



How to list all Windows Processes and their attributes (Task Manager like) with C# in WinForms

[Home](#) > [ARTICLES](#) > [C#](#)

How to list all Windows Processes and their attributes (Task Manager like) with C# in WinForms

Published : April 28th 2017 **Last modification :** April 28th 2017 **7.7K views**

Actions ▼



Process name	PID	Status	Username	Memory (private working set)	Description
SearchIndexer	4040	Responding	SYSTEM	83 MB	Microsoft Windows Search Indexer
SearchUI	12428	Responding	sdcca	102 MB	Search and Cortana application
ServiceHub.DataWarehouseHost	14004	Responding	sdcca	35 MB	
ServiceHub.Host.CLR.x86	1136	Responding	sdcca	21 MB	
ServiceHub.Host.Node.x86	1552	Responding	sdcca	21 MB	Node.js: Server-side JavaScript
ServiceHub.IdentityHost	10352	Responding	sdcca	20 MB	
ServiceHub.RoslynCodeAnalysisService32	6484	Responding	sdcca	58 MB	
ServiceHub.SettingsHost	10644	Responding	sdcca	53 MB	
ServiceHub.VSDetouredHost	6400	Responding	sdcca	33 MB	
services	856	Responding	SYSTEM	11 MB	
SettingSyncHost	9012	Responding	sdcca	4 MB	Host Process for Setting Synchronization
setup	15664	Responding	sdcca	3 MB	Google Chrome Installer
ShellExperienceHost	15024	Responding	sdcca	42 MB	Windows Shell Experience Host
sihost	17004	Responding	sdcca	6 MB	
SkypeHost	6372	Responding	sdcca	32 MB	Microsoft Skype
smss	404	Responding	SYSTEM	404 KB	
spoolsv	1988	Responding	SYSTEM	11 MB	
Spotify	19060	Responding	sdcca	90 MB	Spotify
Spotify	2072	Responding	sdcca	81 MB	Spotify
Spotify	16604	Responding	sdcca	134 MB	Spotify
Spotify	8496	Responding	sdcca	28 MB	Spotify
SpotifyWebHelper	12660	Responding	sdcca	1 MB	SpotifyWebHelper
sqlwriter	2736	Responding	SYSTEM	2 MB	SQL Server VSS Writer - 64 Bit
StandardCollector.Service	5220	Responding	SYSTEM	50 MB	Microsoft (R) Visual Studio Standard Collector
Steam	1640	Responding	sdcca	40 MB	Steam Client Bootstrapper
SteamService	8968	Responding	SYSTEM	7 MB	Steam Client Service
steamwebhelper	10708	Responding	sdcca	17 MB	Steam Client WebHelper
svchost	1964	Responding	LOCAL ...	3 MB	Host Process for Windows Services
svchost	2544	Responding	SYSTEM	7 MB	Host Process for Windows Services

If you're looking for creating your custom Task Manager or just to create a Widget for your app that displays information about the processes in Windows, then you are in the right place.

In this article you will learn how to get information about the processes in Windows and how to create a Task Manager like application easily.

1. Use the Diagnostics namespace

In order to work with processes in your application, you will need to import the `System.Diagnostics` namespace at the top of your class:

```
using System.Diagnostics;
```

[Copy snippet](#)

The `System.Diagnostics` namespace provides classes that allow you to interact with system processes, event logs, and performance counters.

2. List and retrieve basic information of all processes

If you're here because you only need to know how to retrieve the list of all processes in Windows, then you only will need to pay attention to the following snippet:

```
// Create an array to store the processes  
Process[] processList = Process.GetProcesses();
```

[Copy snippet](#)

```
// Loop through the array to show information of every process in your console
foreach (Process process in processList)
{
    Console.WriteLine(@"
        {0} | ID: {1} | Status {2} | Memory (private working set in Bytes) {3}",
        process.ProcessName, process.Id, process.Responding, process.PrivateMemorySize64);
}
```

The `GetProcesses` method will do the trick for you by returning an array that contains information about all the process on Windows. The execution of the previous code in your project, should generate an output similar to:

```
Memory Compression | ID: 2560 | Status True | Memory (private working set)876544
conhost | ID: 9060 | Status True | Memory (private working set)5054464
MSASCuiL | ID: 13196 | Status True | Memory (private working set)3203072
svchost | ID: 1964 | Status True | Memory (private working set)3485696
chrome | ID: 18740 | Status True | Memory (private working set)97230848
chrome | ID: 2044 | Status True | Memory (private working set)26988544
VBCSCompiler | ID: 6688 | Status True | Memory (private working set)103931904
```

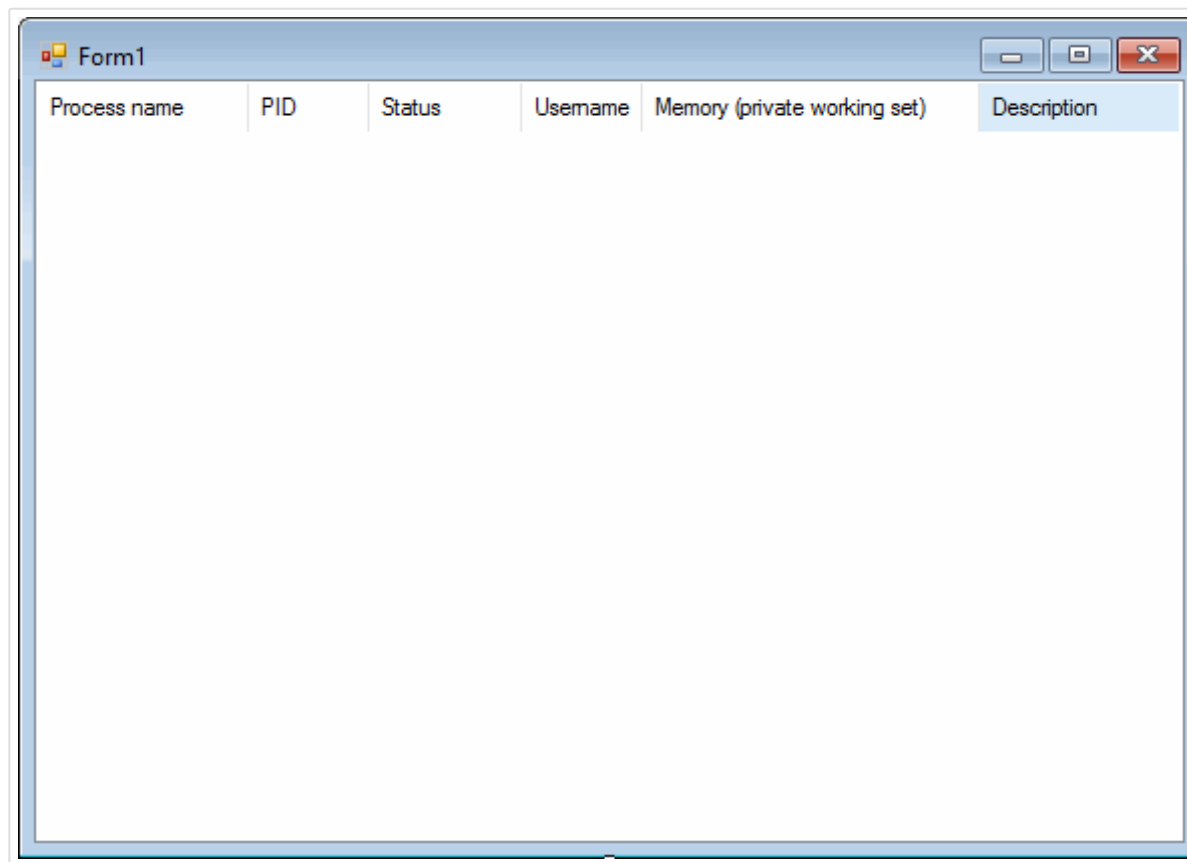
[Copy snippet](#)

Pretty easy right? A process object has a lot of properties that you can display to your user if you need to, [read more about the properties here in MSDN](#).

3. Creating a Task Manager like application

As first step, you will need to create a List View item in your form in order to show the information on it. This List View item needs to be configured as following:

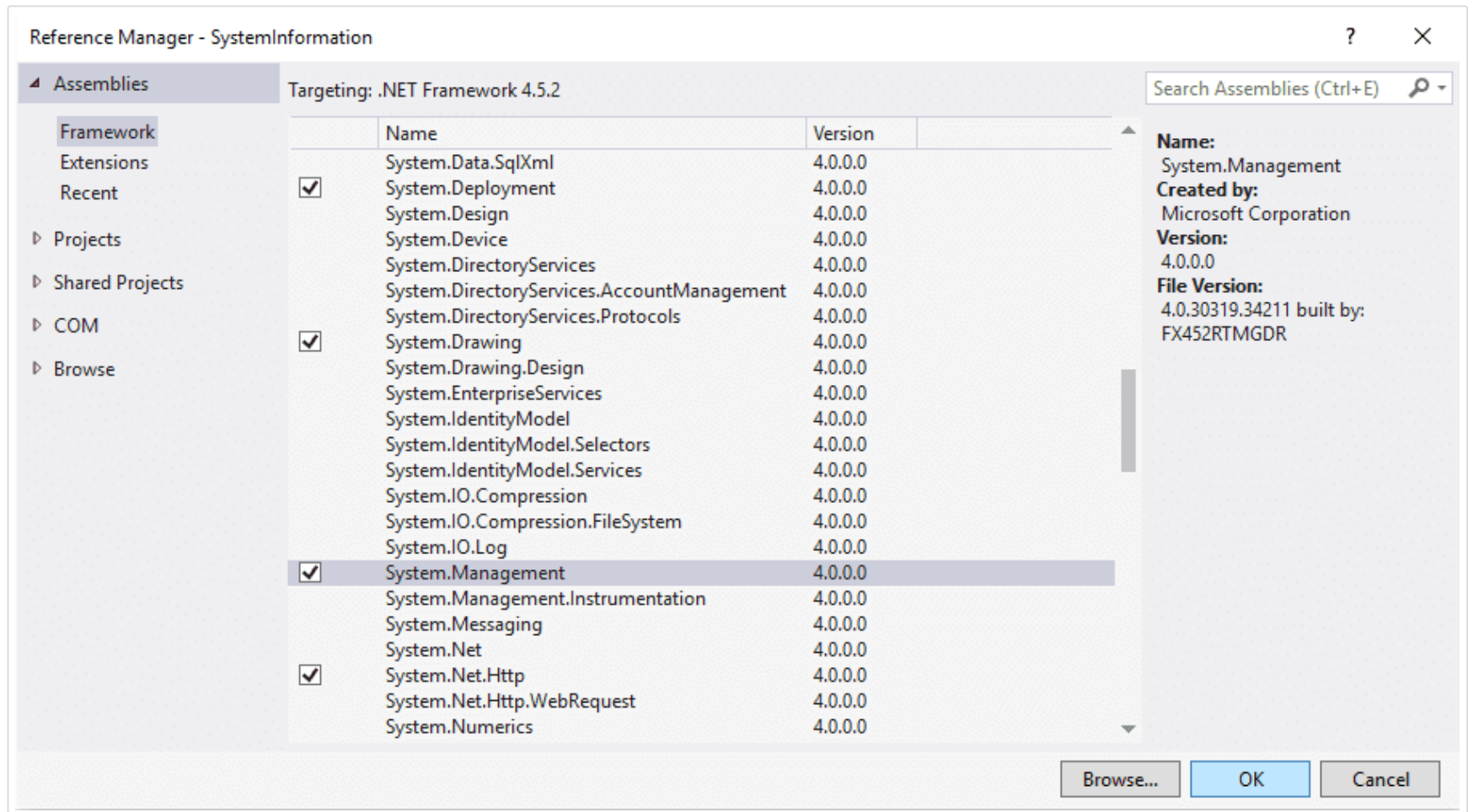
1. On the properties panel of the element (bottom right corner) set the **View** property to **Details** from the dropdown menu.
2. On the same panel select the Columns property and a new window should appear, in this window you will create the 6 required columns that we'll use to display respectively the same information as the Task Manager:



Now you'll be ready to display some information on this list view easily.

If you've already read about all the properties that an object inside the array returned by the `Process.GetProcesses` method, you will see that in comparison with the information that you see in the original Task Manager of Windows it's not so complete at all. The processes list doesn't contain all the information that we need to show in our list, therefore we'll need to add a reference to `System.Management` to get the username related to the process and the description. For it, you will need to add the reference with the reference manager of Visual Studio. To do that follow these steps:

1. Right Click on Project, **Add References**
2. Select the **Assemblies (framework)** Tab and Search for `System.Management` and finally add the reference and click OK.



We need to add the `System.Management` to create queries in `WMI Classes`. [Read more about retrieving WMI Classes in .NET in msdn here](#). Once the reference has been added to your project, you can import the

namespace of `System.Management` in your project. Besides import the `System.Dynamic` too at the top of your class:


```
using System.Management;  
using System.Dynamic;
```

 Copy snippet

Finally we can start with the code to display information about the processes on your application.

Is up to you when and where you want to execute the code that we'll explain now, it can be either executed by the click of a button or at the `Form_Load` event of the form. Before start with the rendering, we need to solve the first problem, namely that the list of processes doesn't contain the Username of the Windows User that started the process and the description of the process. As we said previously, we'll need to query a WMI class namely the `Win32_Process`, this query will search a process by a single ID. As you may know every process in Windows has an ID in order to be able to identify easily, therefore we can create the following method that expects as first argument the ID of the process that you want to query and it will return an Expando object with the 2 properties that we need: Description and Username:

```
/// <summary>  
/// Returns an Expando object with the description and username of a process from the process ID.  
/// </summary>  
/// <param name="processId"></param>  
/// <returns></returns>  
public ExpandoObject GetProcessExtraInformation(int processId)  
{
```

 Copy snippet


```
// Query the Win32_Process
string query = "Select * From Win32_Process Where ProcessID = " + processId;
ManagementObjectSearcher searcher = new ManagementObjectSearcher(query);
ManagementObjectCollection processList = searcher.Get();

// Create a dynamic object to store some properties on it
dynamic response = new ExpandoObject();
response.Description = "";
response.Username = "Unknown";

foreach (ManagementObject obj in processList)
{
    // Retrieve username
    string[] argList = new string[] { string.Empty, string.Empty };
    int returnVal = Convert.ToInt32(obj.InvokeMethod("GetOwner", argList));
    if (returnVal == 0)
    {
        // return Username
        response.Username = argList[0];

        // You can return the domain too like (PCDesktop-123123\Username using instead
        //response.Username = argList[1] + "\\\" + argList[0];
    }

    // Retrieve process description if exists
    if (obj["ExecutablePath"] != null)
    {

```

```

        try
        {
            FileVersionInfo info = FileVersionInfo.GetVersionInfo(obj["ExecutablePath"].ToString());
            response.Description = info.FileDescription;
        }
        catch{}
    }

    return response;
}

```


The function is pretty easy to understand and it will be executed with every process that the GetProcesses provide.

The second problem is, that you don't want to display the value of the used memory to the user in Bytes as it isn't easy to read at all, so if you make it readable with understandable representations as KB, MB or GB etc, they will be thankful. To convert the bytes returned from the process information into a readable value, you can use the following method:

```

/// <summary>
/// Method that converts bytes to its human readable value
/// </summary>
/// <param name="number"></param>
/// <returns></returns>

```

 Copy snippet

```

public string BytesToReadableValue(long number)
{
    List<string> suffixes = new List<string> { " B", " KB", " MB", " GB", " TB", " PB" };

    for (int i = 0; i < suffixes.Count; i++)
    {
        long temp = number / (int)Math.Pow(1024, i + 1);

        if (temp == 0)
        {
            return (number / (int)Math.Pow(1024, i)) + suffixes[i];
        }
    }

    return number.ToString();
}

```

Note that the method expects a long value and it returns a string. As final step we can now start to render the information on the List View.

You need to retrieve the list of processes and store it into a variable, besides you should create an Image List item that will store the icons of every process (in case is there any). Then we'll need to loop through the processes array with a foreach statement, on every loop the `GetProcessExtraInformation` will be executed (with the ID of the process as first argument) to retrieve the necessary information that we don't have from the process variable on the loop. Then the row array will be created and it will store (respectively


as the values need to follow the same order as the Columns in the List View item) the information of every row in the List View item (note that we'll convert the bytes to a readable value using the `BytesToReadableValue` method). Optionally, you can add the icons to the list by adding a new item to the Image List previously created. In the image list you need to provide an identifier (key) for the image that you want to add (in this case the icon of the process) and as second argument the Image Data of the icon that can be retrieved using the `Icon.ExtractAssociatedIcon` method that expects the path to the executable of the process (that can be retrieved through the process object), this icon can be converted to bitmap as shown in the example to increase the quality, however you can remove the `toBitmap` method if you want.

Finally create a new `ListViewItem` to be added with the row as first argument and specify which image from the Image List should be used for the item and add it to the List View:

```
/// <summary>
/// This method renders all the processes of Windows on a ListView with some values and icons.
/// </summary>
public void renderProcessesOnListView()
{
    // Create an array to store the processes
    Process[] processList = Process.GetProcesses();

    // Create an Imagelist that will store the icons of every process
    ImageList Imagelist = new ImageList();

    // Loop through the array of processes to show information of every process in your console
    foreach (Process process in processList)
```

 Copy snippet

```

{
    // Define the status from a boolean to a simple string
    string status = (process.Responding == true ? "Responding" : "Not responding");

    // Retrieve the object of extra information of the process (to retrieve Username and Description)
    dynamic extraProcessInfo = GetProcessExtraInformation(process.Id);

    // Create an array of string that will store the information to display in our
    string[] row = {
        // 1 Process name
        process.ProcessName,
        // 2 Process ID
        process.Id.ToString(),
        // 3 Process status
        status,
        // 4 Username that started the process
        extraProcessInfo.Username,
        // 5 Memory usage
        BytesToReadableValue(process.PrivateMemorySize64),
        // 6 Description of the process
        extraProcessInfo.Description
    };

    //
    // As not every process has an icon then, prevent the app from crash
    try
    {

```

```
        Imagelist.Images.Add(
            // Add an unique Key as identifier for the icon (same as the ID of the process)
            process.Id.ToString(),
            // Add Icon to the List
            Icon.ExtractAssociatedIcon(process.MainModule.FileName).ToBitmap()
        );
    }
    catch { }

    // Create a new Item to add into the list view that expects the row of information as first argument
    ListViewItem item = new ListViewItem(row)
    {
        // Set the ImageIndex of the item as the same defined in the previous try-catch
        ImageIndex = Imagelist.Images.IndexOfKey(process.Id.ToString())
    };

    // Add the Item
    listView1.Items.Add(item);
}

// Set the imagelist of your list view the previous created list :)
listView1.LargeImageList = Imagelist;
listView1.SmallImageList = Imagelist;
}
```

Note that you need to set the value of the `LargelmageList` and `SmalllImageList` properties as the `ImageList` that we've created for the icons. To make this work, you only need to execute the `renderProcessesOnListView` function when you need to display the services in your List View.


4. Final example

If you've already read how everything works, you can simply follow the entire example and implement it by yourself in your project:

Important

Don't forget that you need to add a reference to the `System.Management` to your project as shown in the step #3. By the other side you need to create a List View Item identified as `listView1` with the View property set to Details and with respectively 6 items in the Columns property.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

 Copy snippet

```
using System.Windows.Forms;

// Required namespaces
using System.Diagnostics;
using System.Management;
using System.Dynamic;

namespace Sandbox
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // Once the form loads, render the items on the list
            renderProcessesOnListView();
        }

        /// <summary>
        /// This method renders all the processes of Windows on a ListView with some values and icons.
        /// </summary>
        public void renderProcessesOnListView()
        {

```



```
// Create an array to store the processes
Process[] processList = Process.GetProcesses();

// Create an ImageList that will store the icons of every process
ImageList ImageList = new ImageList();

// Loop through the array of processes to show information of every process in your console
foreach (Process process in processList)
{
    // Define the status from a boolean to a simple string
    string status = (process.Responding == true ? "Responding" : "Not responding");

    // Retrieve the object of extra information of the process (to retrieve Username and Description)
    dynamic extraProcessInfo = GetProcessExtraInformation(process.Id);

    // Create an array of string that will store the information to display in our console
    string[] row = {
        // 1 Process name
        process.ProcessName,
        // 2 Process ID
        process.Id.ToString(),
        // 3 Process status
        status,
        // 4 Username that started the process
        extraProcessInfo.Username,
        // 5 Memory usage
        BytesToReadableValue(process.PrivateMemorySize64),
    };
}
```

```

        // 6 Description of the process
        extraProcessInfo.Description
    };

    //
    // As not every process has an icon then, prevent the app from crash
    try
    {
        Imagelist.Images.Add(
            // Add an unique Key as identifier for the icon (same as the ID of the process)
            process.Id.ToString(),
            // Add Icon to the List
            Icon.ExtractAssociatedIcon(process.MainModule.FileName).ToBitmap()
        );
    }
    catch { }

    // Create a new Item to add into the list view that expects the row of information as file
    ListViewItem item = new ListViewItem(row)
    {
        // Set the ImageIndex of the item as the same defined in the previous try-catch
        ImageIndex = Imagelist.Images.IndexOfKey(process.Id.ToString())
    };

    // Add the Item
    listView1.Items.Add(item);
}

```

```
// Set the imagelist of your list view the previous created list :)
listView1.LargeImageList = Imagelist;
listView1.SmallImageList = Imagelist;
}

/// <summary>
/// Method that converts bytes to its human readable value
/// </summary>
/// <param name="number"></param>
/// <returns></returns>
public string BytesToReadableValue(long number)
{
    List<string> suffixes = new List<string> { " B", " KB", " MB", " GB", " TB", " PB" };

    for (int i = 0; i < suffixes.Count; i++)
    {
        long temp = number / (int)Math.Pow(1024, i + 1);

        if (temp == 0)
        {
            return (number / (int)Math.Pow(1024, i)) + suffixes[i];
        }
    }

    return number.ToString();
}
```

```

/// <summary>
/// Returns an Expando object with the description and username of a process from the process ID
/// </summary>
/// <param name="processId"></param>
/// <returns></returns>
public ExpandoObject GetProcessExtraInformation(int processId)
{
    // Query the Win32_Process
    string query = "Select * From Win32_Process Where ProcessID = " + processId;
    ManagementObjectSearcher searcher = new ManagementObjectSearcher(query);
    ManagementObjectCollection processList = searcher.Get();

    // Create a dynamic object to store some properties on it
    dynamic response = new ExpandoObject();
    response.Description = "";
    response.Username = "Unknown";

    foreach (ManagementObject obj in processList)
    {
        // Retrieve username
        string[] argList = new string[] { string.Empty, string.Empty };
        int returnVal = Convert.ToInt32(obj.InvokeMethod("GetOwner", argList));
        if (returnVal == 0)
        {
            // return Username
            response.Username = argList[0];
        }
    }
}

```

```
        // You can return the domain too like (PCDesktop-123123\Username using instead
        //response.Username = argList[1] + "\\\" + argList[0];
    }

    // Retrieve process description if exists
    if (obj["ExecutablePath"] != null)
    {
        try
        {
            FileVersionInfo info = FileVersionInfo.GetVersionInfo(obj["ExecutablePath"].ToString());
            response.Description = info.FileDescription;
        }
        catch{}
    }

    return response;
}
}
```

As shown in the image of the article, a window that lists all the processes in Windows will appear and the icon will be shown next to the name of the process and the other properties.

Happy coding ♥!

@ E-mail

 Tweet

 Like

 +1

 Pin it

Follow Our Code World on Twitter



Like Our Code World on Facebook



Subscribe to our YouTube Channel

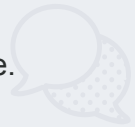


This could interest you

Become a more social person

Our Code World Comment Policy

Our Comments Section is open to every developer, so you can contribute (even code) to the main idea of the Article.
Please read our [Comment Policy](#) before commenting.



0 Comments

Our Code World

1 Login ▾

Recommend

Tweet

Share

Sort by Best ▾



Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS ?



Name

Be the first to comment.

ALSO ON OUR CODE WORLD

Top 20: Best Open Source Vue.js UI Frameworks

2 comments • 10 months ago

Dominic Lapointe — Quasar is missing from the list. It do

What does the "Not allowed to navigate top frame to data ...

2 comments • a year ago

deserve a place here ;)

Creating Collapsible Tree Structures from JSON into ...

1 comment • 6 months ago

samx — thanks a lot for this excellent tutorial, you saved my day!

Liew Xun — These links might help you for an IE implementation, not sure about Safari though, if ...

How to convert a MP3 file to WAV with NAudio in WinForms C#

1 comment • a year ago

Arnel Ambrosio — Please if you can help how to convert wav file to aac format, thanks a lot!



Subscribe



Add Disqus to your site



Disqus' Privacy Policy

DISQUS

Related articles

How to implement and use Circular Progress Bars in WinForms with C#

C# • August 21st 2018

How to record the audio from the sound card (system audio) with C# using NAudio in WinForms

C# • February 22nd 2018

How to convert a MP3 file to WAV with NAudio in WinForms C#

C# • February 19th 2018

[How to create and extract zip files \(compress and decompress zip\) with SharpZipLib with C# in Wi...](#)

C# • November 17th 2017

[Creating a C# Application with Chrome-Style Tabs using EasyTabs in WinForms](#)

C# • September 19th 2017

[Advertise with Our Code World](#)

Looking for new web templates?

Best Free HTML/CSS Templates



 Find the best free templates here

Don't forget to follow us on your favorite social network

Enjoying this article? Follow us and don't miss any new content !



Follow @ourcodeworld on Twitter



Like Our Code World on Facebook



Subscribe to our YouTube channel



Make a donation



Write for us

