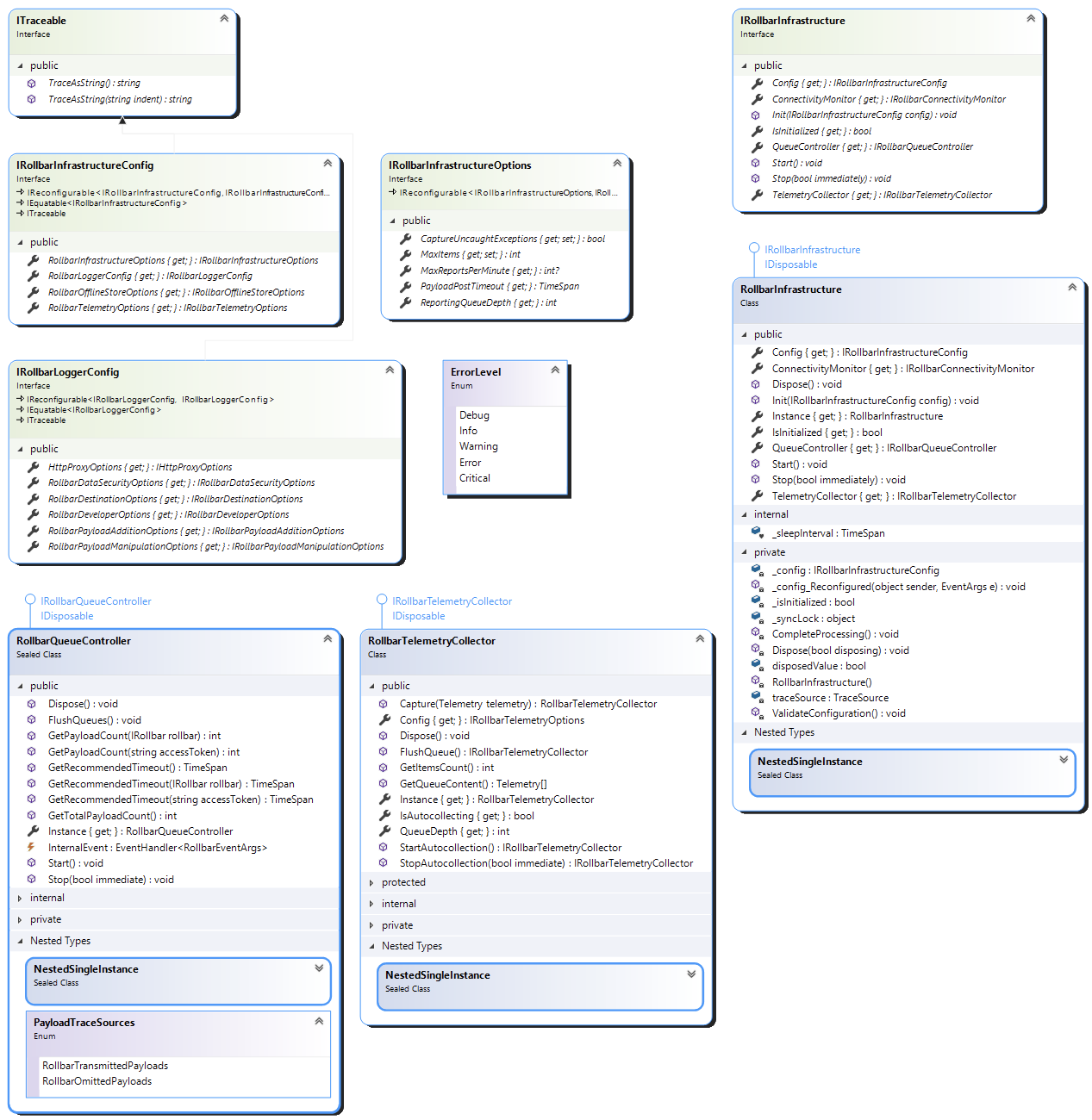
### 

### Rollbar Infrastructure

The `RollbarInfrastructure` abstraction is an optional (but recommended when it is possible to use it) concept that implements most of the payload composition, processing and delivery functionality of the SDK ran on a dedicated background thread so it offloads most of the work done by the SDK from the client threads calling IRollbar and ILogger methods (i.e. the logging methods) of the SDK.

The main reason for this abstraction to be optional is that some runtime environments may not allow/support background thread creation/processing. For example, browser side Blazor implementation as it stands today.

To take advantage of this infrastructure while logging to Rollbar, just initialize the `RollbarInfrastructure` singleton with a valid configuration object before any first use of either `RollbarLocator` or `RollbarFactory` classes. Usually, you would want to do it as early as possible within your application process’s startup/load/initialization.



### 

### Rollbar Loggers

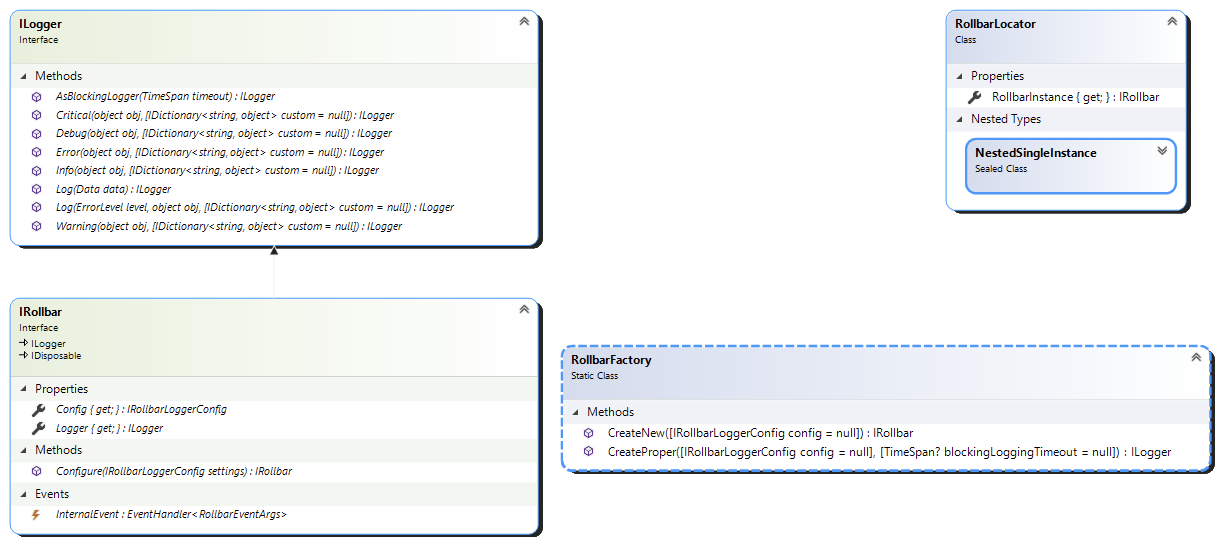
Rollbar loggers are used to collect and send logged data/events to a Project Dashboard allocated on Rollbar.com.

#### Key Interfaces

* `ILogger` - interface for async and synchronous event/data logging
* `IRollbar` - extends `ILogger` with Rollbar specific configuration and convenience methods, properties and events.

#### Key Classes

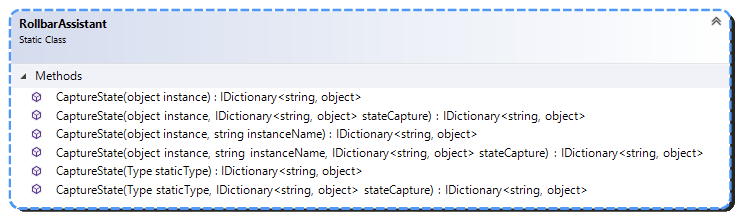
* `RollbarLocator` - a service-class/singleton providing application-wide access to the shared instance of `IRollbar` implementation;
* `RollbarFactory` - a utility class for creating disposable `IRollbar` implementation instances as needed (usually, short-lived ones within the C# `using` blocks).



### 

### Rollbar Assistant

`RollbarAssistant` is a utility class providing convenient ways to capture arbitrary object’s property values into a key-value (i.e. PropertyName-PropertyValue) dictionary so that the dictionary can be attached as a metadata for any SDK logging method.



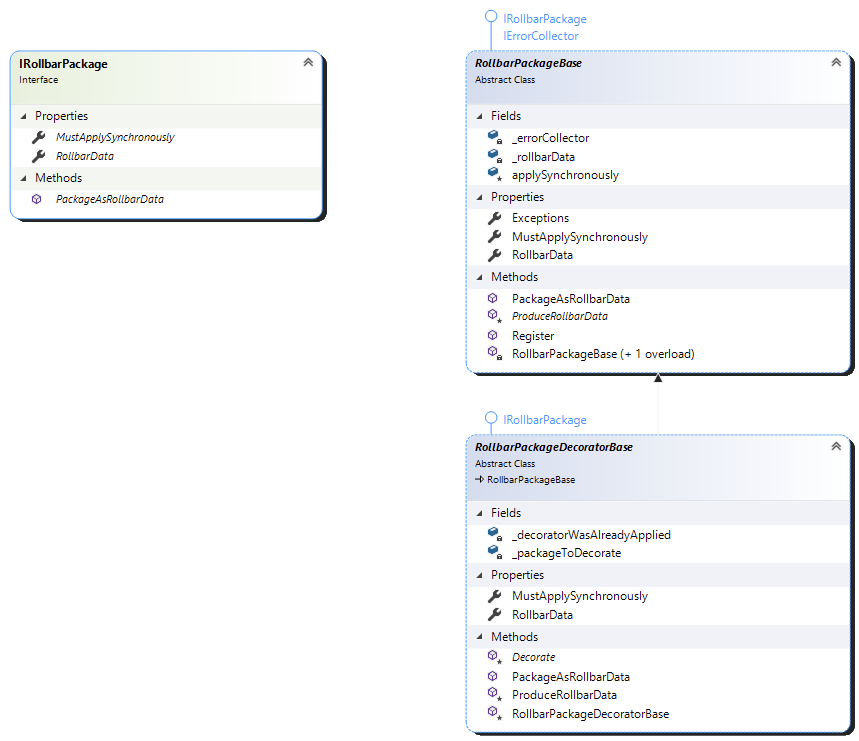
### 

### Rollbar Packages and Package Decorators

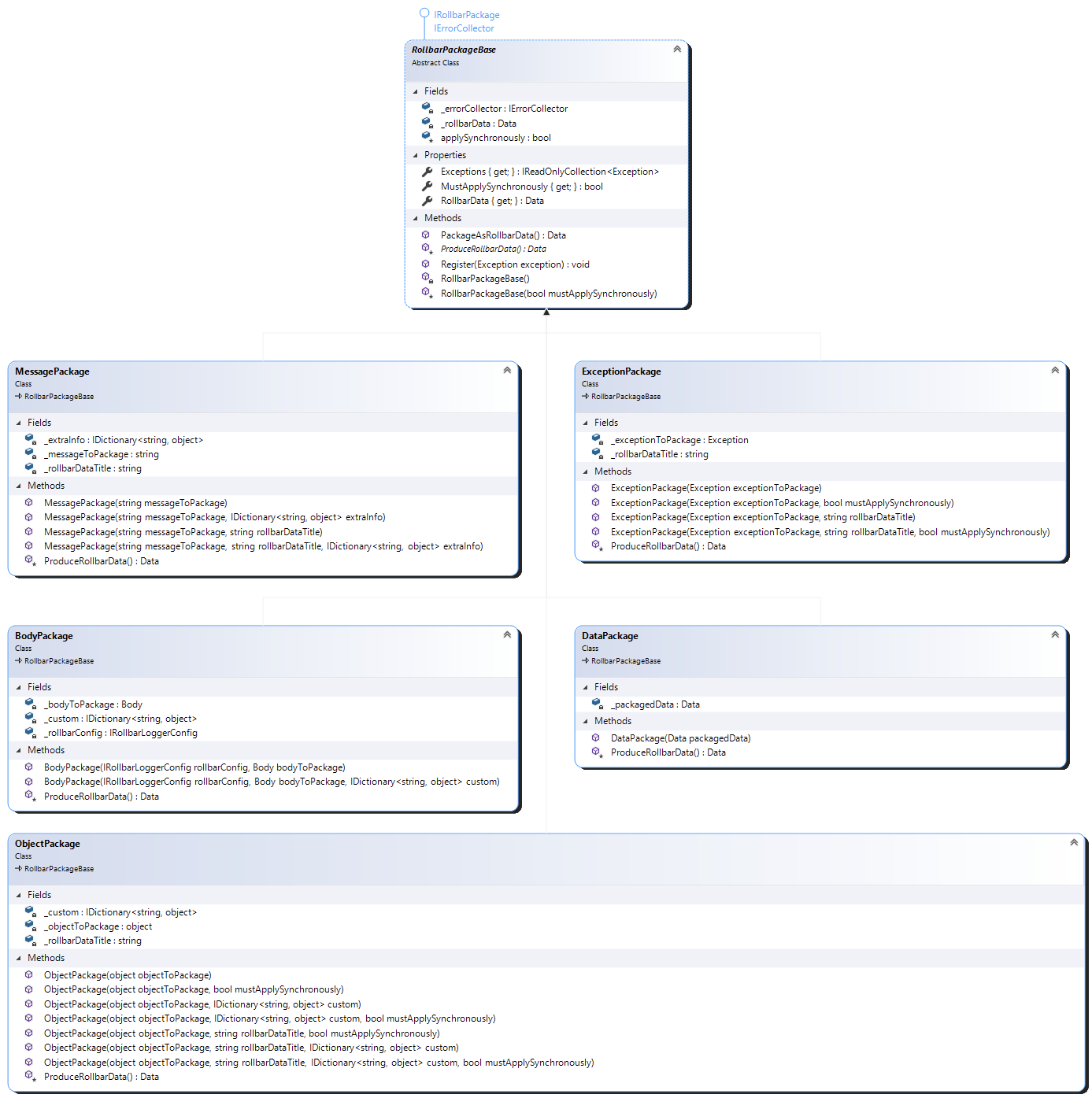
You can think of the Packages as an abstraction that allows to capture and bundle up either specific or arbitrary objects and turn them into Rollbar payloads that could be sent to Rollbar.com. They also allow for a high level of encapsulation and reuse of the functionality required to turn any data into a Rollbar payload. A package can also declare whether the packaged data can be captured later on by the Rollbar infrastructure in the background or must be captured synchronously at the time when the package is passed into the IRollbar logging method.

Each package can be decorated with an infinite amount of package decorators (that are packages on their own). Each decorator is meant to add an arbitrary extra data to a wrapped/decorated package and to have that data appended to the same corresponding payload.

The decorators are applied in the same order they were added to/around the original package.



Currently, the core Rollbar module of the SDK defines following packages:



Other SDK modules may define additional packages, or you can create your own custom ones.

Here are some of the package decorators defined within the core Rollbar module:



Other SDK modules define a lot of other applicable decorators, and you can always create your own custom package decorators.

### Rollbar Internal Events

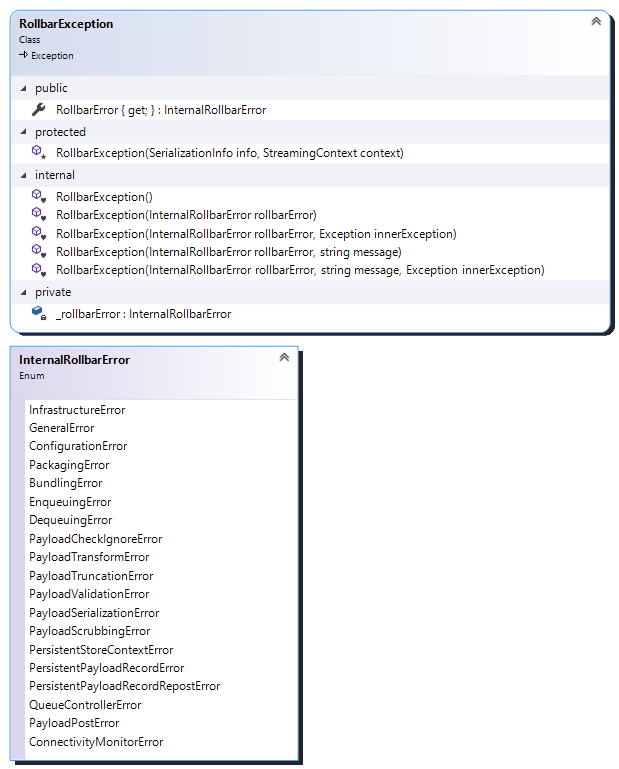
Being a developer tool, we do not want to bring down an application that is hosting the SDK in case anything goes wrong for any reason inside the SDK. So, we are doing everything possible to safely intercept and handle all the SDK internal exceptions if any. Such exceptions are turned into what we call SDK Internal Events. We also report significant internal operation events of the SDK as similar Internal Events.



Every individual IRollbar instance offer a subscription to the Intern Event (via C# InternalEvent event). Similarly, all the events from any IRollbar instance as well as Infrastructure specific Internal Events are reported as InternalEvent of the `RollbarInfrastructure.Instance.QueueController` singleton.

### Rollbar Exceptions

In cases when an internal error happens and we cannot gracefully recover from, such an error will be reported as a RollbarExceptions.



# How-To by Example

## Initialize the Infrastructure

Use of the Infrastructure is optional and not always possible (for example, within Blazor client side). However, we do recommend use of the infrastructure whenever possible. It provides significant performance benefits and adds extra features that are not available otherwise.

To enable the Infrastructure, you need to initialize it with a valid configuration object before using any of the Rollbar loggers. For example:

// minimally required Rollbar configuration:

RollbarInfrastructureConfig config =

new RollbarInfrastructureConfig(

RollbarSamplesSettings.AccessToken,

RollbarSamplesSettings.Environment

);

// optional:

RollbarOfflineStoreOptions offlineStoreOptions = new RollbarOfflineStoreOptions();

offlineStoreOptions.EnableLocalPayloadStore = true;

config.RollbarOfflineStoreOptions.Reconfigure(offlineStoreOptions);

// optional:

RollbarTelemetryOptions telemetryOptions = new RollbarTelemetryOptions(true, 3);

config.RollbarTelemetryOptions.Reconfigure(telemetryOptions);

// optional:

HttpProxyOptions proxyOptions = new HttpProxyOptions("http://something.com");

config.RollbarLoggerConfig.HttpProxyOptions.Reconfigure(proxyOptions);

// optional:

RollbarDataSecurityOptions dataSecurityOptions = new RollbarDataSecurityOptions();

dataSecurityOptions.ScrubFields = new string[] {

"user\_password",

"secret",

};

config.RollbarLoggerConfig.RollbarDataSecurityOptions.Reconfigure(dataSecurityOptions);

// optional:

RollbarPayloadAdditionOptions payloadAdditionOptions = new RollbarPayloadAdditionOptions();

payloadAdditionOptions.Person = new Person() {

Id = "007",

Email = "jbond@mi6.uk",

UserName = "JBOND",

}; config.RollbarLoggerConfig.RollbarPayloadAdditionOptions.Reconfigure(payloadAdditionOptions);

// initialize the Rollbar Infrastructure:

RollbarInfrastructure.Instance.Init(config);

// optionally, if you would like to monitor all Rollbar instances' internal events:

RollbarInfrastructure.Instance.QueueController.InternalEvent += OnRollbarInternalEvent;

There are many more configuration settings/options available to play with. Visual Studio’s IntelliSense is your friend when exploring available options. We will be documenting all the options here as well but this documentation could become a bit out-dated as time goes by.

## Use Shared Logger Singleton

Remember, if you do not use the Rollbar Infrastructure, the shared logger needs to be properly configured before making the first logging method call on it. Please, see details below.

### When the Infrastructure is Initialized

Whenever the Rollbar Infrastructure was initialized prior to the very first access to the shared logger instance, the logger would deduce its configuration from the Infrastructure. Hence, you can just start using the shared logger instance like so:

RollbarLocator.RollbarInstance.Info("Hello from the Rollbar Shared Logger", customFields);

### When the Infrastructure is Not Used

If the Rollbar Infrastructure is not initialized/used, the shared logger needs to be properly configured before making the first logging method call on it, for example:

// minimally required Rollbar logger configuration:

RollbarLoggerConfig config =

new RollbarLoggerConfig(

RollbarSamplesSettings.AccessToken,

RollbarSamplesSettings.Environment

);

// optional:

RollbarDataSecurityOptions dataSecurityOptions = new RollbarDataSecurityOptions();

dataSecurityOptions.ScrubFields = new string[] {

"user\_password",

"secret",

};

config.RollbarLoggerConfig.RollbarDataSecurityOptions.Reconfigure(dataSecurityOptions);

// optional:

RollbarPayloadAdditionOptions payloadAdditionOptions = new RollbarPayloadAdditionOptions();

payloadAdditionOptions.Person = new Person() {

Id = "007",

Email = "jbond@mi6.uk",

UserName = "JBOND",

}; config.RollbarLoggerConfig.RollbarPayloadAdditionOptions.Reconfigure(payloadAdditionOptions);

// configure the shared logger with a valid configuration:

RollbarLocator.RollbarInstance.Configure(config);

// start using the shared logger:

RollbarLocator.RollbarInstance.Info("Hello from the Rollbar Shared Logger", customFields);

## 

## Use Scoped (Short-Lived) Logger Instances

### When the Infrastructure is Initialized

Once the Infrastructure was properly initialized, you can just call `RollbarFactory.CreateNew()` to get a reference to a valid logger without even supplying any logger configuration instance for it. A proper configuration will be used based on the pre-configured Infrastructure:

using(var logger = RollbarFactory.CreateNew())

{

//...

logger.Info("Hello from a scoped logger!");

//...

}

And, of cause, if you would like to create an alternatively configured scoped logger, just supply `RollbarFactory.CreateNew(...)` with a valid RollbarLoggerConfig instance:

// minimally required Rollbar logger configuration:

RollbarLoggerConfig config =

new RollbarLoggerConfig(

RollbarSamplesSettings.AccessToken,

RollbarSamplesSettings.Environment

);

using(var logger = RollbarFactory.CreateNew(config))

{

//...

logger.Info("Hello from a scoped logger!");

//...

}

### When the Infrastructure is Not Used

In cases when the Infrastructure is not used, you always have to supply `RollbarFactory.CreateNew(...)` with a valid RollbarLoggerConfig instance:

// minimally required Rollbar logger configuration:

RollbarLoggerConfig config =

new RollbarLoggerConfig(

RollbarSamplesSettings.AccessToken,

RollbarSamplesSettings.Environment

);

using(var logger = RollbarFactory.CreateNew(config))

{

//...

logger.Info("Hello from a scoped logger!");

//...

}

## Use Packages and Package Decorators

Packages and Package Decorators provide a convenient way to capture a lot specifica extra data within a logging method call. The SDK supplies a wide variety of predefined packages and decorators, plus, you can always add your own custom ones.

Here is a simple example of their use:

IRollbarPackage rollbarPackage =

new ExceptionPackage(ex, $"{nameof(RollbarMiddleware)} processed uncaught exception.");

if (context != null)

{

if (context.Request != null)

{

rollbarPackage =

new HttpRequestPackageDecorator(rollbarPackage, context.Request, true);

}

if (context.Response != null)

{

rollbarPackage =

new HttpResponsePackageDecorator(rollbarPackage, context.Response, true);

}

}

RollbarLocator.RollbarInstance.Critical(rollbarPackage);

## Monitor SDK Internal Events

You can monitor the SDK Internal Events on an individual logger level or on the Infrastructure level:

// if you would like to monitor all Rollbar instances' internal events,

// for example, the shared one:

RollbarLocator.RollbarInstance.InternalEvent += OnRollbarInternalEvent;

// if you would like to monitor all Rollbar instances' internal events

// on the Infrastructure level:

RollbarInfrastructure.Instance.QueueController.InternalEvent += OnRollbarInternalEvent;

private static void OnRollbarInternalEvent(object sender, RollbarEventArgs e)

{

Console.WriteLine(e.TraceAsString());

switch (e)

{

case InternalErrorEventArgs rollbarEvent:

// handle this specific type of Rollbar event...

break;

case RollbarApiErrorEventArgs rollbarEvent:

// handle this specific type of Rollbar event...

break;

case CommunicationErrorEventArgs rollbarEvent:

// handle this specific type of Rollbar event...

break;

case TransmissionOmittedEventArgs rollbarEvent:

// handle this specific type of Rollbar event...

break;

case PayloadDropEventArgs rollbarEvent:

// handle this specific type of Rollbar event...

break;

case CommunicationEventArgs rollbarEvent:

// handle this specific type of Rollbar event...

break;

default:

// handle this specific type of Rollbar event...

break;

}

}

## Capture Extra State When Reporting Errors

You can use the `RollbarAssistant` utility class to easily capture extra data that could be useful when logged side by side with an error. The `RollbarAssistant` is capable of capturing the state of any arbitrary object including static data. For example:

var criticalObj = new InstanceType();

criticalObj.AutoProperty = -100;

try

{

///...

/// oh, no - we have an exception:

throw new System.Exception("An exception with state capture!");

///...

}

catch (System.Exception ex)

{

// capture state of this instance:

var state = RollbarAssistant.CaptureState(this, "Self");

// also, capture state of some other critical object:

state = new Dictionary<string, object>(

state.Concat(

RollbarAssistant.CaptureState(

criticalObj,

nameof(criticalObj)))

);

// also, capture current state of a static type:

state = new Dictionary<string, object>(

state.Concat(RollbarAssistant.CaptureState(typeof(StaticType))

));

// report the captured states along with the caught exception:

RollbarLocator.RollbarInstance.Error(ex, state);

}

## Capture Telemetry Events/Data

Whenever the Rollbar Infrastructure is properly initialized, you can use it to capture many kinds of Telemetry events. These events will be captured within a fixed-depth queue maintained by the infrastructure so that the last N events will be attached to every subsequent payload sent by the SDK (where N equal to the depth of the Telemetry queue). For example:

RollbarInfrastructure.Instance.TelemetryCollector.Capture(

new Telemetry(

TelemetrySource.Client,

TelemetryLevel.Info,

new LogTelemetry("Something interesting happened.")

)

);

//...

RollbarInfrastructure.Instance.TelemetryCollector.Capture(

new Telemetry(

TelemetrySource.Client,

TelemetryLevel.Error,

new ErrorTelemetry(new System.Exception("Worth mentioning!"))

)

);

//...

RollbarInfrastructure.Instance.TelemetryCollector.Capture(

new Telemetry(

TelemetrySource.Client,

TelemetryLevel.Error,

new ManualTelemetry(new Dictionary<string, object>() {{"somthing", "happened" },})

)

);

# SDK Usage Working Samples

We have a lot of sample applications that demonstrate different aspects of the SDK usage including the ones that rely on additional technology related extension modules of the SDK or various SDK plug-ins for specific commonly used application development libraries and frameworks.

The Sample solutions are always available here: <https://github.com/rollbar/Rollbar.NET/tree/master/Samples>