Stripe.net

The official Stripe .NET library, supporting .NET Standard 2.0+, .NET Core 2.0+, and .NET Framework 4.6.1+.

Installation

Using the .NET Core command-line interface (CLI) tools:

```
dotnet add package Stripe.net
```

Using the NuGet Command Line Interface (CLI):

```
nuget install Stripe.net
```

Using the Package Manager Console:

```
Install-Package Stripe.net
```

From within Visual Studio:

1. Open the Solution Explorer.
2. Right-click on a project within your solution.
3. Click on *Manage NuGet Packages...*
4. Click on the *Browse* tab and search for "Stripe.net".
5. Click on the Stripe.net package, select the appropriate version in the right-tab and click *Install*.

Documentation

For a comprehensive list of examples, check out the API documentation. See video demonstrations covering how to use the library.

Usage

Authentication

Stripe authenticates API requests using your account's secret key, which you can find in the Stripe Dashboard. By default, secret keys can be used to perform any API request without restriction.

Use `Stripe.ApiKey` property to set the secret key.

```
Stripe.ApiKey = "sk_test_...";
```

Creating a resource

The `Create` method of the service class can be used to create a new resource:

```
var options = new CustomerCreateOptions
{
    Email = "customer@example.com"
};

var service = new CustomerService();
Customer customer = service.Create(options);

// Newly created customer is returned
Console.WriteLine(customer.Email);
```

Retrieve a resource

The `Retrieve` method of the service class can be used to retrieve a resource:

```
var service = new CustomerService();
Customer customer = service.Get("cus_1234");

Console.WriteLine(customer.Email);
```

Updating a resource

The `Update` method of the service class can be used to update a resource:

```
var options = new CustomerUpdateOptions
{
    Email = "updated-email@example.com"
};

var service = new CustomerService();
Customer customer = service.Update("cus_123", options);

// The updated customer is returned
Console.WriteLine(customer.Email);
```

Deleting a resource

The `Delete` method of the service class can be used to delete a resource:

```
var service = new CustomerService();
Customer customer = service.Delete("cus_123", options);
```

Listing a resource

The `List` method on the service class can be used to list resources page-by-page.

> **NOTE** The `List` method returns only a single page, you have to manually continue the iteration using the `StartingAfter` parameter.

```
var service = new CustomerService();
var customers = service.List();

string lastId = null;

// Enumerate the first page of the list
foreach (Customer customer in customers)
{
   lastId = customer.Id;
   Console.WriteLine(customer.Email);
}

customers = service.List(new CustomerListOptions()
{
    StartingAfter = lastId,
});

// Enumerate the subsequent page
foreach (Customer customer in customers)
{
   lastId = customer.Id;
   Console.WriteLine(customer.Email);
}
```

Listing a resource with auto-pagination

The `ListAutoPaging` method on the service class can be used to automatically iterate over all pages.

```
var service = new CustomerService();
var customers = service.ListAutoPaging();

// Enumerate all pages of the list
foreach (Customer customer in customers)
{
   Console.WriteLine(customer.Email);
}
```

Per-request configuration

All of the service methods accept an optional `RequestOptions` object. This is used if you want to set an idempotency key, if you are using Stripe Connect, or if you want to pass the secret API key on each method.

```
var requestOptions = new RequestOptions();
requestOptions.ApiKey = "SECRET API KEY";
requestOptions.IdempotencyKey = "SOME STRING";
requestOptions.StripeAccount = "CONNECTED ACCOUNT ID";
```

Using a custom `HttpClient`

You can configure the library with your own custom `HttpClient`:

```
StripeConfiguration.StripeClient = new StripeClient(
    apiKey,
    httpClient: new SystemNetHttpClient(httpClient));
```

Please refer to the Advanced client usage wiki page to see more examples of using custom clients, e.g. for using a proxy server, a custom message handler, etc.

Automatic retries

The library automatically retries requests on intermittent failures like on a connection error, timeout, or on certain API responses like a status `409 Conflict`. Idempotency keys are always added to requests to make any such subsequent retries safe.

By default, it will perform up to two retries. That number can be configured with `StripeConfiguration.MaxNetworkRetries`:

```
StripeConfiguration.MaxNetworkRetries = 0; // Zero retries
```

How to use undocumented parameters and properties

stripe-dotnet is a typed library and it supports all public properties or parameters.

Stripe sometimes has beta features which introduce new properties or parameters that are not immediately public. The library does not support these properties or parameters until they are public but there is still an approach that allows you to use them.

Parameters

To pass undocumented parameters to Stripe using stripe-dotnet you need to use the `AddExtraParam()` method, as shown below:

```
var options = new CustomerCreateOptions
{
    Email = "jenny.rosen@example.com"
}
options.AddExtraParam("secret_feature_enabled", "true");
options.AddExtraParam("secret_parameter[primary]", "primary value");
options.AddExtraParam("secret_parameter[secondary]", "secondary value");

var service = new CustomerService();
var customer = service.Create(options);
```

Properties

To retrieve undocumented properties from Stripe using csharp you can use an option in the library to return the raw JSON object and return the property. An example of this is shown below:

```
var service = new CustomerService();
var customer = service.Get("cus_1234");

customer.RawJObject["secret_feature_enabled"];
customer.RawJObject["secret_parameter"]["primary"];
customer.RawJObject["secret_parameter"]["secondary"];
```

Writing a plugin

If you're writing a plugin that uses the library, we'd appreciate it if you identified using `StripeConfiguration.AppInfo`:

```
StripeConfiguration.AppInfo = new AppInfo
{
    Name = "MyAwesomePlugin",
    Url = "https://myawesomeplugin.info",
    Version = "1.2.34",
};
```

This information is passed along when the library makes calls to the Stripe API. Note that while `Name` is always required, `Url` and `Version` are optional.

Beta SDKs

Stripe has features in the beta phase that can be accessed via the beta version of this package. We would love for you to try these and share feedback with us before these features reach the stable phase. To install a beta version of Stripe.net use the version parameter with `dotnet add package` command:

```
dotnet add package Stripe.net --version 40.3.0-beta.1
```

> **Note** There can be breaking changes between beta versions. Therefore we recommend pinning the package version to a specific beta version in your project file. This way you can install the same version each time without breaking changes unless you are intentionally looking for the latest beta version.

We highly recommend keeping an eye on when the beta feature you are interested in goes from beta to stable so that you can move from using a beta version of the SDK to the stable version.

If your beta feature requires a `Stripe-Version` header to be sent, use the `StripeConfiguration.ApiVersion` property to set it:

> **Note** The `ApiVersion` can only be set in beta versions of the library.

```
StripeConfiguration.ApiVersion += "; feature_beta=v3";
```

Support

New features and bug fixes are released on the latest major version of the Stripe .NET client library. If you are on an older major version, we recommend that you upgrade to the latest in order to use the new features and bug fixes including those for security vulnerabilities. Older major versions of the package will continue to be available for use, but will not be receiving any updates.

Development

The test suite depends on stripe-mock, so make sure to fetch and run it from a background terminal (stripe-mock's README also contains instructions for installing via Homebrew and other methods):

```
go get -u github.com/stripe/stripe-mock
stripe-mock
```

Run all tests from the `src/StripeTests` directory:

```
dotnet test
```

Run some tests, filtering by name:

```
dotnet test --filter FullyQualifiedName~InvoiceServiceTest
```

Run tests for a single target framework:

```
dotnet test --framework netcoreapp2.1
```

The library uses `dotnet-format` for code formatting. Code must be formatted before PRs are submitted, otherwise CI will fail. Run the formatter with:

```
dotnet format src/Stripe.net.sln
```

For any requests, bug or comments, please open an issue or submit a pull request.