

# GWS-Kohlhammer-Drucker einrichten

Um den PDF-Druck zu Kohlhammer mit Textextrahierbarer PDF zu realisieren muss ein Script auf dem jeweiligen Server eingerichtet werden.

## Voraussetzungen installieren

### .NET SDK

Zuerst muss das Microsoft [.NET SDK](#) installiert werden. Ob alles richtig installiert ist, kann man mit diesem Befehl prüfen:

```
dotnet --info
```

### Powershell 7

Zur korrekten Ausführung benötigen wir auch Powershell 7, das man [hier](#) laden kann.

### Ordnerstruktur anlegen

Danach legen wir die Ordnerstruktur an. Das kann automatisch mit diesen Befehlen erfolgen:

```
$folders = @(
    "C:\PDF\IN",
    "C:\PDF\OUT",
    "C:\PDF\ERROR",
    "C:\PDF\TEMP"
)

foreach ($folder in $folders) {
    New-Item -ItemType Directory -Path $folder -Force | Out-Null
}
```

```
$work = "C:\Tools\itext_bundle"
New-Item -ItemType Directory -Path $work -Force | Out-Null
Set-Location $work
```

Den Installationspfad kann man je nach Umgebung natürlich anpassen.

## Mini-Projekt erstellen

Die folgenden Befehle führen wir alle in Powershell 7 aus.

```
dotnet new console -n ITextBundle
Set-Location "$work\ITextBundle"
```

## iText + Adapter als NuGet-Pakete hinzufügen

```
dotnet add package itext --version 9.5.0
dotnet add package itext.bouncy-castle-adapter --version 9.5.0
```

jetzt müssen wir das Ganze noch in einen Publish-Ordner ausgeben

```
$pub = "$work\publish"
dotnet publish -c Release -o $pub
```

jetzt noch prüfen, ob die notwendigen Abhängigkeiten vorhanden sind:

```
Get-ChildItem $pub -Filter "Microsoft.Extensions.Logging.dll" | Select Name, FullName
Get-ChildItem $pub -Filter "itext*.dll" | Select Name
Get-ChildItem $pub -Filter "*BouncyCastle*.dll" | Select Name
```

Die eigentlichen Powershell-Scripte:

### [pdf\\_worker.ps1](#)

```
param(
    [Parameter(Mandatory)] [string] $InputPdf,
    [Parameter(Mandatory)] [string] $OutputPdf,
    [Parameter(Mandatory)] [string] $WatermarkPdf,
    [Parameter(Mandatory)] [string] $ITextDllDir,
    [Parameter(Mandatory)] [string] $LogFile,
    [Parameter(Mandatory)] [string] $AddDebugMarker
)

Set-StrictMode -Version Latest
$ErrorActionPreference = "Stop"

$AddDebugMarkerBool = ($AddDebugMarker -eq "True") #Dies setzt, den wieder aktiviert, einen
```

Debug-Marker, der unten Links WM1 oder WM2 andruckt, damit man sehen kann, ob die richtigen Seiten des Briefpapiers genutzt werden.

```
function Write-Log([string]$msg) {
    $line = "[{0}] {1}`r`n" -f (Get-Date), $msg
    try {
        [System.IO.File]::AppendAllText($LogFile, $line)
    } catch {
        # Fallback, falls LogFile kaputt/leer/nicht schreibbar ist
        $fallback = Join-Path $env:TEMP "pdf_worker_fallback.log"
        [System.IO.File]::AppendAllText($fallback, $line)
    }
}

function Import-IText7 {
    param([Parameter(Mandatory)] [string] $DllDir)

    if (-not (Test-Path $DllDir)) { throw "IText-DLL-Verzeichnis existiert nicht: $DllDir" }

    # PS7 / .NET: sauberes Laden über AssemblyLoadContext, ohne Resolve-Handler
    Add-Type -AssemblyName "System.Runtime.Loader" -ErrorAction Stop
    $alc = [System.Runtime.Loader.AssemblyLoadContext]::Default

    # Lade alle DLLs aus dem Ordner. Wichtig: erst commons/io, dann kernel/layout etc.
    $preferredOrder = @(
        "itext.common.dll",
        "itext.io.dll",
        "itext.kernel.dll",
        "itext.layout.dll",
        "itext.barcode.dll"
    )

    # 1) bevorzugte zuerst (wenn vorhanden)
    foreach ($name in $preferredOrder) {
        $p = Join-Path $DllDir $name
        if (Test-Path $p) {
            $full = (Resolve-Path $p).Path
            [void]$alc.LoadFromAssemblyPath($full)
        }
    }
}
```

```

# 2) dann alle übrigen DLLs (z.B. BouncyCastle Adapter/Abhängigkeiten), die noch nicht
geladen sind
$dlls = Get-ChildItem -LiteralPath $DllDir -Filter "*.dll" -File | Sort-Object Name
foreach ($dll in $dlls) {
    try {
        $full = $dll.FullName
        # Wenn bereits geladen, skip (heuristisch über Name)
        $already = [AppDomain]::CurrentDomain.GetAssemblies() |
            Where-Object { $_.GetName().Name -ieq
[IO.Path]::GetFileNameWithoutExtension($dll.Name) }
        if ($already) { continue }

        [void]$alc.LoadFromAssemblyPath($full)
    } catch {
        # absichtlich leise: manche DLLs sind evtl. native/inkompatibel, wir wollen nicht hier
abbrechen
    }
}
}

```

```

function Add-StationeryWatermark {
    param(
        [Parameter(Mandatory)] [string] $InputPdf,
        [Parameter(Mandatory)] [string] $OutputPdf,
        [Parameter(Mandatory)] [string] $WatermarkPdf,
        [bool] $AddDebugMarker
    )

    $PdfReaderType = [iText.Kernel.Pdf.PdfReader]
    $PdfWriterType = [iText.Kernel.Pdf.PdfWriter]
    $PdfDocType = [iText.Kernel.Pdf.PdfDocument]
    $PdfCanvasType = [iText.Kernel.Pdf.Canvas.PdfCanvas]

    $reader=$null; $writer=$null; $pdf=$null
    $wmReader=$null; $wmDoc=$null

    try {

```

```

$reader = New-Object $PdfReaderType($InputPdf)
$writer = New-Object $PdfWriterType($OutputPdf)
$pdf    = New-Object $PdfDocType($reader, $writer)
$pages  = $pdf.GetNumberOfPages()

$wmReader = New-Object $PdfReaderType($WatermarkPdf)
$wmDoc    = New-Object $PdfDocType($wmReader)
$wmPages  = $wmDoc.GetNumberOfPages()

Write-Log ("INFO InputPages={0} WatermarkPages={1} WM={2}" -f $pages, $wmPages,
$WatermarkPdf)

if ($wmPages -lt 1) { throw "Wasserzeichen-PDF hat keine Seiten: $WatermarkPdf" }

$wmFirstXObject = $wmDoc.GetPage(1).CopyAsFormXObject($pdf)
$wmNextXObject  = if ($wmPages -ge 2) { $wmDoc.GetPage(2).CopyAsFormXObject($pdf) } else {
$wmFirstXObject }

for ($p=1; $p -le $pages; $p++) {
    $page = $pdf.GetPage($p)
    $canvas = New-Object $PdfCanvasType($page.NewContentStreamBefore(),
$page.GetResources(), $pdf)

    $use = if ($pages -eq 1 -or $p -eq 1) { 1 } else { 2 }
    if ($use -eq 1) { $canvas.AddXObjectAt($wmFirstXObject, 0, 0) | Out-Null }
    else
        { $canvas.AddXObjectAt($wmNextXObject, 0, 0) | Out-Null }

# Debug-Marker: macht sofort sichtbar, ob Seite 2 wirklich benutzt wird
    if ($AddDebugMarker) {
        $canvas2 = New-Object $PdfCanvasType($page.NewContentStreamAfter(),
$page.GetResources(), $pdf)
        $canvas2.BeginText() | Out-Null
        $canvas2.SetFontAndSize([iText.Kernel.Font.PdfFontFactory]::CreateFont(), 8) | Out-
Null
        $canvas2.MoveText(10, 10) | Out-Null
        $canvas2.ShowText(("WM{0}" -f $use)) | Out-Null
        $canvas2.EndText() | Out-Null
        $canvas2.Release()
    }
}

```

```

        $canvas.Release()
    }
}
finally {
    foreach ($o in @($wmDoc,$pdf,$wmReader,$reader,$writer)) {
        try { if ($null -ne $o) { $o.Close() } } catch {}
    }
}
}

try {
    Write-Log ("START Input={0} Out={1}" -f $InputPdf, $OutputPdf)

    if (-not (Test-Path $InputPdf)) { throw "Input-PDF nicht gefunden: $InputPdf" }
    if (-not (Test-Path $WatermarkPdf)) { throw "Wasserzeichen nicht gefunden: $WatermarkPdf" }
    if (-not (Test-Path $ITextDllDir)) { throw "ITextDllDir nicht gefunden: $ITextDllDir" }

    Import-IText7 -DllDir $ITextDllDir

    Add-StationeryWatermark -InputPdf $InputPdf -OutputPdf $OutputPdf -WatermarkPdf
    $WatermarkPdf -AddDebugMarker:$AddDebugMarkerBool

    Write-Log "OK"
    exit 0
}
catch {
    Write-Log ("FAIL :: {0}" -f $_.Exception.Message)
    $e = $_.Exception
    while ($e) {
        Write-Log ("EX: {0}: {1}" -f $e.GetType().FullName, $e.Message)
        $e = $e.InnerException
    }
    exit 2
}
}

```

Dazu brauchen wir dann noch das Ausführungs-Script.

[run\\_once.ps1](#)

```

Set-StrictMode -Version Latest
$ErrorActionPreference = "Stop"

$InputDir      = "C:\PDF\IN"
$ExportDir     = "C:\PDF\OUT"
$ErrorDir      = "C:\PDF\ERROR"
$TempDir       = "C:\PDF\TEMP"

$WatermarkPdf  = "C:\PDF\briefpapier.pdf"      # MUSS 2 Seiten haben (1+2)
$ITextDllDir   = "C:\Tools\itext_bundle\publish"
$WorkerScript  = "C:\PDF\pdf_worker.ps1"

$DeleteOriginalAfterSuccess = $true
$AddDebugMarker = $true # setzt klein "WM1"/"WM2" ins PDF, damit man sieht, ob die richtige
Siete gewählt wurde. Auf $false setzen, wenn fertig

function Initialize-Directories {
    foreach ($d in @($InputDir, $ExportDir, $ErrorDir, $TempDir)) {
        if (-not (Test-Path $d)) { New-Item -ItemType Directory -Path $d | Out-Null }
    }
}

function New-SafeOutputName {
    param([Parameter(Mandatory)] [string] $InputPath, [Parameter(Mandatory)] [string] $OutDir)
    $base = [IO.Path]::GetFileNameWithoutExtension($InputPath)
    $ext  = [IO.Path]::GetExtension($InputPath)
    $out  = Join-Path $OutDir ($base + $ext)
    if (-not (Test-Path $out)) { return $out }
    $ts = Get-Date -Format "yyyyMMdd_HHmss_fff"
    return Join-Path $OutDir ("{0}_{1}{2}" -f $base, $ts, $ext)
}

function Quote-Arg([string]$s) {
    if ($null -eq $s) { return '' }
    '"' + ($s -replace '"', '\\"') + '"'
}

Initialize-Directories

if (-not (Test-Path $WorkerScript)) { throw "Worker-Script fehlt: $WorkerScript" }

```

```

if (-not (Test-Path $WatermarkPdf)) { throw "Wasserzeichen-Datei fehlt: $WatermarkPdf" }
if (-not (Test-Path $ITextDllDir)) { throw "ITextDllDir fehlt: $ITextDllDir" }

$runLog = Join-Path $ErrorDir ("run_{0}.log" -f (Get-Date -Format "yyyyMMdd_HHmss"))
# Logfile garantiert anlegen, damit "keine Log-Datei" ausgeschlossen ist
[System.IO.File]::WriteAllText($runLog, ("[{0}] RUN START`r`n" -f (Get-Date)))

$files = @(Get-ChildItem -LiteralPath $InputDir -Filter "*.pdf" -File -Force | Sort-Object
LastWriteTime)
if ($files.Count -eq 0) {
    Write-Host "Keine PDFs gefunden in: $InputDir"
    Write-Host "Log: $runLog"
    exit 0
}

Write-Host ("Gefunden: {0} PDF(s)" -f $files.Count)
Write-Host "Log: $runLog"

foreach ($file in $files) {
    $tempOut = New-SafeOutputName -InputPath $file.FullName -OutDir $TempDir
    $finalOut = New-SafeOutputName -InputPath $file.FullName -OutDir $ExportDir

    Write-Host ("Worker startet: {0}" -f $file.Name)

    # WICHTIG: Argumente als EIN String mit sicherem Quoting -> keine Param-Zerlegung bei
Leerzeichen
    $argString = @(
        "-NoProfile",
        "-ExecutionPolicy Bypass",
        "-File", (Quote-Arg $WorkerScript),
        "-InputPdf", (Quote-Arg $file.FullName),
        "-OutputPdf", (Quote-Arg $tempOut),
        "-WatermarkPdf", (Quote-Arg $WatermarkPdf),
        "-ITextDllDir", (Quote-Arg $ITextDllDir),
        "-LogFile", (Quote-Arg $runLog),
        "-AddDebugMarker", (Quote-Arg ($AddDebugMarker.ToString()))
    ) -join " "

    $p = Start-Process -FilePath "pwsh.exe" -ArgumentList $argString -Wait -PassThru -
WindowStyle Hidden

```

```
if ($p.ExitCode -eq 0 -and (Test-Path $tempOut)) {
    Move-Item -LiteralPath $tempOut -Destination $finalOut -Force
    if ($DeleteOriginalAfterSuccess) { Remove-Item -LiteralPath $file.FullName -Force }
    Write-Host ("OK: {0}" -f $file.Name)
}
else {
    if (Test-Path $tempOut) { Remove-Item -LiteralPath $tempOut -Force -ErrorAction
SilentlyContinue }
    $errDest = New-SafeOutputName -InputPath $file.FullName -OutDir $ErrorDir
    try { Move-Item -LiteralPath $file.FullName -Destination $errDest -Force } catch {}
    Write-Host ("FAIL (ExitCode {0}): {1}" -f $p.ExitCode, $file.Name)
}
}

Write-Host "Fertig."
Write-Host "Log: $runLog"
```

Hier können in den ersten Zeilen die Pfade angepasst werden.

Diese beiden Scripte müssen in den PDF-Ordner gelegt werden und run\_once.ps1 zur gewünschten Zeit ausgeführt werden, am Besten per Taskplaner.

Das das Script etwas braucht, bis alle PDFs gelesen/geschrieben sind, werden, deshalb werden die Dateien etwas zeitversetzt durch ein separates Script in eine .zip-Datei verpackt und per FTP zu Kohlhammer übertragen.

---

Revision #5

Created 30 January 2026 15:40:45 by Thomas Fried

Updated 5 February 2026 12:08:46 by Thomas Fried