

How to extract text from large PDF document in parallel processing in C# with ByteScout Data Extraction Suite

Learning is essential in computer world and the tutorial below will demonstrate how to extract text from large PDF document in parallel processing in C#

An easy to understand sample source code to learn how to extract text from large PDF document in parallel processing in C# ByteScout Data Extraction Suite can extract text from large PDF document in parallel processing. It can be applied from C#. ByteScout Data Extraction Suite is the set that includes 3 SDK products for data extraction from PDF, scans, images and from spreadsheets: PDF Extractor SDK, Data Extraction SDK, Barcode Reader SDK.

Want to quickly learn? This fast application programming interfaces of ByteScout Data Extraction Suite for C# plus the guidelines and the code below will help you quickly learn how to extract text from large PDF document in parallel processing. IF you want to implement the functionality, just copy and paste this code for C# below into your code editor with your app, compile and run your application. Use of ByteScout Data Extraction Suite in C# is also described in the documentation included along with the product.

ByteScout provides the free trial version of ByteScout Data Extraction Suite along with the documentation and source code samples.

FOR MORE INFORMATION AND FREE TRIAL:

[Download Free Trial SDK \(on-premise version\)](#)

[Read more about ByteScout Data Extraction Suite](#)

[Explore API Documentation](#)

[Get Free Training for ByteScout Data Extraction Suite](#)

[Get Free API key for Web API](#)

[visit www.Bytescout.com](http://www.Bytescout.com)

Source Code Files:

MultithreadProcessing.sln

```
Microsoft Visual Studio Solution File, Format Version 12.00
# Visual Studio 15
VisualStudioVersion = 15.0.26730.10
MinimumVisualStudioVersion = 10.0.40219.1
Project("{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}") = "MultithreadProcessing", "MultithreadProcessing.csproj", "{0B102DA4-C143-481D-A076-1F56E3CB1CF5}"
EndProject
Global
    GlobalSection(SolutionConfigurationPlatforms) = preSolution
        Debug|Any CPU = Debug|Any CPU
        Release|Any CPU = Release|Any CPU
    EndGlobalSection
    GlobalSection(ProjectConfigurationPlatforms) = postSolution
        {0B102DA4-C143-481D-A076-1F56E3CB1CF5}.Debug|Any CPU.ActiveCfg = Debug|Any CPU
        {0B102DA4-C143-481D-A076-1F56E3CB1CF5}.Debug|Any CPU.Build.0 = Debug|Any CPU
        {0B102DA4-C143-481D-A076-1F56E3CB1CF5}.Release|Any CPU.ActiveCfg = Release|Any CPU
        {0B102DA4-C143-481D-A076-1F56E3CB1CF5}.Release|Any CPU.Build.0 = Release|Any CPU
    EndGlobalSection
    GlobalSection(SolutionProperties) = preSolution
        HideSolutionNode = FALSE
    EndGlobalSection
    GlobalSection(ExtensibilityGlobals) = postSolution
        SolutionGuid = {50466307-7059-438B-8545-42FDA71BC1A6}
    EndGlobalSection
EndGlobal
```

Program.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Threading;
using Bytescout.PDFExtractor;

namespace MultithreadProcessing
{
    class Program
    {
        // Limit to 4 threads in queue.
        // Set this value to number of cores in your CPU for max performance.
        private static readonly Semaphore _threadLimiter = new Semaphore(4, 4);
        private static int _runningThreadsCounter;

        static void Main(string[] args)
        {
            // ...
        }
    }
}
```

```

{
    const string inputFileName = "sample.pdf";
    const string resultFileName = "result.txt";
    int CHUNK_SIZE = 10;

    int pageCount;

    // Get document page count
    using (var infoExtractor = new InfoExtractor("demo", "demo"))
    {
        infoExtractor.LoadDocumentFromFile(inputFileName);
        pageCount = infoExtractor.GetPageCount();
    }

    Stopwatch stopwatch = Stopwatch.StartNew();

    int numberOfThreads = pageCount / CHUNK_SIZE;
    if (pageCount - numberOfThreads * CHUNK_SIZE > 0)
        numberOfThreads += 1;

    ManualResetEvent allFinishedEvent = new ManualResetEvent(false);
    _runningThreadsCounter = 0;
    string[] chunks = new string[numberOfThreads];

    for (int i = 0; i < numberOfThreads; i++)
    {
        // Wait for the queue
        _threadLimiter.WaitOne();

        var startPage = i * CHUNK_SIZE;
        var endPage = Math.Min(pageCount - 1, (i + 1) * CHUNK_SIZE - 1);

        // Prepare temp file name for the chunk
        chunks[i] = string.Format("temp-{0}-{1}.txt", startPage, endPage);

        // Increase the thread counter
        Interlocked.Increment(ref _runningThreadsCounter);

        ThreadPool.QueueUserWorkItem(new WaitCallback(ThreadProc),
            new object[] { i, allFinishedEvent, inputFileName, chunks[i], startPage, endPage });
    }

    // Wait for all threads
    allFinishedEvent.WaitOne();

    // Merge pieces into a single text file
    using (Stream resultFileStream = File.Create(resultFileName))
    {
        foreach (string tempFile in chunks)
            using (Stream srcStream = File.OpenRead(tempFile))
                srcStream.CopyTo(resultFileStream);
    }

    // Delete temp files
    foreach (string tempFile in chunks)
        File.Delete(tempFile);

    Console.WriteLine("All done in {0}.", stopwatch.Elapsed);
    Console.WriteLine();
}

```

```

        Console.WriteLine("Press any key to exit...");
        Console.ReadKey();
    }

    private static void ThreadProc(object stateInfo)
    {
        int threadIndex = (int) ((object[]) stateInfo)[0];
        ManualResetEvent allFinishedEvent = (ManualResetEvent) ((object[]) stateInfo)[1];
        string inputFile = (string) ((object[]) stateInfo)[2];
        string outputFile = (string) ((object[]) stateInfo)[3];
        int startPage = (int) ((object[]) stateInfo)[4];
        int endPage = (int) ((object[]) stateInfo)[5];

        try
        {
            Console.WriteLine("Thread #{0} started with the page range from {1} to {2}");

            Stopwatch stopwatch = Stopwatch.StartNew();

            // Process the piece
            using (TextExtractor textExtractor = new TextExtractor("demo", "demo"))
            {
                // Set page separator. Default is '\f' (Form Feed)
                textExtractor.PageSeparator = Environment.NewLine;
                // Since we are only extracting text, disable the caching to reduce memory usage
                textExtractor.PageDataCaching = PageDataCaching.None;

                textExtractor.OCRMode = OCRMode.Auto;
                textExtractor.OCRLanguageDataFolder = @"c:\Program Files\Bytescout\OCRLanguageData";
                textExtractor.OCRLanguage = "eng";
                // 300 DPI resolution is recommended.
                // Using of higher values will slow down the processing but does not affect the quality
                textExtractor.OCRResolution = 300;

                textExtractor.LoadDocumentFromFile(inputFile);

                textExtractor.SaveTextToFile(startPage, endPage, outputFile);
            }

            Console.WriteLine("Thread #{0} finished in {1}.", threadIndex, stopwatch.Elapsed);
        }
        finally
        {
            // If it was the last thread, signal the main thread about the finish.
            if (Interlocked.Decrement(ref _runningThreadsCounter) == 0)
                allFinishedEvent.Set();

            // Release semaphore
            _threadLimiter.Release();
        }
    }
}

```

VIDEO

<https://www.youtube.com/watch?v=NEwNs2b9YN8>

ON-PREMISE OFFLINE SDK

[60 Day Free Trial](#) or [Visit ByteScout Data Extraction Suite Home Page](#)
[Explore ByteScout Data Extraction Suite Documentation](#)
[Explore Samples](#)
[Sign Up for ByteScout Data Extraction Suite Online Training](#)

ON-DEMAND REST WEB API

[Get Your API Key](#)
[Explore Web API Docs](#)
[Explore Web API Samples](#)

[visit www.ByteScout.com](http://www.ByteScout.com)

[visit www.PDF.co](http://www.PDF.co)

www.bytescout.com