Getting Started with OpenRouter SDK: A Complete Guide to Simple Text Completion

Welcome to this comprehensive tutorial on using the OpenRouter SDK for .NET! In this guide, we'll walk through Example01.SimpleTextCompletion, a foundational project that demonstrates how to make your first API call to OpenRouter's AI models.

Table of Contents

Overview

This example demonstrates the most basic use case of the OpenRouter SDK: sending a simple text prompt to an AI model and receiving a response. It's the perfect starting point for beginners who want to understand how to integrate AI capabilities into their .NET applications.

What you'll learn:

- How to configure a .NET console application with OpenRouter SDK
- How to initialize the OpenRouter client
- How to create and send a chat completion request
- How to handle responses and errors gracefully
- Best practices for API key management

Project Structure

The Example01.SimpleTextCompletion project consists of two main files:

```
Example01.SimpleTextCompletion/
├── Example01.SimpleTextCompletion.csproj  # Project configuration file
├── Program.cs                              # Main application code
└── BLOG.md                                 # This guide
```

Additionally, it depends on:

- **OpenRouter.SDK**: The main SDK package
- **OpenRouter.Examples.EnvConfig**: A shared configuration library for managing API keys

---

Prerequisites

Before starting, ensure you have:

1. **.NET 8.0 SDK** installed (Download here)
2. **OpenRouter API Key** (Get one here)
3. **Visual Studio 2022** or **VS Code** (optional but recommended)
4. Basic understanding of C# and async/await patterns

Understanding the Project File

Let's examine `Example01.SimpleTextCompletion.csproj` in detail:

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net8.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="OpenRouter.SDK" Version="1.0.0" />
  </ItemGroup>

  <ItemGroup>
    <ProjectReference
Include="..\OpenRouter.Examples.EnvConfig\OpenRouter.Examples.EnvConfig.csproj"
/>
  </ItemGroup>

</Project>
```

Breaking Down the Project File

PropertyGroup Section

```
<PropertyGroup>
  <OutputType>Exe</OutputType>
  <TargetFramework>net8.0</TargetFramework>
  <ImplicitUsings>enable</ImplicitUsings>
  <Nullable>enable</Nullable>
</PropertyGroup>
```

- **OutputType**: `Exe` indicates this is a console application that produces an executable
- **TargetFramework**: `net8.0` specifies we're using .NET 8.0, the latest LTS version
- **ImplicitUsings**: Automatically includes common namespaces like `System`, `System.Collections.Generic`, etc.
- **Nullable**: Enables nullable reference types for better null-safety

Package References

```
<ItemGroup>
  <PackageReference Include="OpenRouter.SDK" Version="1.0.0" />
</ItemGroup>
```

This section declares our dependency on the OpenRouter SDK NuGet package. This package provides:

- `OpenRouterClient` class for API communication
- Request/Response models
- Exception types for error handling
- Streaming capabilities

Project References

```
<ItemGroup>
  <ProjectReference
Include="..\OpenRouter.Examples.EnvConfig\OpenRouter.Examples.EnvConfig.csproj"
/>
</ItemGroup>
```

We reference the shared configuration project which handles:

- Loading API keys from `.env` files
- Managing environment variables
- Providing default model configurations

Setting Up Configuration

Before running the example, you need to configure your API key. The `ExampleConfig` class looks for configuration in this order:

1. `.env` file in the solution root
2. Environment variables
3. Default values (where applicable)

Creating a .env File

Create a `.env` file in your solution root directory:

```
# Required: Your OpenRouter API Key
OPENROUTER_API_KEY=your_api_key_here

# Optional: Default model to use
OPENROUTER_MODEL=openai/gpt-4

# Optional: Base URL (default: https://openrouter.ai/api/v1)
OPENROUTER_BASE_URL=https://openrouter.ai/api/v1

# Optional: Analytics
OPENROUTER_SITE_URL=https://yourwebsite.com
OPENROUTER_SITE_NAME=YourAppName
```

Configuration Properties

The `ExampleConfig` class provides these properties:

- **ApiKey**: Your OpenRouter API key (required)
- **ModelName**: The AI model to use (default: `openai/gpt-3.5-turbo`)
- **BaseUrl**: API endpoint (default: `https://openrouter.ai/api/v1`)
- **SiteUrl**: Your website for analytics (optional)
- **SiteName**: Your app name for analytics (optional)

Code Walkthrough

Now let's dive into `Program.cs` and understand every line of code:

Step 1: Using Directives

```
using OpenRouter.SDK;
using OpenRouter.Examples.EnvConfig;
```

- **OpenRouter.SDK**: Imports the main SDK namespace containing `OpenRouterClient` and models
- **OpenRouter.Examples.EnvConfig**: Imports our configuration helper

Step 2: Welcome Messages

```
Console.WriteLine("============================================");
Console.WriteLine("Example 1: Simple Text Completion");
Console.WriteLine("============================================\n");
```

Simple console output to identify which example is running. This is helpful when running multiple examples.

Step 3: Execute the Example

```
await Example01.RunAsync();
```

Calls the main example method asynchronously. The `await` keyword ensures we wait for the async operation to complete before continuing.

Step 4: Completion Message

```
Console.WriteLine("\n==========================================");
Console.WriteLine("Example completed!");
Console.WriteLine("==========================================");
```

Indicates successful completion of the example.

Step 5: The Main Example Class

```
public static class Example01
{
    public static async Task RunAsync()
    {
        // Implementation here
    }
}
```

A static class containing our example logic. The method returns `Task` to support async/await patterns.

Step 6: Initialize the Client

```
// Get API key from .env file or environment variable
var apiKey = ExampleConfig.ApiKey;
var client = new OpenRouterClient(apiKey);
Console.WriteLine("=== Example 1: Simple Text Completion ===");
```

What's happening:

1. We retrieve the API key from configuration
2. Create a new `OpenRouterClient` instance with the API key
3. Print a header message

The `OpenRouterClient` is the main entry point for all SDK operations. It handles:

- HTTP communication with OpenRouter API
- Authentication (sending API key in headers)
- Request serialization and response deserialization
- Connection pooling and retry logic

Step 7: Create the Request

```
var request = new OpenRouter.SDK.Models.ChatCompletionRequest
{
    Model = ExampleConfig.ModelName,
    Messages = new List<OpenRouter.SDK.Models.Message>
    {
        new OpenRouter.SDK.Models.UserMessage
        {
            Role = "user",
            Content = "How many r's are in the word 'strawberry'?"
        }
    },
    Reasoning = new OpenRouter.SDK.Models.ReasoningConfig
    {
        Enabled = true
    }
};
```

Breaking down the request object:

Model Selection

```
Model = ExampleConfig.ModelName,
```

Specifies which AI model to use. Common options:

- `openai/gpt-4` - Most capable, higher cost
- `openai/gpt-3.5-turbo` - Fast and cost-effective
- `anthropic/claude-2` - Great for long contexts
- `meta-llama/llama-2-70b-chat` - Open source alternative

Messages Array

```
Messages = new List<OpenRouter.SDK.Models.Message>
{
    new OpenRouter.SDK.Models.UserMessage
    {
        Role = "user",
        Content = "How many r's are in the word 'strawberry'?"
    }
}
```

The conversation is represented as an array of messages. Each message has:

- **Role**: Who sent the message (`user`, `assistant`, or `system`)
- **Content**: The actual text of the message

The SDK provides strongly-typed message classes:

- `UserMessage`: Messages from the end user
- `AssistantMessage`: Responses from the AI

- `SystemMessage`: Instructions that guide the AI's behavior

Reasoning Configuration

```
Reasoning = new OpenRouter.SDK.Models.ReasoningConfig
{
    Enabled = true
}
```

Enables reasoning capabilities for supported models. When enabled:

- The model shows its thinking process
- You can see how it arrives at conclusions
- Useful for debugging and understanding AI decisions

Step 8: Send the Request

```
Console.WriteLine("Sending request (matches your curl command)...\n");

var response = await client.Chat.CreateAsync(request);
```

What happens here:

1. We print a status message
2. Call `client.Chat.CreateAsync()` which:
    - Serializes the request to JSON
    - Sends an HTTP POST to OpenRouter API
    - Waits for the response
    - Deserializes the JSON response
    - Returns a strongly-typed response object

The `await` keyword means this is an asynchronous operation. The method will pause here until the API responds, but won't block other threads.

Step 9: Display the Response

```
Console.WriteLine($"Response:\n{response.Choices[0].Message.Content}");
```

Understanding the response structure:

- `response.Choices`: Array of possible completions (usually just one)
- `[0]`: Get the first (and typically only) choice
- `.Message`: The message object containing the AI's response
- `.Content`: The actual text content

Error Handling

The example includes comprehensive error handling for common scenarios:

1. JSON Parsing Errors

```
catch (System.Text.Json.JsonException jsonEx)
{
    Console.WriteLine($"JSON Error: {jsonEx.Message}");
    Console.WriteLine($"\nThis usually means the API returned HTML instead of
JSON.");
    Console.WriteLine($"Common causes:");
    Console.WriteLine($"  1. Invalid API key");
    Console.WriteLine($"  2. API key doesn't have proper permissions");
    Console.WriteLine($"  3. Network/firewall blocking the request");
    Console.WriteLine($"\nPlease verify your API key at:
https://openrouter.ai/keys");
}
```

When this occurs:

- API returns HTML error page instead of JSON
- Usually indicates authentication or network issues
- Provides helpful troubleshooting steps

2. Authentication Errors

```
catch (OpenRouter.SDK.Exceptions.UnauthorizedException)
{
    Console.WriteLine("ERROR: Unauthorized - Your API key is invalid!");
    Console.WriteLine("Please get a valid API key from:
https://openrouter.ai/keys");
}
```

When this occurs:

- API key is missing or invalid
- API key has been revoked
- Clear, actionable error message

3. General Exceptions

```
catch (Exception ex)
{
    Console.WriteLine($"Error: {ex.Message}");
    if (ex.InnerException != null)
    {
        Console.WriteLine($"Inner Error: {ex.InnerException.Message}");
    }
}
```

Catches all other exceptions:

- Network connectivity issues
- Timeout errors

- Unexpected API responses
- Shows both outer and inner exception details

Running the Example

Using Visual Studio

1. Open `OpenRouter.SDK.sln` in Visual Studio
2. Set `Example01.SimpleTextCompletion` as the startup project (right-click → Set as Startup Project)
3. Press **F5** or click **Run**

Using .NET CLI

Navigate to the project directory and run:

```
cd Examples/Example01.SimpleTextCompletion
dotnet run
```

Using PowerShell/Command Prompt

```
# Navigate to the project folder
cd
"c:\Users\subhr\source\repos\OpenRouter.SDK\OpenRouter.SDK\Examples\Example01.Sim
pleTextCompletion"

# Run the project
dotnet run
```

Understanding the Output

When you run the example successfully, you'll see output like this:

```
==========================================
Example 1: Simple Text Completion
==========================================

=== Example 1: Simple Text Completion ===
Sending request (matches your curl command)...

Response:
The word "strawberry" contains three r's:
1. The first 'r' appears in position 3 (st-R-awberry)
2. The second 'r' appears in position 4 (str-R-awberry)
3. The third 'r' appears in position 9 (strawber-R-y)

So the answer is: there are 3 r's in the word 'strawberry'.

==========================================
Example completed!
```

```
==========================================
```

What to Look For

✅ **Success indicators:**

- No error messages
- Well-formatted response
- Completion message appears

❌ **Failure indicators:**

- Authentication errors → Check your API key
- JSON errors → Verify API key and network
- Timeout errors → Check internet connection

Key Concepts Explained

Async/Await Pattern

```
await Example01.RunAsync();
```

- **Why use async?** API calls are I/O-bound operations that can take seconds
- **Benefits:** Keeps your application responsive during network operations
- **Task:** Represents an asynchronous operation that returns type T

Strongly-Typed Models

The SDK uses C# classes instead of raw JSON:

```
var request = new ChatCompletionRequest { ... };
```

Advantages:

- IntelliSense support in your IDE
- Compile-time type checking
- Better refactoring support
- Self-documenting code

Configuration Separation

API keys are stored separately from code:

```
var apiKey = ExampleConfig.ApiKey;
```

Security benefits:

- Keys never committed to source control
- Easy to use different keys per environment

- Follows 12-factor app principles

Common Issues and Solutions

Issue 1: "API Key not found"

Solution:

1. Create a `.env` file in the solution root
2. Add `OPENROUTER_API_KEY=your_key_here`
3. Ensure the file is saved

Issue 2: "Unauthorized" Error

Solution:

1. Verify your API key at https://openrouter.ai/keys
2. Ensure no extra spaces in the `.env` file
3. Check if the key has been revoked

Issue 3: "Model not found"

Solution:

1. Check the model name is correct
2. Visit https://openrouter.ai/models for available models
3. Update `OPENROUTER_MODEL` in your `.env` file

Issue 4: Slow Response Times

Possible causes:

- Large models take longer to respond
- Network latency
- High API load

Solutions:

- Try a faster model like `gpt-3.5-turbo`
- Check your internet connection
- Consider implementing timeout handling

Best Practices

1. **Always use async/await** for API calls to keep your app responsive
2. **Implement error handling** for network and API errors
3. **Store API keys securely** - never hardcode them
4. **Choose the right model** - balance cost, speed, and capability

5. **Monitor your usage** at https://openrouter.ai/activity
6. **Use try-catch blocks** for robust error handling
7. **Log API interactions** in production applications

Additional Resources

- **OpenRouter Documentation:** https://openrouter.ai/docs
- **Available Models:** https://openrouter.ai/models
- **API Keys:** https://openrouter.ai/keys
- **Pricing:** https://openrouter.ai/pricing
- **SDK Source Code:** Browse the `src/` folder in this repository

---

Conclusion

Congratulations! You've completed your first OpenRouter SDK integration. You now understand:

✅ How to structure a .NET project with OpenRouter SDK
✅ How to manage API keys securely
✅ How to create and send chat completion requests
✅ How to handle responses and errors
✅ Best practices for API integration

This foundation will serve you well as you build more complex AI-powered applications. Happy coding!

Troubleshooting and Support

If you encounter issues:

1. **Check the console output** for specific error messages
2. **Verify your API key** is correctly configured
3. **Review this guide** for missed steps
4. **Check OpenRouter status** at https://status.openrouter.ai
5. **Review other examples** in the Examples folder
6. **Open an issue** on the GitHub repository

Created: February 2026
SDK Version: 1.0.0
Target Framework: .NET 8.0
Example: Example01.SimpleTextCompletion
