# Script Task in SSIS — What, Why, When, and How (with VS 2022 steps)

## 1. What is the Script Task?

The Script Task is a Control Flow task in SSIS that lets you write small bits of C# (or VB.NET) code to do things the built-in tasks can’t (or can’t do easily). Think of it as a “custom glue” block in your workflow.  
  
Key abilities include:  
- Read/write SSIS variables  
- Work with files/folders using .NET (System.IO)  
- Call APIs / web services  
- Do custom validations and branching (set Dts.TaskResult)  
- Use ADO.NET to run ad-hoc database logic  
- Produce custom logs and events  
  
Not to confuse with Script Component (a Data Flow transformation). Script Task = Control Flow; Script Component = Data Flow.

 **Script Task (Control Flow):** runs **once** when the task executes. Use it to do setup/cleanup and orchestration work—file system, variables, looping, web calls, calling stored procs, creating folders, etc. It **does not touch rows** flowing through a pipeline.

 **Script Component (Data Flow):** lives **inside a Data Flow** and runs **for every buffer of rows** (and you can write code that runs once per row). Use it to create rows (Source), modify rows (Transformation), or consume rows (Destination). It’s for data work, not orchestration.

## 2. Why use the Script Task?

Use it when you need flexibility or logic that’s not available in stock tasks:  
- Conditional logic too complex for expressions  
- Custom file operations (rename with timestamp, pattern-based deletes, integrity checks)  
- Invoking command-line tools, APIs, or libraries  
- Building dynamic values (e.g., URLs, tokens, paths) and pushing them into variables  
- Advanced error handling and custom logging

## 3. When to use it (common scenarios)

- Pre-checks before load: “If today’s file is missing, fail the package and email ops.”  
- File hygiene: Create folders if missing; archive/delete files older than N days.  
- Dynamic naming: Append timestamps to filenames before a File System Task runs.  
- Call a REST API: Download a JSON file to land zone, then process it.  
- Gatekeeping: Read a control table and decide which branch of the workflow to run.

## 4. Real-world use case

Scenario: Daily vendor file gatekeeper  
Your package must:  
1) Check that C:\Drop\Orders\_YYYYMMDD.csv exists  
2) If found, rename it to Orders\_YYYYMMDD\_HHMMSS.csv in C:\Landing  
3) If not found, fail the package with a clear, logged message  
  
Why Script Task?  
You could chain File System Tasks, but the conditional checks + dynamic naming + logging are simpler and clearer in one Script Task.

## 5. Steps in VS 2022

1) Install Visual Studio 2022 + SQL Server Integration Services Projects extension.  
2) File → New → Project → Integration Services Project.  
3) Create variables: User::DropFolder, User::LandingFolder, User::BusinessDate, User::RenamedFilePath.  
4) Add a Script Task, edit ReadOnlyVariables and ReadWriteVariables accordingly.  
5) Write your C# code in the Script Editor (VSTA).  
6) Connect the task to downstream tasks.

## 6. Example 1: File presence check + rename with timestamp

using System;  
using System.IO;  
using Microsoft.SqlServer.Dts.Runtime;  
  
public void Main()  
{  
 try  
 {  
 string drop = (string)Dts.Variables["User::DropFolder"].Value;  
 string landing = (string)Dts.Variables["User::LandingFolder"].Value;  
 string bizDate = (string)Dts.Variables["User::BusinessDate"].Value;  
  
 string expected = Path.Combine(drop, $"Orders\_{bizDate.Replace("-", "")}.csv");  
 if (!File.Exists(expected))  
 {  
 Dts.Events.FireError(0, "ScriptTask",  
 $"Expected file not found: {expected}", string.Empty, 0);  
 Dts.TaskResult = (int)ScriptResults.Failure;  
 return;  
 }  
  
 if (!Directory.Exists(landing))  
 Directory.CreateDirectory(landing);  
  
 string stamp = DateTime.Now.ToString("HHmmss");  
 string newName = Path.Combine(landing, $"Orders\_{bizDate.Replace("-", "")}\_{stamp}.csv");  
  
 File.Move(expected, newName);  
  
 Dts.Variables["User::RenamedFilePath"].Value = newName;  
 Dts.TaskResult = (int)ScriptResults.Success;  
 }  
 catch (Exception ex)  
 {  
 Dts.Events.FireError(0, "ScriptTask", ex.ToString(), string.Empty, 0);  
 Dts.TaskResult = (int)ScriptResults.Failure;  
 }  
}  
  
public enum ScriptResults  
{  
 Success = Microsoft.SqlServer.Dts.Runtime.DTSExecResult.Success,  
 Failure = Microsoft.SqlServer.Dts.Runtime.DTSExecResult.Failure  
}

## 7. Example 2: Call a REST API and save to file

using System;  
using System.IO;  
using System.Net;  
using Microsoft.SqlServer.Dts.Runtime;  
  
public void Main()  
{  
 try  
 {  
 string url = "https://api.example.com/orders/today";  
 string outFile = @"C:\Landing\orders.json";  
  
 using (var wc = new WebClient())  
 {  
 string json = wc.DownloadString(url);  
 Directory.CreateDirectory(Path.GetDirectoryName(outFile));  
 File.WriteAllText(outFile, json);  
 }  
  
 Dts.TaskResult = (int)ScriptResults.Success;  
 }  
 catch (Exception ex)  
 {  
 Dts.Events.FireError(0, "ScriptTask", ex.ToString(), string.Empty, 0);  
 Dts.TaskResult = (int)ScriptResults.Failure;  
 }  
}  
  
public enum ScriptResults  
{  
 Success = Microsoft.SqlServer.Dts.Runtime.DTSExecResult.Success,  
 Failure = Microsoft.SqlServer.Dts.Runtime.DTSExecResult.Failure  
}

## 8. Best Practices

- Use variables for flexibility  
- Fail early with clear logging  
- Keep scripts focused and maintainable  
- Use built-in logging events for visibility  
- Avoid hardcoding paths and sensitive data