

Market Data — Biblioteca .NET / C# (CD3 Connector)

Cedro Technologies · Documentação para o cliente · Versão 2026.1

1. Sobre este documento

Este documento descreve a biblioteca oficial `Cd3.Connector` para **.NET / C#**, que conecta a sua aplicação à API **Socket** do Cedro Market Data (protocolo **Cedro Crystal**, servido pelo Crystal Difusor Cedro).

Ele cobre instalação, uso e boas práticas da biblioteca. **Os comandos do protocolo** (`SQT`, `BQT`, `GQT`, `GCH` etc.), o formato das mensagens e os limites do seu pacote estão na **documentação da API Socket** do seu plano, enviada junto com este documento.

2. O que é a biblioteca

O Cedro Crystal é um servidor de streaming sobre TCP: você abre uma conexão, autentica, assina os dados que deseja e o servidor **empurra** as atualizações continuamente.

A `Cd3.Connector` encapsula toda a parte de transporte dessa conexão:

- **Handshake** — a sequência de autenticação logo após abrir a conexão.
- **Leitura do socket** — recebe as mensagens e as entrega à sua aplicação por *callback*.
- **Envio de comandos** — você envia os comandos do protocolo como texto (`"SQT PETR4"`).

Você passa usuário, senha e três funções de *callback* — e trata apenas as **mensagens** do protocolo. **Esta é a forma recomendada de integrar com a API Socket.**

3. Instalação

O pacote é distribuído pelo **feed NuGet da Cedro**.



Item	Valor
Pacote	Cd3.Connector
Namespace	Cd3Connector
Feed NuGet	https://nuget.cedrotech.com/v3/index.json

3.1 Adicione o feed da Cedro (NuGet.Config)

Adicione o arquivo **NuGet.Config** na raiz da solução:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <packageSources>
    <add key="nuget.org" value="https://api.nuget.org/v3/index.json" protocolVersion="3" />
    <add key="Cedro-Cedro" value="https://nuget.cedrotech.com/v3/index.json" />
  </packageSources>
  <packageRestore>
    <add key="enabled" value="True" />
    <add key="automatic" value="True" />
  </packageRestore>
  <bindingRedirects>
    <add key="skip" value="False" />
  </bindingRedirects>
  <packageManagement>
    <add key="format" value="0" />
    <add key="disabled" value="False" />
  </packageManagement>
</configuration>
```

3.2 Instale o pacote no projeto

```
dotnet add <MeuProjeto>.csproj package Cd3.Connector -s
https://nuget.cedrotech.com/v3/index.json
```



4. Como usar

4.1 Inicialize o conector

Crie uma instância da classe `Connector`, passando usuário, senha e as funções de *callback* para os eventos de conexão, desconexão e mensagem recebida — **nessa ordem**:

```
using Cd3Connector;

class Program
{
    static void Main(string[] args)
    {
        var connector = new Connector(
            Environment.GetEnvironmentVariable("CEDRO_USER"),
            Environment.GetEnvironmentVariable("CEDRO_PASSWORD"),
            OnConnect, OnDisconnect, OnMessage);

        connector.Start(); // inicia a conexão (faz o handshake)
    }

    static void OnConnect()
    {
        Console.WriteLine("Conectado com sucesso!");
    }

    static void OnDisconnect()
    {
        Console.WriteLine("Desconectado!");
    }

    static void OnMessage(string message)
    {
        Console.WriteLine($"Mensagem recebida: {message}");
    }
}
```

3



Av. Anselmo Alves dos Santos, 1111
Uberlândia-MG



www.cedrotech.com



contato@cedrotech.com



(34) 3239-0003

```
}  
}
```

⚠ **Não escreva usuário e senha no código.** Leia as credenciais de variáveis de ambiente ou de um cofre de segredos, como no exemplo acima.

4.2 Métodos principais

Método	O que faz
<code>Start()</code>	Inicia a conexão com o servidor (faz o handshake)
<code>Stop()</code>	Finaliza a conexão com o servidor
<code>SendCommand(string command)</code>	Envia um comando do protocolo ao servidor

4.3 Enviar comandos

Com a conexão estabelecida, envie os comandos com `SendCommand`. Cada comando é uma **string**, exatamente como descrito na documentação da API Socket:

```
connector.SendCommand("SQT PETR4"); // assina a cotação de PETR4  
connector.SendCommand("BQT PETR4"); // assina o livro de ofertas de PETR4
```

i O **livro de ofertas** (`BQT`) está disponível a partir do **Pacote PRO**. No Pacote Basic, use apenas os comandos listados na documentação da API Socket do seu plano.

💡 O lugar certo para enviar as assinaturas iniciais é o *callback* `OnConnect`: assim elas são reenviadas automaticamente sempre que a conexão for restabelecida.

4.4 Callbacks

Callback	Quando é invocado
<code>OnConnect</code>	Quando a conexão é estabelecida com sucesso



Callback	Quando é invocado
OnDisconnect	Quando a conexão é perdida
OnMessage	Sempre que uma nova mensagem é recebida

Exemplo de *callback* de mensagens tratando falha de login:

```
static void OnMessage(string message)
{
    if (message.Equals("invalid login.", StringComparison.OrdinalIgnoreCase))
    {
        Console.WriteLine("Login inválido. Verifique suas credenciais.");
    }
    else
    {
        Console.WriteLine($"Mensagem recebida: {message}");
    }
}
```

5. Exemplo completo

Exemplo com uma **fila** e um consumidor separado, reconexão automática e parada da reconexão quando o login é inválido:

```
using Cd3Connector;

namespace Example
{
    class Receiver
    {
        private readonly Queue<string> messages;
        private readonly ManualResetEventSlim eventSlim;
        private readonly Connector connector;
        private bool reconnect;
```

5



Av. Anselmo Alves dos Santos, 1111
Uberlândia-MG



www.cedrotech.com



contato@cedrotech.com



(34) 3239-0003

```
public Receiver(string user, string password)
{
    connector = new Connector(user, password, OnConnect, OnDisconnect, OnMessage);
    messages = new Queue<string>();
    eventSlim = new ManualResetEventSlim(false);
    ThreadPool.QueueUserWorkItem(state => ProcessMessages());
    reconnect = true;
}

public void Start()
{
    connector.Start();
}

private void OnConnect()
{
    connector.SendCommand("sqz petr4");
    connector.SendCommand("bqt petr4");
    connector.SendCommand("sqz winm24");
    connector.SendCommand("bqt winm24");
}

private void OnDisconnect()
{
    if (reconnect)
        connector.Start();
}

private void OnMessage(string message)
{
    lock (messages) // só enfileira – nunca processa aqui
    {
        messages.Enqueue(message);
        eventSlim.Set();
    }
}
```



```
}

private void ProcessMessages()
{
    while (true)
    {
        eventSlim.Wait();
        string msg;
        lock (messages)
        {
            msg = messages.Dequeue();
            if (messages.Count == 0)
                eventSlim.Reset();
        }
        if (msg.Equals("invalid login.", StringComparison.OrdinalIgnoreCase))
            reconnect = false;
        Console.WriteLine(msg);
    }
}

class Program
{
    static void Main(string[] args)
    {
        var receiver = new Receiver(
            Environment.GetEnvironmentVariable("CEDRO_USER"),
            Environment.GetEnvironmentVariable("CEDRO_PASSWORD"));
        receiver.Start();
    }
}
```



6. Boas práticas

6.1 Nunca processe dentro do `OnMessage`

O servidor **desconecta o consumidor lento**: se a sua aplicação não drenar as mensagens rápido o suficiente e a fila do socket acumular, a conexão é derrubada. Por isso o `OnMessage` deve apenas **enfileirar** a mensagem — o processamento acontece em outra thread, como no exemplo da seção 5.

6.2 Reconexão com critério

Reconectar no `OnDisconnect` é o padrão recomendado, **exceto** quando a mensagem recebida indica erro de credencial ou permissão — nesse caso, reconectar em laço não resolve e pode bloquear a conta:

Mensagem	Significado
<code>invalid login.</code>	Usuário ou senha inválidos
<code>software key not found.</code>	Chave de software não encontrada
<code>you don't have any permission for this software.</code>	Usuário sem permissão para este software

6.3 Uma conexão por usuário

As contas aceitam **pouquíssimas conexões simultâneas (na prática, ~1)**. Uma segunda conexão com o **mesmo usuário** derruba a primeira (erro `E:6`). Garanta que só exista **uma instância** do conector por usuário na sua aplicação — inclusive entre processos e servidores diferentes.

6.4 Tempo real ou delay

O dado pode chegar em **tempo real** ou em **delay** (tipicamente ~15 minutos), conforme a permissão da sua conta por mercado. Consulte com o comando `GUP PM` (documentação da API Socket, seção Usuário e sistema) — o modo efetivo de cada cotação aparece no **campo 96** do `SQT (RT / D / EOD)`.



6.5 Encerre a conexão ao terminar

Chame `Stop()` ao final da coleta. Nunca deixe uma conexão pendurada.

7. Próximos passos

Com a conexão funcionando, consulte a **documentação da API Socket** do seu plano para:

- A referência completa de comandos (`SQT` , `BQT` , `GQT` , `GCH` , ...) e o formato de cada mensagem.
- Os limites do seu pacote (ativos assinados simultaneamente, retenção de candles).
- A tabela de erros `E:` e os guias práticos (watchlist, fita de negócios, gráfico de candles).

