
# InstantRemoting

## .NET Programmer's Guide

Version 1.1

# Table of Contents

# The SkyBridge® technology suite

The SkyBridge® suite contains two main products:

## SkyBridge® Instant Remoting

SkyBridge® Instant Remoting is the simplest and yet most powerful instant remoting technology.

It is the simplest because only **three lines of code** are needed on each computer against our .NET class library, which can be downloaded from nuget. Both computers establish outgoing TCP connections to port 52888 of one of the SkyBridge® servers, which acts as the middlemen to facilitate the data exchange.

It is the most powerful because it works purely on OSI model layer 4 - transportation layer. It allows any two computers in the world to instantly exchange arbitrary byte arrays. There can be infinite types of applications. An invoke-response round trip can take as little as 0.1 second.

## SkyBridge® ProxyDLL

SkyBridge® ProxyDLL allows any normal .NET DLL (class library) to be invoked remotely by a .NET software program from anywhere in the world.

Using the SkyBridge® ProxyDLL Windows app, in just a few mouse clicks, a proxy DLL is generated from the original DLL, which has exactly the same public interface. This proxy is then supplied to the remote software program, which teats this proxy as if it is the original and calls its functions.

When invoked, each function in the proxy DLL uses SkyBridge® InstantRemoting to route the invocation to the original DLL to invoke the eponymous function. The response is then routed back to the proxy function and returned to its caller.

This way, the remote software doesn't know that this proxy DLL is any different from the original DLL, because its behavior is the same as the original.

### Use case 1 – Protecting IP while working with remote software developers

A software house is developing an application, which has a user interface (UI) and an algorithm class library. They want to hire a remote freelancer to design the UI, but they can't provide the algorithm DLL to the freelancer because it contains critically important intellectual property. So, they generate a proxy of the algorithm DLL and supply it to the freelancer. The freelancer can then develop and test the UI code which invokes the proxy DLL.

### Use case 2 – Offering limited access to protected resources

A business has a confidential corporate database. They want to expose a handful of its tables to an external business partner in a read-only manner. So they write a DLL that has a few functions that query these tables in a read-only manner. Then they generate a proxy DLL and give it to the external partner, who then writes software to invoke this proxy DLL. **Their programmers don't need to know any networking.**

### User case 3 – remote control

An instrument manufacturer needs to remotely access their instruments in the field in order to for example turn on/off certain licensed features or download log files. So they write a DLL and place it in the instrument which turns the features on/off and get log files. Then they generate a proxy DLL and write an *instrument remote control* user interface which calls functions on the proxy DLL. **Their programmers don't need to know any networking.**

# Free and anonymous use

If the usage of SkyBridge® Instant Remoting is casual, and the amount of data sent is small, then you can use SkyBridge® Instant Remoting for free permanently, without ever going to the SkyBridge web portal. Just download the SDK from nuget, write a few lines to invoke it, and execute.

The number of connections, the total connection time and the total data sent from every IP address in a rolling window of the last 24 hours is gauged by the SkyBridge® servers. When any of the three figures exceeds the free quota (something like 2 connections, 2 hours and 1 MB per IP address), exceptions will be thrown. Then, you either have to register a paid API account, or wait until some of your usages fall out of the rolling window.

# Terminologies

Using the SkyBridge® Instant Remoting SDK, one computer sends an arbitrary byte array, which is an **invocation**, to another computer. The receiver performs a service for the sender by processing the byte array and returning a byte array in response. The receiver is a **service**, and the sender is a **client**.

When a service connects to the SkyBridge® server, it supplies a **service identifier**. From now on, it is known to its clients by this identifier.

When a client wants to invoke this service, it sends an invocation byte array to the server with this identifier. The SkyBridge® server identifies the service using that identifier and forwards the invocation to the service.

There are two types of service identifiers:

**Service name**:

- It is an arbitrary string of 40 characters or shorter.
- When used in a paid API account, it is only visible within the account. This means that different accounts can use the same service name without causing conflicts.
- When used in a free and anonymous condition, it is visible globally to all other free and anonymous users. In this case, it is recommended that a globally unique identifier is used to make sure that other free and anonymous users won't accidentally invoke your service and vice versa.
- If a service tries to connect to the server with a service name, but another service is currently connected using this service name, this connection will fail with an exception.

**Service type ID**:

- It is an 8-byte integer allocated by the SkyBridge® web portal within a paid API account.
- A free and anonymous user cannot use service type ID.
- It is only visible within the paid account.

- Multiple services can connect to the servers using the same service type ID. In this case, when a client tries to invoke the service by this service type ID, the server randomly picks one of the services to serve the request. This way, multiple computers can run the same service to share the load if there are too many invocations to be handled by a single computer. If there is no need to have multiple computers running the same service, then it is recommended to use service name because it does not need prior work on the web portal and it is more flexible.

# The SDK for .NET framework and .NET core

The SkyBridge® Instant Remoting SDK is contained in a .NET class library which are available from NuGet. It has a .NET Framework and a .NET Core version. Later, SDKs for other languages and platforms will be developed.

# Class InstantRemotingClient

To create a client to invoke other services, create an instance of `InstantRemotingClient`. After its creation, or after an invocation on another service is made, if there is no more invocation, it disconnects.

One `InstantRemotingClient` instance can be used to invoke multiple services multiple times.

## The constructors

```
public InstantRemotingClient()
```

*[Description]*

Create a client used to invoke other services. No account ID and API key is supplied, therefore it is using the SkyBridge service freely and anonymously. There are limits on data and connection time.

```
public InstantRemotingClient(long accountId, string accountPassword)
```

*[Description]*

Create a client used to invoke other services.

*[Parameters]*

| Parameter | Description |
|---|---|
| accountId | The API account ID of your paid account. |
| apiKey | The API key of your paid account. Must be 64 characters long or less. It is allocated by the SkyBridge web portal when you register a paid account which has no limit on data and connection time. |

## Service-invoking methods that expect a response

There are two ways to invoke a service. One is to send a byte array to the service and block and wait for the response byte array (even if it is null). Consequently, the `ServiceInvokedEventArgs` passed to the `ServiceInvoked` event handler of the targeted `InstantRemotingService` will have the `ClientExpectingResponse` property being true.

When this method returns without an exception, then we know for sure that the invocation data has been received by the targeted service and a response has been received – even if it is null.

There are two methods that allow invoking services in this manner.

```
public byte[] InvokeServiceByName(
    string serviceName,
    byte[] data,
    int retrySendingForSeconds = 0,
    CancellationToken? cancellationToken = null)
```

*[Description]*

Invoke a service by its service name and wait for it to send back a response byte array even if it is null.

*[Parameters]*

| Parameter | Description |
|---|---|
| serviceName | The service name of the service to invoke. Must be 40 characters long or less. |
| data | The invocation data to send to the service. |
| retrySendingForSeconds | If this parameter is 0 and the invocation fails, an exception is thrown immediately.<br>If this parameter is greater than 0, if the invocation fails, then, within the given number of seconds, this method will keep retrying until the given time elapses, then an exception will be thrown. |
| cancellationToken | If you want this method to exit before the number of seconds specified by parameter "retrySendingForSeconds" elapses, then you shall pass a CancellationToken via this parameter, then, when a cancellation is triggered, an Exception will be thrown. |

*[Return]*

The response data sent back by the service.

```
public byte[] InvokeServiceByType(
    long serviceTypeId,
    byte[] data,
    int retrySendingForSeconds = 0,
    CancellationToken? cancellationToken = null)
```

*[Description]*

Invoke a service by its service type ID and wait for it to send back a response byte array even if it is null.

This method can only be called if this `InstantRemotingClient` instance was created with a paid API account. If this method is called on an instance that was created in a free and anonymous condition, an exception will be thrown.

*[Parameters]*

| Parameter | Description |
|---|---|
| serviceTypeId | The service type ID of the service to invoke. |

All other parameters are the same as those of `InvokeServiceByName`.

*[Return]*

The response data sent back by the service.

## Service-invoking methods that do not expect a response

The other way to invoke a service is to send a byte array to the service and do not expect a response. Consequently, the `ServiceInvokedEventArgs` passed to the `ServiceInvoked` event handler of the `InstantRemotingService` instance will have the `ClientExpectingResponse` property being false. These methods are faster to return than those that expect a round-trip response from the service. However, there is no guarantee that every invocation will reach the service, especially the first one or two, and the client will not get any notification if some have gone missing.

If this method in this category is called too frequently with too much data, and the data can't reach the service fast enough, or the service can't fetch data from the socket fast enough, then the data will accumulate in the buffers of the underlying operating system. To avoid such accumulation, a response-expecting method can be called once after the no-response method is called five or ten times. The response-expecting method call will block until all previous data in the pipeline has been received and processed by the service.

There are two methods that can be used to invoke services without waiting for response.

```
public void SendToServiceByName(
    string serviceName,
    byte[] data,
    int retrySendingForSeconds = 0,
    CancellationToken? cancellationToken = null)
```

*[Description]*

Send a byte array to a service by its service name and do not expect or wait for a response.

*[Parameters]*

The same as the parameters of **`Error! Reference source not found.`**.

```
public void SendToServiceByType(
    long serviceTypeId,
    byte[] data,
    int retrySendingForSeconds = 0,
    CancellationToken? cancellationToken = null)
```

*[Description]*

Send a byte array to a service by its service type ID and do not expect or wait for a response.

This method can only be called if this `InstantRemotingClient` was created with a paid account. If this method is called on an instance that was created in a free and anonymous condition, an exception will be thrown.

*[Parameters]*

The same as the parameters of method **Error! Reference source not found.**.

## The Terminate method

```
public void Terminate()
```

Once this method is called, this `InstantRemotingClient` instance is terminally disposed and cannot be used again.

## The GetOkBytes method

```
public static byte[] GetOkBytes();
```

*[Description]*

Create a two-byte array which can be converted by UTF8 encoding to "OK".

In your method which is invoked by the remote invoker, if you have nothing to send back to the invoker, you should call this method to generate the two-byte array and send that back.

## The IsOkBytes method

```
public static bool IsOkBytes(byte[] bytes);
```

*[Description]*

Check if an array is the two-byte "OK" array, which can be acquired by calling method `GetOkBytes`.

# Class InstantRemotingService

To create a service, create an `InstantRemotingService` Instance. An `InstantRemotingService` instance is very light weighted. The only resource it uses is the TCP socket connection.

A service can stay connected until its Terminate method is called, or it can stay disconnected and keep polling for demands. This behavior is controlled by a parameter of the constructors. When it is meant to stay connected, if external disturbances such as temporary server failure causes disconnection, it will keep trying to reconnect.

A `InstantRemotingService` has the same set of four service-invoking methods as a `InstantRemotingClient`, which can be called to invoke other services.

## The two constructors

```
public InstantRemotingService(
    string serviceName,
    EventHandler<ServiceInvokedEventArgs> serviceInvokedHandler,
    long? accountId = null,
    string apiKey = null,
    ushort pollForDemandIntervalInSecs = 0,
    ushort disconnectAfterIdleForSecs = 300)
```

*[Description]*

Create a service known to the clients by the specified service name.

*[Parameters]*

| Parameter | Description |
|---|---|
| serviceName | The service name that this service is known to the clients. Must be **40** characters long or less. |
| serviceInvokedHandler | This event handler is called when the service receives an invocation. The code in this handler defines what service is provided to the clients. |
| accountId | The API account ID of your paid account. Null if you want to call it as a free and anonymous user. |
| apiKey | The API key of your API account. Null if you want to call this method as a free and anonymous user. Must be 64 characters long or less. |
| pollForDemandIntervalInSecs | If this is zero, then this service remains connected until the `Terminate` method is called. If it is not zero, then it connects to check for demands of its service at an interval specified by this parameter. There is a minimum interval, something like 15 seconds, which is decided by the server. If this parameter specifies a value smaller than the minimum value, then the minimum value is used. Only a premium API account can have this parameter being greater than zero. |
| disconnectAfterIdleForSecs | This parameter is ignored if parameter pollForDemandIntervalInSecs is zero. If that parameter is greater than zero, then, when this service has been idle for |

| | |
|---|---|
| | longer than this parameter and there is no demand for this service, it disconnects. Then it will reconnect to check for demands at the interval specified by pollForDemandIntervalInSecs.<br>There is a minimum value for this parameter, something like 15 seconds. If the value passed in is greater than the minimum value, the minimum value is used. |

```
public InstantRemotingService(
    long serviceTypeId,
    EventHandler<ServiceInvokedEventArgs> serviceInvokedHandler,
    long? accountId = null,
    string apiKey = null,
    ushort pollForDemandIntervalInSecs = 0,
    ushort disconnectAfterIdleForSecs = 300)
```

*[Description]*

Create a service known to the clients by the specified service type ID.

*[Parameters]*

| Parameter | Description |
|---|---|
| serviceTypeId | The service type ID that this service is known to the clients. |

All other parameters are the same as those of the other constructor.

## Service-invoking methods

A `InstantRemotingService` has the same set of service-invoking methods as a `InstantRemotingClient`, which can be called to invoke other services.

## The events

```
public class ServiceInvokedEventArgs : EventArgs
{
    public long? TargetingServiceTypeId { get; set; }
    public string TargetingServiceName { get; set; }
    public byte [] InvocationData { get; set; }
    public bool ClientExpectingResponse { get; set; }
    public byte[] ResponseData { get; set; }
}
```

*[Description]*

This event is fired when an invocation is received from a client. You cannot use the "+=" operator to subscribe to this event. The event handler must be passed in as the second parameter of the constructor.

*[Event properties]*

| Parameter | Description |
|---|---|
| TargetingServiceTypeId | What the client is targeting. Should match the service type ID of this service. |
| TargetingServiceName | What the client is targeting. Should match the service name of this service. 40 characters long or less. |
| InvocationData | The invocation data sent by the client. |
| ClientExpectingResponse | Whether the client is expecting a response. If this flag is false, if possible, this method should not spend time and resource to prepare the response data. |
| ResponseData | A response byte array of at least **2 bytes** must be assigned to this property. If this is not done, the remote invoking party will keep waiting for this response until it receives the following exception:<br>FrontEdge.SkyBridge.ServerAndClientCommon.SendGenericPayloadException: 'No response received in time.' |

```
public class ServiceTerminatedEventArgs : EventArgs
{
    public DisconnectReason DisconnectReason { get; set; }
}
```

*[Description]*

This event is fired when the service is terminated.

*[Event properties]*

| Parameter | Description |
|---|---|
| DisconnectReason | An enum indicating the reason of the termination. |

## The Terminate method

```
public void Terminate()
```

*[Description]*

Once this method is called, this `InstantRemotingService` instance is terminally disposed.

This method is guaranteed not to throw any exception.

## The GetOkBytes method

```
public static byte[] GetOkBytes();
```

*[Description]*

Create a two-byte array which can be converted by UTF8 encoding to “OK”.

In your method which is invoked by the remote invoker, if you have nothing to send back to the invoker, you should call this method to generate the two-byte array and send that back.

## The IsOkBytes method

```
public static bool IsOkBytes(byte[] bytes);
```

*[Description]*

Check if an array is the two-byte “OK” array, which can be acquired by calling method `GetOkBytes`.

# Sample code

## Simplest example

The following sample code runs as a free and anonymous user, therefore no account ID and API key is needed.

### Start a service

1. Open Visual Studio
2. Create a console application targeting .NET Framework 4.7.2 or above.
3. Download the SkyBridge.InstantRemoting library from NuGet.
4. Paste the following code.
5. Hit F5 key.

```
using System;
using System.Text;
using FrontEdge.SkyBridge.InstantRemoting;

class Program
{
    static void ServiceInvoked(object sender, ServiceInvokedEventArgs args)
    {
        Console.WriteLine(Encoding.UTF8.GetString(args.InvocationData));
        args.ResponseData = Encoding.UTF8.GetBytes("What's up?");
    }

    static void Main(string[] args)
    {
        var api = new InstantRemotingService("<your service name>", ServiceInvoked);
        Console.ReadKey();
        api.Terminate();
    }
}
```

### Create a client to invoke the service

Go through the same steps as creating a service, except to paste the following code:

```
using System;
using System.Text;
using FrontEdge.SkyBridge.InstantRemoting;

class Program
{
    static void Main(string[] args)
    {
        var bytes = Encoding.UTF8.GetBytes("Hello!");
        var api = new InstantRemotingClient();
        var response = api.InvokeServiceByName("<your service name>", bytes);
        Console.WriteLine(Encoding.UTF8.GetString(response));
        Console.ReadKey();
        api.Terminate();
    }
}
```

## A service connecting only on demand

### Service

The following service stays disconnected and polls the server for demands for itself every 20 seconds. Once the server tells it that it is in demand, it connects and stay connected, until there is no invocation or demand for 15 seconds.

```
using System;
using System.Text;
using FrontEdge.SkyBridge.InstantRemoting;

class Program
{
    static void ServiceInvoked(object sender, ServiceInvokedEventArgs args)
    {
        Console.WriteLine(Encoding.UTF8.GetString(args.InvocationData));
        args.ResponseData = Encoding.UTF8.GetBytes("What's up?");
    }

    static void Main(string[] args)
    {
        var api = new InstantRemotingService(
            serviceName: "My frist service",
            serviceInvokedHandler: ServiceInvoked,
            accountId: 123,
            apiKey: "abc123",
            pollForDemandIntervalInSecs: 20,
            disconnectAfterIdleForSecs: 15);

        Console.ReadKey();
        api.Terminate();
    }
}
```

## Client

The following client does not give up if the server tells it that the service which it targets is not connected. Instead, it keeps trying for 60 seconds.

The server, seeing that this client is tenacious, remembers its demand for the target service.

When the service polls for demand next time, the server tells the service that it is in demand. Thus, the service connects and stay connected for 15 seconds. This way, when the client tries again, the server will connect it with the service.

```
using System;
using System.Text;
using FrontEdge.SkyBridge.InstantRemoting;

class Program
{
    static void Main(string[] args)
    {
        var api = new InstantRemotingClient(accountId: 1,  apiKey: "abc123");

        while (true)
        {
            Console.WriteLine("Hit any key to invoke service, or 'Q'/'q' to quit...");
            var key = Console.ReadKey();

            if (key.KeyChar == 'Q' || key.KeyChar == 'q')
                break;

            var response = api.InvokeServiceByName(
                serviceName: "My frist service",
                data: Encoding.UTF8.GetBytes("Hello!"),
                retrySendingForSeconds: 60);

            Console.WriteLine(Encoding.UTF8.GetString(response));
        }

        api.Terminate();
    }
}
```

For this client-awakened service mechanism to work, pparameter `retrySendingForSeconds` passed to the client must have a bigger value than parameter `pollForDemandIntervalInSecs` passed to the service. Otherwise, when the service polls again, the client has already given up.