Statistical Database API SOAP Web Service

Technical Manual for Developers

banco central Chile

Table of Contents

1. Introduction

This manual aims to provide a detailed guide to integrating and consuming the **Central Bank of Chile BDE API** through its **SOAP Web Service**, one of the available methods to access the economic data available in its **Statistical Database (BDE).** This service provides access to a wide variety of economic indicators, such as data series with daily, monthly, quarterly, and annual frequencies, which are essential for automated financial and economic analysis. Through this service, developers can obtain relevant information efficiently and in real time, integrating it directly into their applications.

To begin using the service, developers must have an active account in the **Central Bank of Chile BDE**, using their credentials (username and password) to authenticate requests. If they do not have an account, they must access the BDE and register. If this is the first time using the BDE API in any of its variants (bcchapi library, REST, or SOAP), developers must go to the official BDE API portal, where under the "API Access" section they will find the "Enable API" option, which will allow them to authenticate and enable access by clicking the "Enable API" button. Once this step is completed, it will not be necessary to repeat it again.

To guide the use of the BDE API through its SOAP service, this manual presents a C# example in Visual Studio 2022, where a client will be generated and the SOAP API will be used via the available WSDL file (https://si3.bcentral.cl/SieteWS/sietews.asmx?wsdl). In this example, both .NET Framework and .NET 5/6/7/8 implementations will be covered, showing how to generate the client and how to interact with the SOAP service in each of these environments.

At the end of the manual, the complete source code will be made available so developers can replicate the implementation in their own environment. In addition, a PHP example will be included that reproduces the same functionality as the C# code, illustrating how to consume the SOAP Web Service in a web page, thus allowing you to see the differences between platforms and how to adapt to different development environments.

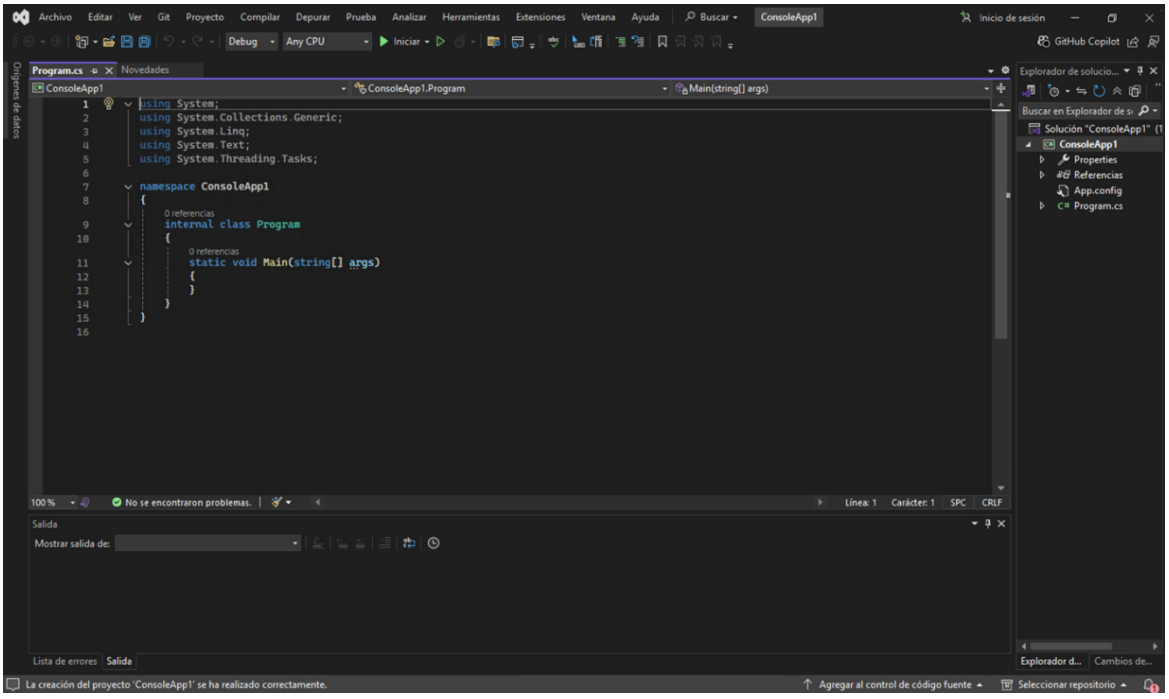
If you want to use the BDE API through another available service (bcchapi library in Python or the REST Web Service), consult the official BDE API portal to access the official documentation.

2. How to consume the SOAP Web Service?

In this section, it is explained how to generate the client and consume the **BDE API SOAP service** using **C#** in two environments: **.NET Framework and .NET 5/6/7/8.** In .NET Framework, you work with the traditional Web Services implementation through Add Web Reference, while in modern .NET (5/6/7/8), the process is adapted via WCF Connected Services, which generates a different client that uses asynchronous methods by default for greater efficiency in modern applications. Next, the steps to create the application, generate the client, run queries, and handle errors in Visual Studio 2022 are detailed.

2.1 Application Creation

To start consuming the BDE API SOAP service, the first step is to create an application in the .NET environment preferred by the developer. You can choose **.NET Framework** or a modern version such as **.NET 5, 6, 7, or 8**, depending on your project needs. Below, a reference .NET Framework console application is shown.

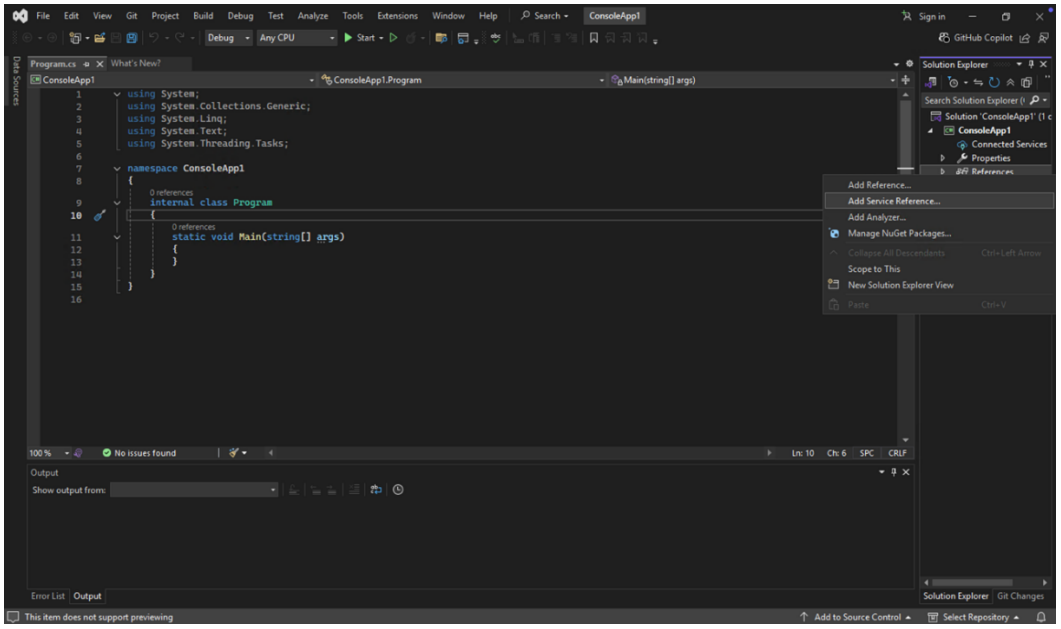


2.2 Generate a SOAP Client Referencing the Web Service

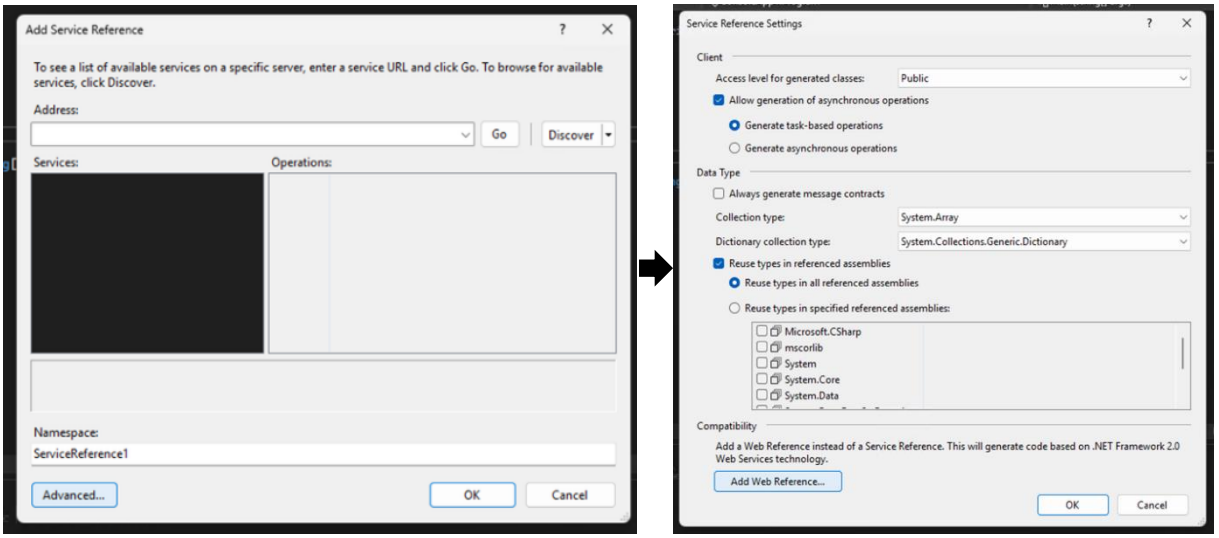
Once the application has been created in the desired .NET environment, the next step is to generate the client that will allow interaction with the BDE API SOAP service using the Web Service WSDL. This process varies depending on the type of application used, so this section presents the steps for both scenarios: **.NET Framework and .NET 5/6/7/8.**

2.2.1 .NET Framework: Web Service Reference – Compatibility Version

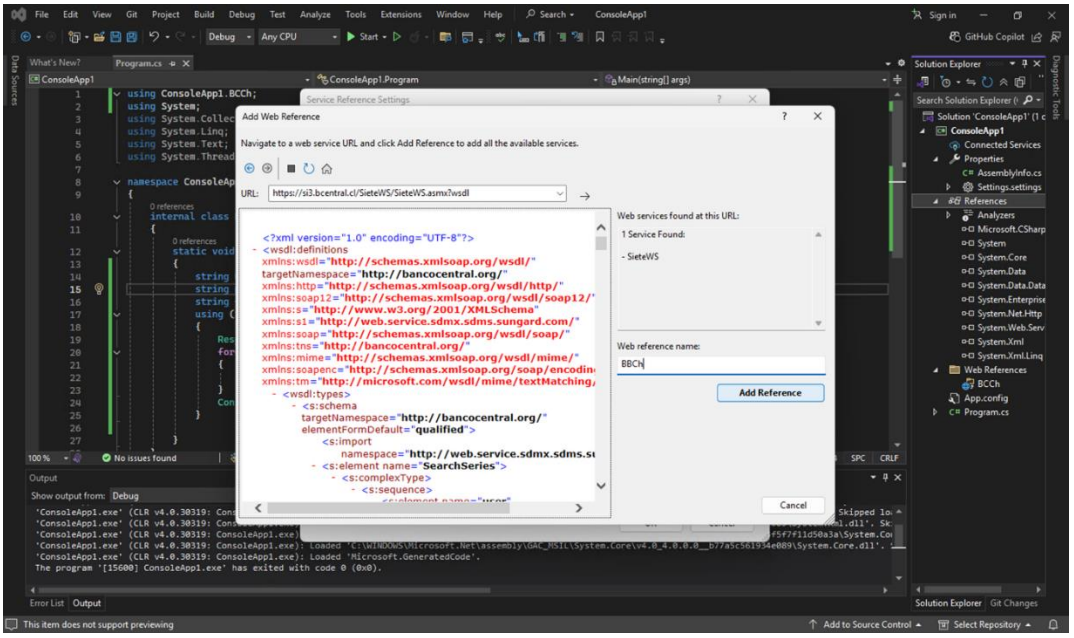
Step 1: In Solution Explorer, right-click References and select Add Service Reference.



Step 2: In the pop-up window, select Advanced → Compatibility → Add Web Reference.

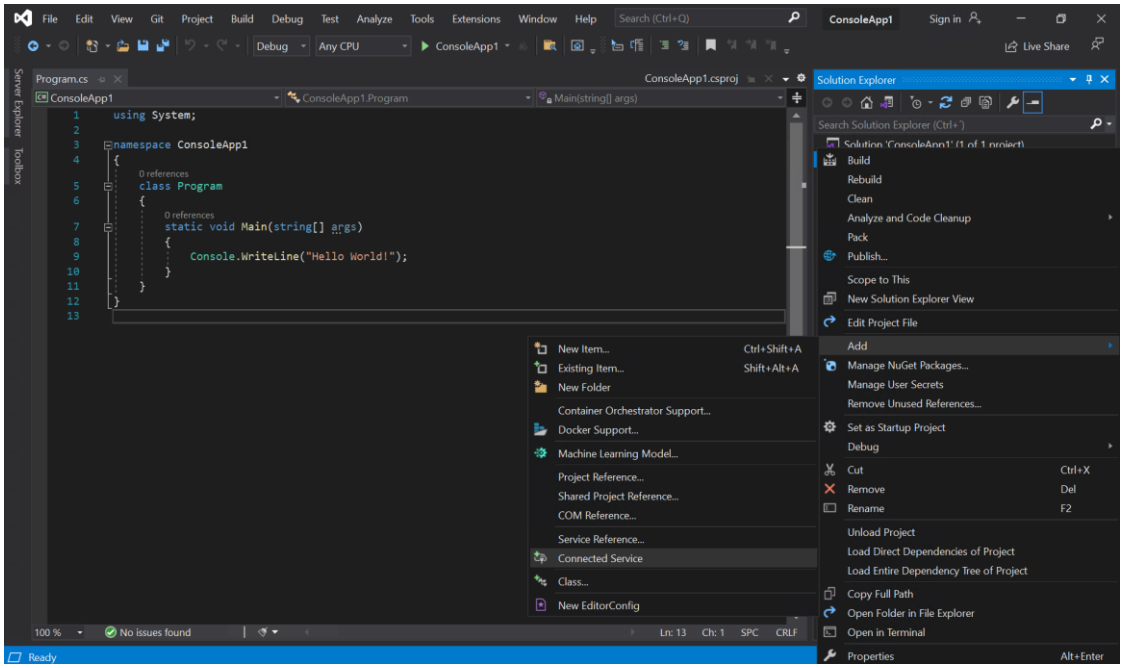


Step 3: In the URL field, enter the WSDL address. Then click the arrow to load the service and, once visible, define a name for the web service and click Add Reference to start using it.

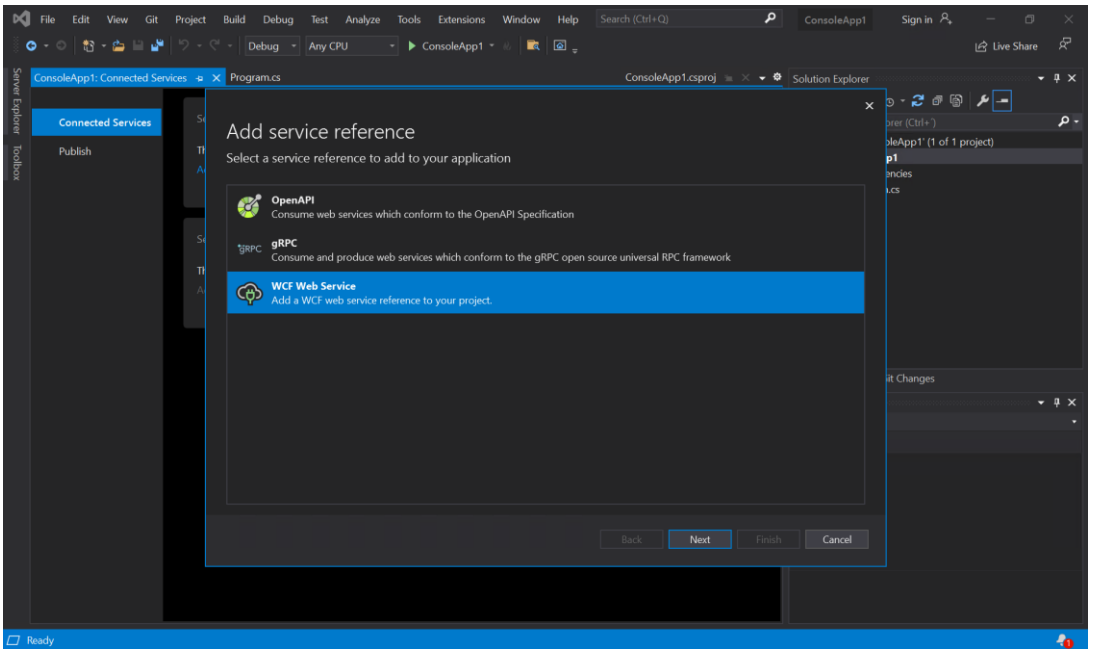


2.2.2 .NET 5/6/7/8: with WCF Connected Services

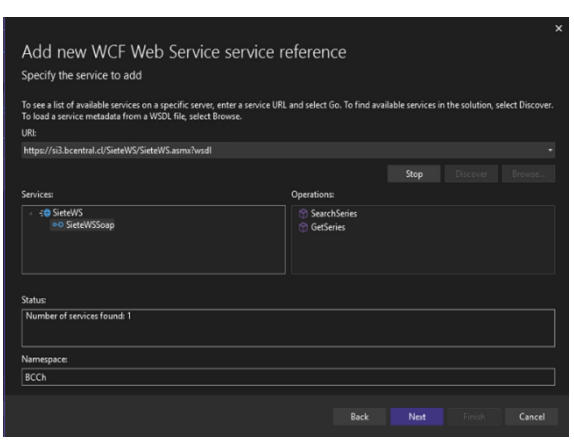
Step 1: In Solution Explorer, right-click the console project and select Add → Connected Services.

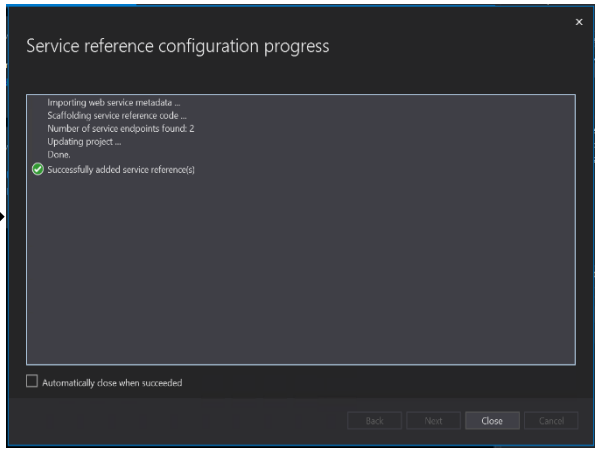


Step 2: In the Add service reference window, choose WCF Web Service.



Step 3: In the URL field, enter the WSDL address and click "Go" to identify the service. Once identified, assign a name to the web service under "Namespace". Then click Next and Finish to configure the service; once you see the success indicator, it is ready to use.





Once Visual Studio finishes adding the reference and creating the necessary classes, they can be used from the console application.

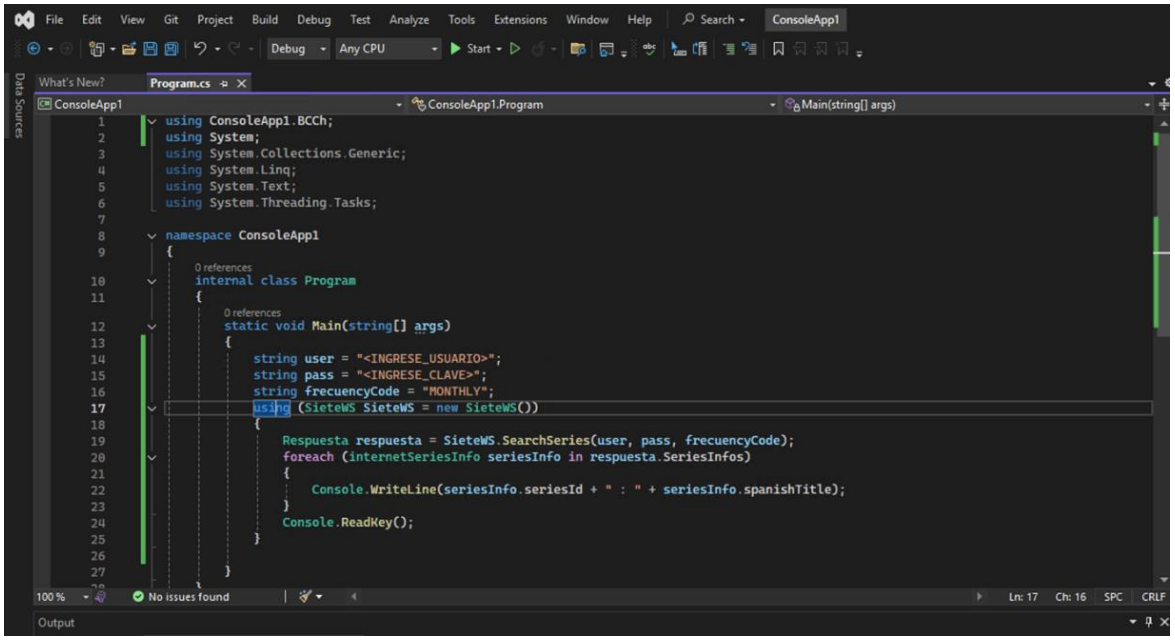
2.3 Buscar las series disponibles: Método SearchSeries

Once the client has been generated, you can invoke the available methods. To query the series available in the BDE API, the **SearchSeries** method is used, which receives three string parameters:

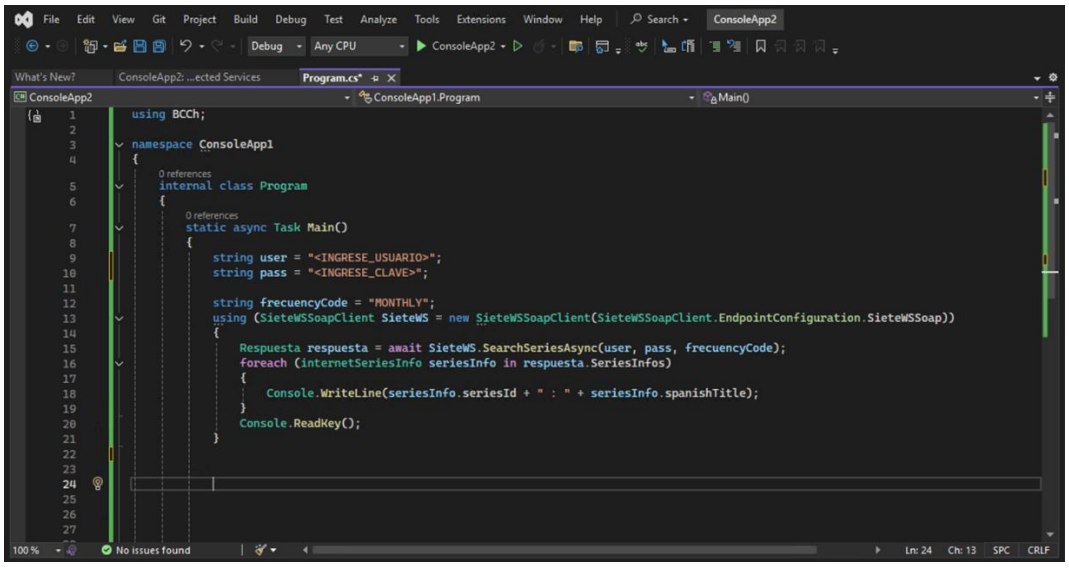
- **user:** email address registered in the BDE.
- **password:** password used to log into the BDE.
- **frecuencyCode**: filter to limit the results to series with a specific frequency. Valid options are **DAILY, MONTHLY, QUARTERLY and ANNUAL.**

The following images show a code example to retrieve the series that contain data with MONTHLY frequency, highlighting the code differences between .NET Framework and .NET 5/6/7/8.

In **.NET Framework**, the Main method is synchronous and the service call is performed directly; to instantiate the client you use: "SieteWS SieteWS = new SieteWS()" as shown in the image.



In contrast, in **modern .NET versions**, the Main method must be asynchronous and the service call uses await and methods ending in Async, while to instantiate the client you use: "SieteWSSoapClient SieteWS = new SieteWSSoapClient (SieteWSSoapClient.EndpointConfiguration.SieteWSSoap))".

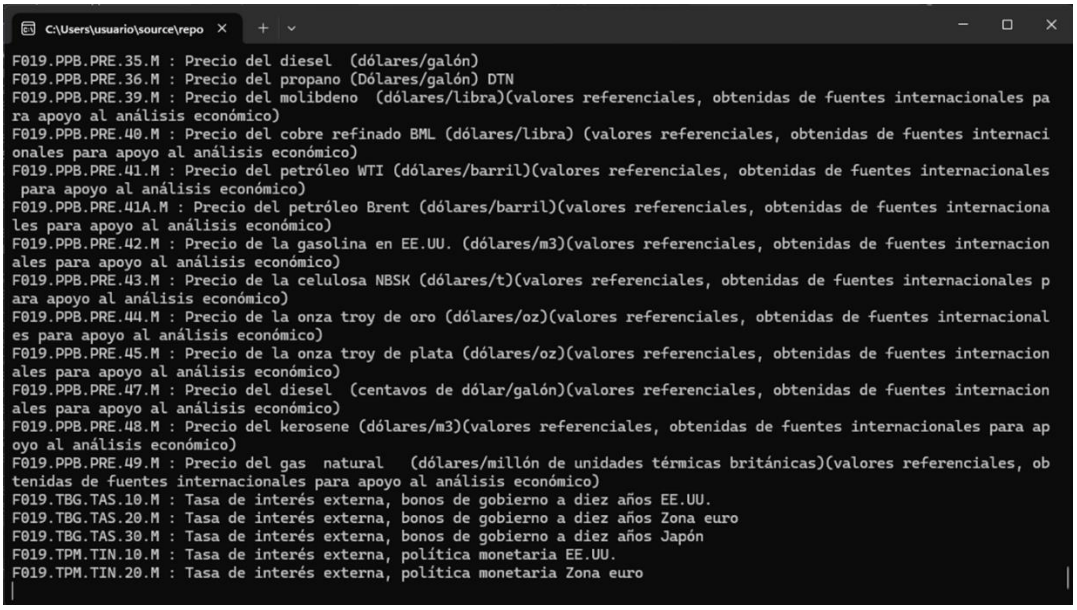


```
using BCCh;

namespace ConsoleApp1
{
    0 references
    internal class Program
    {
        0 references
        static async Task Main()
        {
            string user = "<INGRESE_USUARIO>";
            string pass = "<INGRESE_CLAVE>";

            string frecuencyCode = "MONTHLY";
            using (SieteWSSoapClient SieteWS = new SieteWSSoapClient(SieteWSSoapClient.EndpointConfiguration.SieteWSSoap))
            {
                Respuesta respuesta = await SieteWS.SearchSeriesAsync(user, pass, frecuencyCode);
                foreach (internetSeriesInfo seriesInfo in respuesta.SeriesInfos)
                {
                    Console.WriteLine(seriesInfo.seriesId + " : " + seriesInfo.spanishTitle);
                }
                Console.ReadKey();
            }
```

Running either of these code samples displays the series list shown below, where the series ID and the Spanish description are displayed.



```
C:\Users\usuario\source\repo
F019.PPB.PRE.35.M : Precio del diesel  (dólares/galón)
F019.PPB.PRE.36.M : Precio del propano (Dólares/galón) DTN
F019.PPB.PRE.39.M : Precio del molibdeno  (dólares/libra)(valores referenciales, obtenidas de fuentes internacionales pa
ra apoyo al análisis económico)
F019.PPB.PRE.40.M : Precio del cobre refinado BML (dólares/libra) (valores referenciales, obtenidas de fuentes internaci
onales para apoyo al análisis económico)
F019.PPB.PRE.41.M : Precio del petróleo WTI (dólares/barril)(valores referenciales, obtenidas de fuentes internacionales
 para apoyo al análisis económico)
F019.PPB.PRE.41A.M : Precio del petróleo Brent (dólares/barril)(valores referenciales, obtenidas de fuentes internaciona
les para apoyo al análisis económico)
F019.PPB.PRE.42.M : Precio de la gasolina en EE.UU. (dólares/m3)(valores referenciales, obtenidas de fuentes internacion
ales para apoyo al análisis económico)
F019.PPB.PRE.43.M : Precio de la celulosa NBSK (dólares/t)(valores referenciales, obtenidas de fuentes internacionales p
ara apoyo al análisis económico)
F019.PPB.PRE.44.M : Precio de la onza troy de oro (dólares/oz)(valores referenciales, obtenidas de fuentes internacional
es para apoyo al análisis económico)
F019.PPB.PRE.45.M : Precio de la onza troy de plata (dólares/oz)(valores referenciales, obtenidas de fuentes internacion
ales para apoyo al análisis económico)
F019.PPB.PRE.47.M : Precio del diesel  (centavos de dólar/galón)(valores referenciales, obtenidas de fuentes internacion
ales para apoyo al análisis económico)
F019.PPB.PRE.48.M : Precio del kerosene (dólares/m3)(valores referenciales, obtenidas de fuentes internacionales para ap
oyo al análisis económico)
F019.PPB.PRE.49.M : Precio del gas  natural  (dólares/millón de unidades térmicas británicas)(valores referenciales, ob
tenidas de fuentes internacionales para apoyo al análisis económico)
F019.TBG.TAS.10.M : Tasa de interés externa, bonos de gobierno a diez años EE.UU.
F019.TBG.TAS.20.M : Tasa de interés externa, bonos de gobierno a diez años Zona euro
F019.TBG.TAS.30.M : Tasa de interés externa, bonos de gobierno a diez años Japón
F019.TPM.TIN.10.M : Tasa de interés externa, política monetaria EE.UU.
F019.TPM.TIN.20.M : Tasa de interés externa, política monetaria Zona euro
```

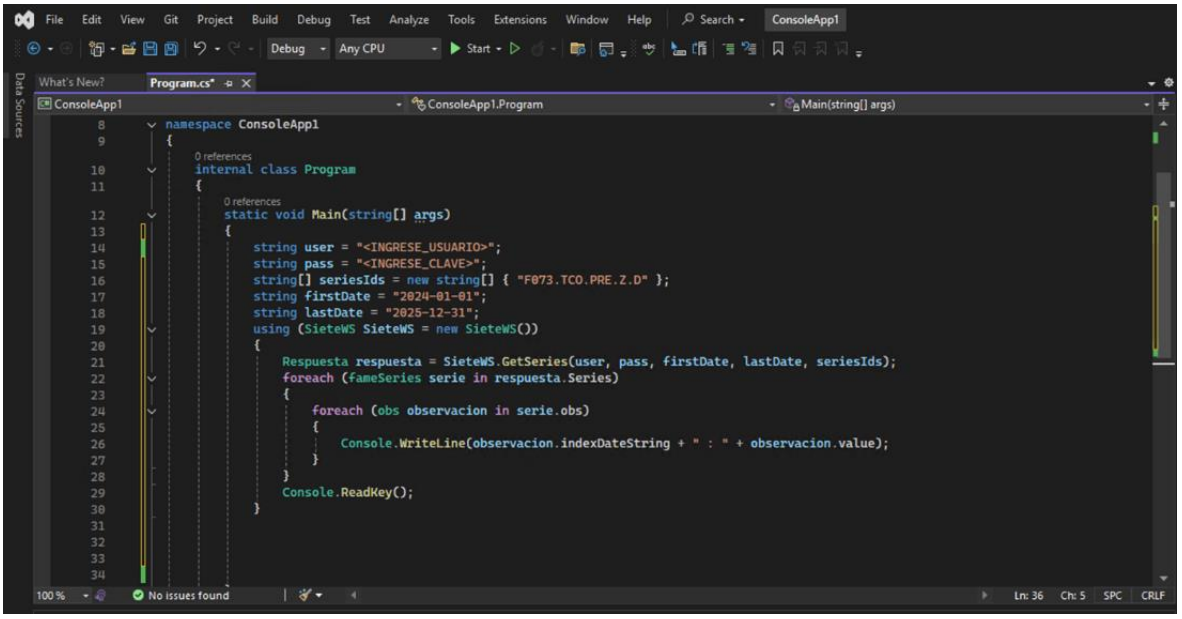
2.4 Retrieve Observations for a Series: GetSeries Method

To retrieve observations for a series, there is a method called **GetSeries**, which receives five string parameters:

- **user:** email address registered in the BDE.
- **password:** password used to log into the BDE.
- **firstDate (opcional):** filter to constrain observations from a start date. Format must be YYYY-MM-DD.
- **lastDate (opcional):** filter to constrain observations up to an end date. Format must be YYYY-MM-DD.
- **seriesIds:** filter to indicate the series code for which you want to retrieve observations. Although this parameter is an array, **only one series can be queried per method call.**

For example, if you want to retrieve the observations of the USD exchange rate series, you must use its ID, which is "F073.TCO.PRE.Z.D". If you want all observations for years 2024 and 2025, you must call the web service as shown in the code below.

In the **.NET Framework** case, the same synchronous invocation characteristics explained in the previous section are maintained.



```
namespace ConsoleApp1
{
    0 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
            string user = "<INGRESE_USUARIO>";
            string pass = "<INGRESE_CLAVE>";
            string[] seriesIds = new string[] { "F073.TCO.PRE.Z.D" };
            string firstDate = "2024-01-01";
            string lastDate = "2025-12-31";
            using (SieteWS SieteWS = new SieteWS())
            {
                Respuesta respuesta = SieteWS.GetSeries(user, pass, firstDate, lastDate, seriesIds);
                foreach (fameSeries serie in respuesta.Series)
                {
                    foreach (obs observacion in serie.obs)
                    {
                        Console.WriteLine(observacion.indexDateString + " : " + observacion.value);
                    }
                }
                Console.ReadKey();
            }
```

On the other hand, in **modern .NET**, the invocation uses asynchronous methods.

```
using BCCh;

namespace ConsoleApp1
{
    internal class Program
    {
        static async Task Main()
        {
            string user = "<INGRESE_USUARIO>";
            string pass = "<INGRESE_CLAVE>";

            string[] seriesIds = new string[] { "F073.TCO.PRE.Z.D" };
            string firstDate = "2024-01-01";
            string lastDate = "2025-12-31";
            using (SieteWSSoapClient SieteWS = new SieteWSSoapClient(SieteWSSoapClient.EndpointConfiguration.SieteWSSoap))
            {
                Respuesta respuesta = await SieteWS.GetSeriesAsync(user, pass, firstDate, lastDate, seriesIds);
                foreach (fameSeries serie in respuesta.Series)
                {
                    foreach (obs observacion in serie.obs)
                    {
                        Console.WriteLine(observacion.indexDateString + " : " + observacion.value);
                    }
                }
                Console.ReadKey();
            }
```

In the code, you can see that it is necessary to run a foreach loop inside another foreach loop. The first loop selects the requested series and the second loop iterates through the observations of that series.

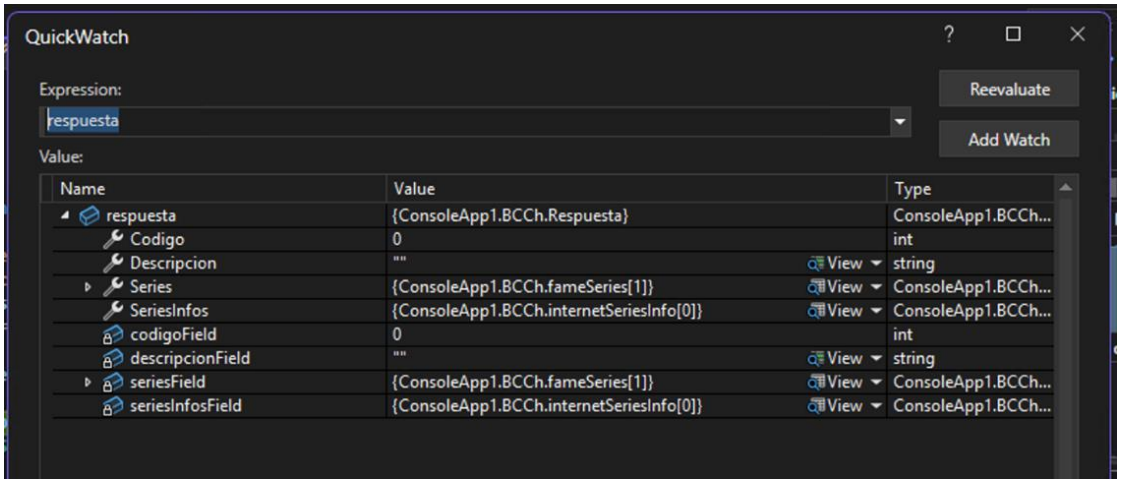
Running this code produces the output shown below, highlighting the observation date, one per day because it is a daily series (DAILY), and the observation value.

```
C:\Users\usuario\source\repo
06-10-2025 : 963,17
07-10-2025 : 963,09
08-10-2025 : 961,61
09-10-2025 : 951,45
10-10-2025 : 948,77
11-10-2025 : NaN
12-10-2025 : NaN
13-10-2025 : 954,37
14-10-2025 : 957,66
15-10-2025 : 963,69
16-10-2025 : 959,36
17-10-2025 : 956,13
18-10-2025 : NaN
19-10-2025 : NaN
20-10-2025 : 960,34
21-10-2025 : 953,21
22-10-2025 : 951,78
23-10-2025 : 950,46
24-10-2025 : 945,34
25-10-2025 : NaN
26-10-2025 : NaN
27-10-2025 : 943,47
28-10-2025 : 938,19
29-10-2025 : 941,63
30-10-2025 : 940,57
31-10-2025 : NaN
01-11-2025 : NaN
02-11-2025 : NaN
03-11-2025 : 943,69
```

2.5 Error Handling

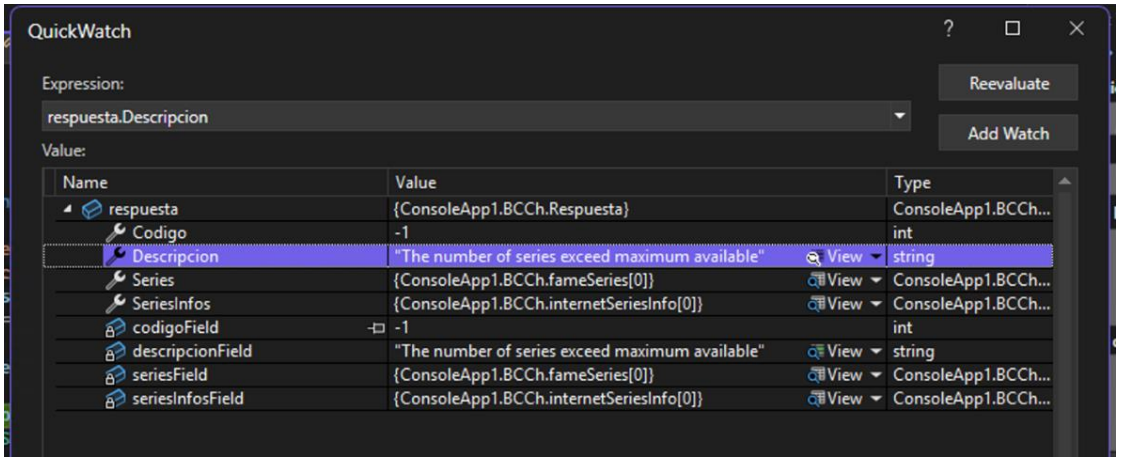
Next, it is shown how to inspect the returned code in case an error occurs when invoking any of the methods shown previously. To check whether the operation succeeded, you must inspect the **"Codigo" and "Descripcion"** attributes of the **"Respuesta" object** that stores the result returned by the methods.

In case of success, the Respuesta "Codigo" will be 0 and "Descripcion" will be empty, indicating that the method returned an expected response. The following image shows the Respuesta object attributes in this case.

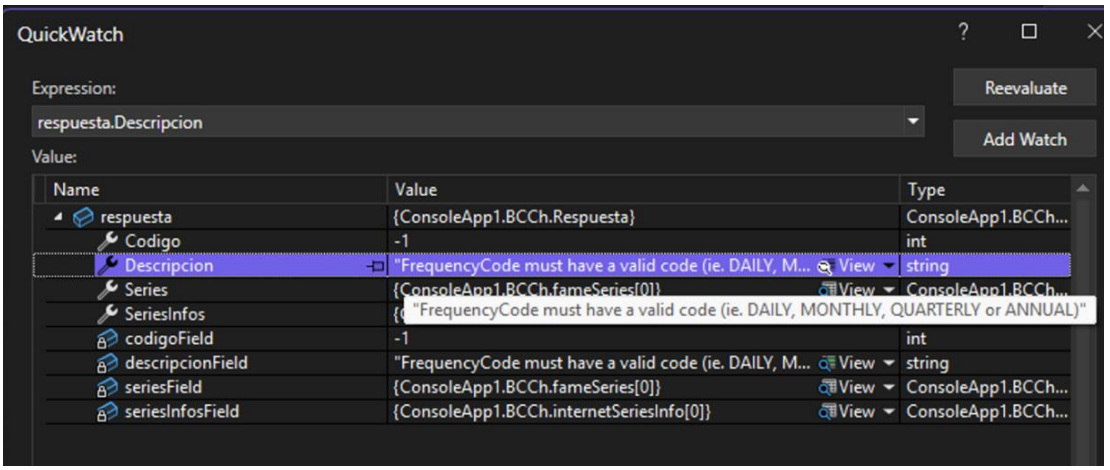


In case of error, "Codigo" will take values different from 0 depending on the issue, and "Descripcion" will detail the cause of the error. Below are examples of how the Respuesta object attributes look for certain frequent errors.

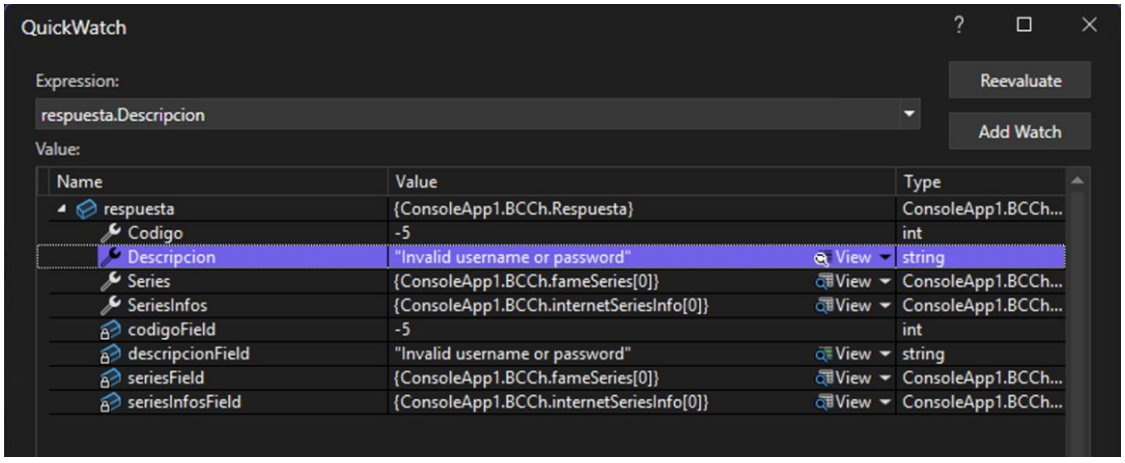
1. **Querying more than one series at a time:** as mentioned in the previous section, when using the GetSeries method you can query at most one series. If you add more than one series code to the seriesIds array, "Codigo" will be -1 and "Descripcion" will say: "The number of series exceed máximum available".



2. **Entering an invalid frequency:** the SearchSeries method only admits the frequencies DAILY, MONTHLY, QUARTERLY and ANNUAL. If you enter something different in frecuencyCode, "Codigo" will be -1 and "Descripcion" will say: "FrecuencyCode must have a valid code".



3. **Access credential error:** if you enter an incorrect username or password, "Codigo" will be -5 and "Descripcion" will say: "Invalid username or password".



2.6 Official Documentation and Support

To explore further the use of the Central Bank of Chile BDE API SOAP service, as well as other available services, the Bank provides official documentation on its portal:

🔗 **Official BDE API Portal**

This portal includes practical examples for the three available services:

1) **bcchapi Python library:** allows querying indicators such as the observed USD exchange rate, IMACEC, and GDP, using the library's specific functions.
2) **REST Web Service:** shows how to access the same indicators via REST URLs. It includes consumption examples in R, useful to enrich statistical analyses.
3) **SOAP Web Service:** presents the same examples implemented as SOAP clients for different C# development environments, with results displayed in the console.

The BDE API portal includes a **Help and Support** section, where you can find frequently asked questions, common errors, and their solutions. This allows users to resolve questions without requiring direct contact.

If you require personalized assistance, the Central Bank provides a **centralized contact center.** To access it, visit:

🔗 **https://contactocentral.bcentral.cl/**

When submitting a request, it is recommended to include the following information to speed up the response:

- Detailed description of the error or issue.
- Development environment used (Python, R, C#, etc.).
- Access method (bcchapi library, REST, or SOAP).
- Relevant source code snippet.
- Series codes being queried.
- Exact date and time when the issue occurred.

3. Appendix: Source Code

This section provides complete code examples to consume the BDE API SOAP service in .NET Framework and .NET 5/6/7/8 environments using C#. Additionally, a PHP code sample is provided that reproduces the same functionality as the C# examples, so developers can compare how to consume the SOAP service in a web page.

3.1 C# Code - .NET Framework

```
using ConsoleApp1.BCCh;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApp1 {
    internal class Program {
        static void Main(string[] args) {
            string user = "<ENTER_USERNAME>";
            string pass = "<ENTER_PASSWORD>";
            string frecuencyCode = "MONTHLY";
            using (SieteWS SieteWS = new SieteWS()) {
                Respuesta respuesta = SieteWS.SearchSeries(user, pass, frecuencyCode);
                foreach (internetSeriesInfo seriesInfo in respuesta.SeriesInfos) {
                    Console.WriteLine(seriesInfo.seriesId + " : " + seriesInfo.spanishTitle);
                }
                Console.ReadKey();
            }
            string[] seriesIds = new string[] { "F073.TCO.PRE.Z.D "};
            string firstDate = "2024-01-01";
            string lastDate = "2025-12-31";
            using (SieteWS SieteWS = new SieteWS()) {
                Respuesta respuesta = SieteWS.GetSeries(user, pass, firstDate, lastDate, seriesIds);
                foreach (fameSeries serie in respuesta.Series) {
                    foreach (obs observacion in serie.obs) {
                        Console.WriteLine(observacion.indexDateString + " : " + observacion.value);
                    }
                }
                Console.ReadKey();
            }
        }
    }
}
```

3.2 C# Code - .NET 5/6/7/8

```
using BCCh

namespace ConsoleApp1 {

    internal class Program {

        static async Task Main() {

            string user = "<ENTER_USERNAME>";

            string pass = "<ENTER_PASSWORD>";

            string frecuencyCode = "MONTHLY";

            using (SieteWSSoapClient SieteWS = new
SieteWSSoapClient(SieteWSSoapClient.EndpointConfiguration.SieteWSSoap)) {

                Respuesta respuesta = await SieteWS.SearchSeriesAsync(user, pass, frecuencyCode);

                foreach (internetSeriesInfo seriesInfo in respuesta.SeriesInfos) {

                    Console.WriteLine(seriesInfo.seriesId + " : " + seriesInfo.spanishTitle);

                }

                Console.ReadKey();

            }

            string[] seriesIds = new string[] { "F073.TCO.PRE.Z.D" };

            string firstDate = "2024-01-01";

            string lastDate = "2025-12-31";

            using (SieteWSSoapClient SieteWS = new
SieteWSSoapClient(SieteWSSoapClient.EndpointConfiguration.SieteWSSoap)) {

                Respuesta respuesta = await SieteWS.GetSeriesAsync(user, pass, firstDate, lastDate,
seriesIds);

                foreach (fameSeries serie in respuesta.Series) {

                    foreach (obs observacion in serie.obs) {

                        Console.WriteLine(observacion.indexDateString + " : " + observacion.value);

                    }

                }

                Console.ReadKey();

            }

        }

    }

}
```

3.3 Extra: Código en PHP

```php
<?php
$user = '<ENTER_USERNAME>';
$password='<ENTER_PASSWORD>';
$frequencyCode ='MONTHLY';
$wsdl="<WEBSITE_ADDRESS>?WSDL";
$client = new SoapClient($wsdl);
$params = new stdClass;
$params->user = $user;
$params->password = $password;
$params->frequencyCode = $frequencyCode;
$result = $client->SearchSeries($params)->SearchSeriesResult;
foreach ($result->SeriesInfos->internetSeriesInfo as $serie) {
     echo $serie->seriesId . " : " . $serie->spanishTitle . "<br/>";
}

$seriesIds = array ("F073.TCO.PRE.Z.D");
$firstDate = "2024-01-01";
$lastDate = "2025-12-31";
$client =lient($wsdl);
$params = new stdClass;
$params->user = $user;
$params->password = $password;
$params->firstDate = $firstDate;
$params->lastDate = $lastDate;
$params->seriesIds = $seriesIds;
$result = $client->GetSeries($params)->GetSeriesResult;
$fameSeries =$result->Series->fameSeries;

// When requesting a single series, the response is not interpreted as an array
if (!is_array($fameSeries)) $fameSeries = array($fameSeries);
foreach ($fameSeries as $serieFame) {
     echo $serieFame->seriesKey->seriesId . " : " . $serieFame->precision . "<br/>";
     foreach ($serieFame->obs as $obs) {
              echo $obs->indexDateString . " : " . $obs->value . "<br/>";
     }
}
?>php
```