Process.GetProcessesByName Method

Definition

Namespace: System.Diagnostics

Assembly: System.Diagnostics.Process.dll

Creates an array of new Process components and associates them with the existing process resources that all share the specified process name.

In this article

Overloads

GetProcessesByName(String, String)	Creates an array of new Process components and associates them with all the process resources on a remote computer that share the specified process name.
GetProcessesByName(String)	Creates an array of new Process components and associates them with all the process resources on the local computer that share the specified process name.

GetProcessesByName(String, String)

Creates an array of new Process components and associates them with all the process resources on a remote computer that share the specified process name.

C# Copy

```
public static System.Diagnostics.Process[] GetProcessesByName (string?
processName, string machineName);
```

Parameters

`processName` String

The friendly name of the process.

`machineName` String

The name of a computer on the network.

Returns

Process[]

An array of type Process that represents the process resources running the specified application or file.

Exceptions

ArgumentException

The `machineName` parameter syntax is invalid. It might have length zero (0).

ArgumentNullException

The `machineName` parameter is `null`.

PlatformNotSupportedException

The operating system platform does not support this operation on remote computers.

InvalidOperationException

The attempt to connect to `machineName` has failed.

-or-

There are problems accessing the performance counter APIs used to get process information. This exception is specific to Windows NT, Windows 2000, and Windows XP.

Win32Exception

A problem occurred accessing an underlying system API.

Examples

The following example retrieves information of the current process, processes running on the local computer, all instances of Notepad running on the local computer, and a

specific process on the local computer. It then retrieves information for the same processes on a remote computer.

C# Copy

```
using System;
using System.Diagnostics;
using System.ComponentModel;

namespace MyProcessSample
{
    class MyProcess
    {
        void BindToRunningProcesses()
        {
            // Get the current process.
            Process currentProcess = Process.GetCurrentProcess();

            // Get all processes running on the local computer.
            Process[] localAll = Process.GetProcesses();

            // Get all instances of Notepad running on the local
computer.
            // This will return an empty array if notepad isn't
running.
            Process[] localByName =
Process.GetProcessesByName("notepad");

            // Get a process on the local computer, using the process
id.
            // This will throw an exception if there is no such
process.
            Process localById = Process.GetProcessById(1234);

            // Get processes running on a remote computer. Note that
this
            // and all the following calls will timeout and throw an
exception
            // if "myComputer" and 169.0.0.0 do not exist on your
local network.

            // Get all processes on a remote computer.
            Process[] remoteAll = Process.GetProcesses("myComputer");

            // Get all instances of Notepad running on the specific
computer, using machine name.
            Process[] remoteByName =
Process.GetProcessesByName("notepad", "myComputer");

            // Get all instances of Notepad running on the specific
computer, using IP address.
            Process[] ipByName = Process.GetProcessesByName("notepad",
"169.0.0.0");

            // Get a process on a remote computer, using the process
id and machine name.
```

```
            Process remoteById = Process.GetProcessById(2345,
"myComputer");
        }

        static void Main()
        {
            MyProcess myProcess = new MyProcess();
            myProcess.BindToRunningProcesses();
        }
    }
}
```

Remarks

Use this method to create an array of new Process components and associate them with all the process resources that are running the same executable file on the specified computer. The process resources must already exist on the computer, because GetProcessesByName does not create system resources but rather associates them with application-generated Process components. A `processName` can be specified for an executable file that is not currently running on the local computer, so the array the method returns can be empty.

The process name is a friendly name for the process, such as Outlook, that does not include the .exe extension or the path. GetProcessesByName is helpful for getting and manipulating all the processes that are associated with the same executable file. For example, you can pass an executable file name as the `processName` parameter, in order to shut down all the running instances of that executable file.

Although a process Id is unique to a single process resource on the system, multiple processes on the local computer can be running the application specified by the `processName` parameter. Therefore, GetProcessById returns one process at most, but GetProcessesByName returns an array containing all the associated processes. If you need to manipulate the process using standard API calls, you can query each of these processes in turn for its identifier. You cannot access process resources through the process name alone but, once you have retrieved an array of Process components that have been associated with the process resources, you can start, terminate, and otherwise manipulate the system resources.

You can use this overload to get processes on the local computer as well as on a remote computer. Use "." to specify the local computer. Another overload exists that uses the local computer by default.

You can access processes on remote computers only to view information, such as statistics, about the processes. You cannot close, terminate (using Kill), or start processes on remote computers.

See also

- ProcessName
- MachineName
- GetProcessById(Int32, String)
- GetProcesses()
- GetCurrentProcess()

Applies to

▶ .NET 6.0 RC 1 and other versions

GetProcessesByName(String)

Creates an array of new Process components and associates them with all the process resources on the local computer that share the specified process name.

C# Copy

```
public static System.Diagnostics.Process[] GetProcessesByName (string?
processName);
```

Parameters

processName String

The friendly name of the process.

Returns

Process[]

An array of type Process that represents the process resources running the specified application or file.

Exceptions

InvalidOperationException

There are problems accessing the performance counter APIs used to get process information. This exception is specific to Windows NT, Windows 2000, and Windows XP.

Examples

The following example retrieves information of the current process, processes running on the local computer, all instances of Notepad running on the local computer, and a specific process on the local computer. It then retrieves information for the same processes on a remote computer.

C# Copy

```
using System;
using System.Diagnostics;
using System.ComponentModel;

namespace MyProcessSample
{
    class MyProcess
    {
        void BindToRunningProcesses()
        {
            // Get the current process.
            Process currentProcess = Process.GetCurrentProcess();

            // Get all processes running on the local computer.
            Process[] localAll = Process.GetProcesses();

            // Get all instances of Notepad running on the local
computer.
            // This will return an empty array if notepad isn't
running.
            Process[] localByName =
Process.GetProcessesByName("notepad");

            // Get a process on the local computer, using the process
id.
            // This will throw an exception if there is no such
process.
            Process localById = Process.GetProcessById(1234);

            // Get processes running on a remote computer. Note that
this
            // and all the following calls will timeout and throw an
exception
            // if "myComputer" and 169.0.0.0 do not exist on your
local network.

            // Get all processes on a remote computer.
            Process[] remoteAll = Process.GetProcesses("myComputer");

            // Get all instances of Notepad running on the specific
computer, using machine name.
            Process[] remoteByName =
Process.GetProcessesByName("notepad", "myComputer");

            // Get all instances of Notepad running on the specific
computer, using IP address.
```

```
            Process[] ipByName = Process.GetProcessesByName("notepad",
"169.0.0.0");

            // Get a process on a remote computer, using the process
id and machine name.
            Process remoteById = Process.GetProcessById(2345,
"myComputer");
        }

        static void Main()
        {
            MyProcess myProcess = new MyProcess();
            myProcess.BindToRunningProcesses();
        }
    }
}
```

Remarks

Use this method to create an array of new Process components and associate them with all the process resources that are running the same executable file on the local computer. The process resources must already exist on the computer, because GetProcessesByName does not create system resources but rather associates them with application-generated Process components. A `processName` can be specified for an executable file that is not currently running on the local computer, so the array the method returns can be empty.

The process name is a friendly name for the process, such as Outlook, that does not include the .exe extension or the path. GetProcessesByName is helpful for getting and manipulating all the processes that are associated with the same executable file. For example, you can pass an executable file name as the `processName` parameter, in order to shut down all the running instances of that executable file.

Although a process Id is unique to a single process resource on the system, multiple processes on the local computer can be running the application specified by the `processName` parameter. Therefore, GetProcessById returns one process at most, but GetProcessesByName returns an array containing all the associated processes. If you need to manipulate the process using standard API calls, you can query each of these processes in turn for its identifier. You cannot access process resources through the process name alone but, once you have retrieved an array of Process components that have been associated with the process resources, you can start, terminate, and otherwise manipulate the system resources.

See also

- ProcessName
- GetProcessById(Int32, String)
- GetProcesses()
- GetCurrentProcess()

Applies to

▶ .NET 6.0 RC 1 and other versions

Is this page helpful?

Yes No