Reading and Scraping Adobe PDFs with SSIS

Objective:
To give the ability to open, parse, and process Adobe PDF files destined for either CSV output or writing to a database.

The Thought Process:
I wanted to use a modular approach when creating the SSIS Package to prevent duplicating code that could be coded once and included with other "tasks". There is always going to be an "**open**" and "**writing**" that the other tasks could utilize.

C# and Adobe PDF files:
I researched several C# libraries and other prewritten code that would either open, read, or create other Adobe PDF files. It seemed that most, if not all, of these libraries were created to be utilized in console applications and not SSIS to create other PDF files. I finally came across a library called iTextSharp which I had looked at earlier in the process, but eventually came across a small sample of code that used the library that was very helpful.

How it works:
Currently, there is one entry point and exit point in the Control Flow window as shown below in Figure 1.

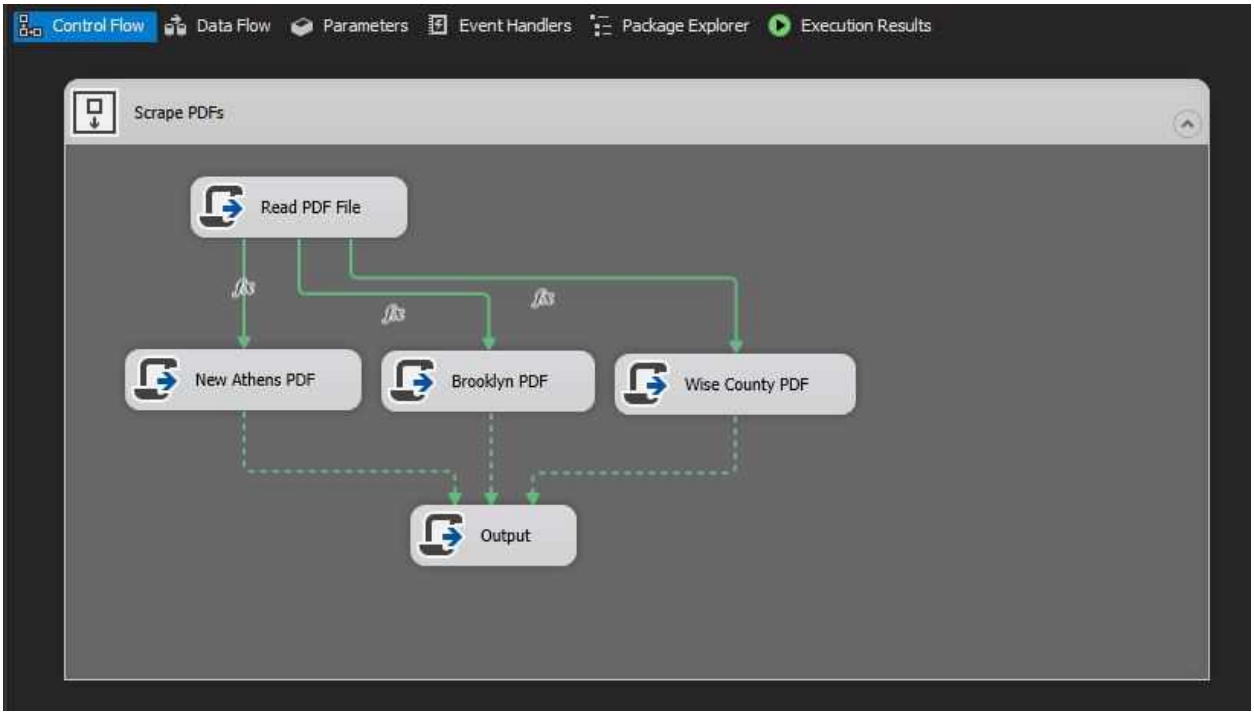

Figure 1. Scrape PDFs Control Flow window

The first task script, **Read PDF File**, opens the PDF file declared in the **Project.params** window, named as **filename**, reads this file line by line, and assigns this data to a text string. This variable can be manually changed or possibly in the future, dynamically. The **Project.params** window is shown below in Figure 2.

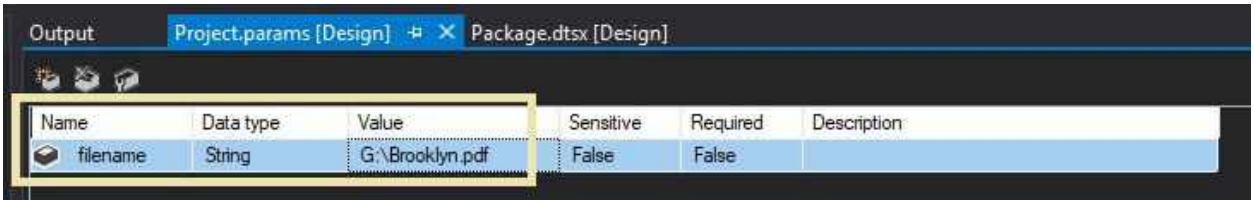

Figure 2. Scrape PDFs Project Params values

This is passed to the first task, Read PDF File, by double clicking the task and changing the **ReadOnlyVariables** in the Script Task Editor as shown below:

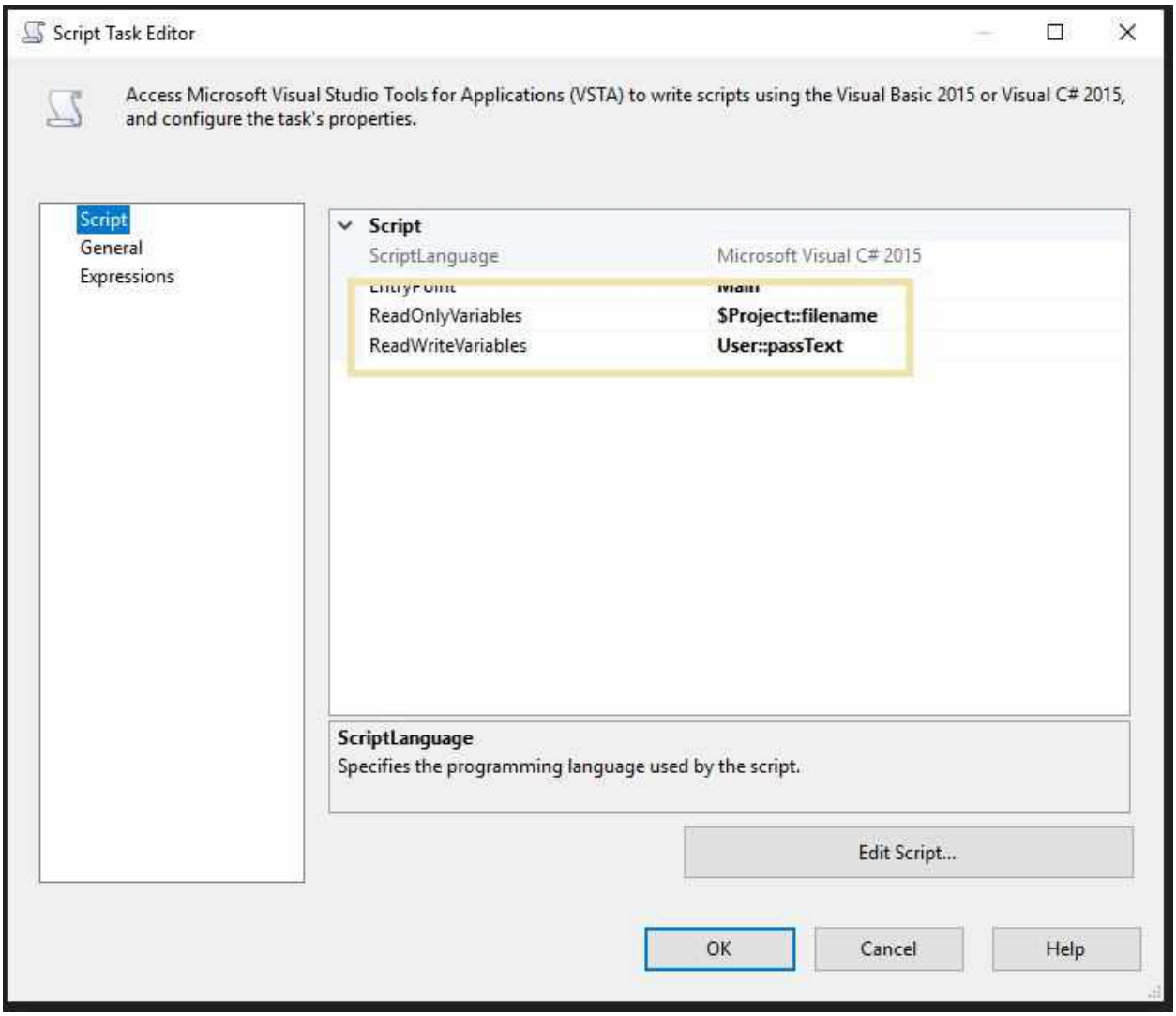

Figure 3. Variables being passed to the first script task

NOTE: The **ReadWriteVariables** option contains the **User::passText** variable which is required as a read/write variable to pass on the PDF lines read in the first script task.

I also programmed a small bit of error checking to inform the user that the file does not exist if this parameter is incorrect, the file does not exist, or the folder does not contain the PDF file. This error happened to me while testing and the program just crashed without any kind of feedback other than an unhelpful SSIS error message. A screen shot of the error message is shown below.

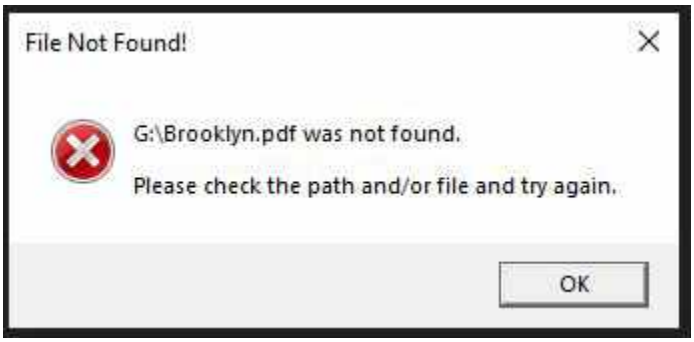

Figure 4. Project.params file not found error

Once the first task has completed and the PDF data has been assigned to a text string a decision has to be made: Which task should be run next to process this data?

Answer: Program an expression in the Precedence Constraint Editor to determine this.

The Precedence Constraint Editor can be located by double clicking the connection line between tasks. The figure below shows the line that can either be double click or clicked once and then choosing "**Edit…**".

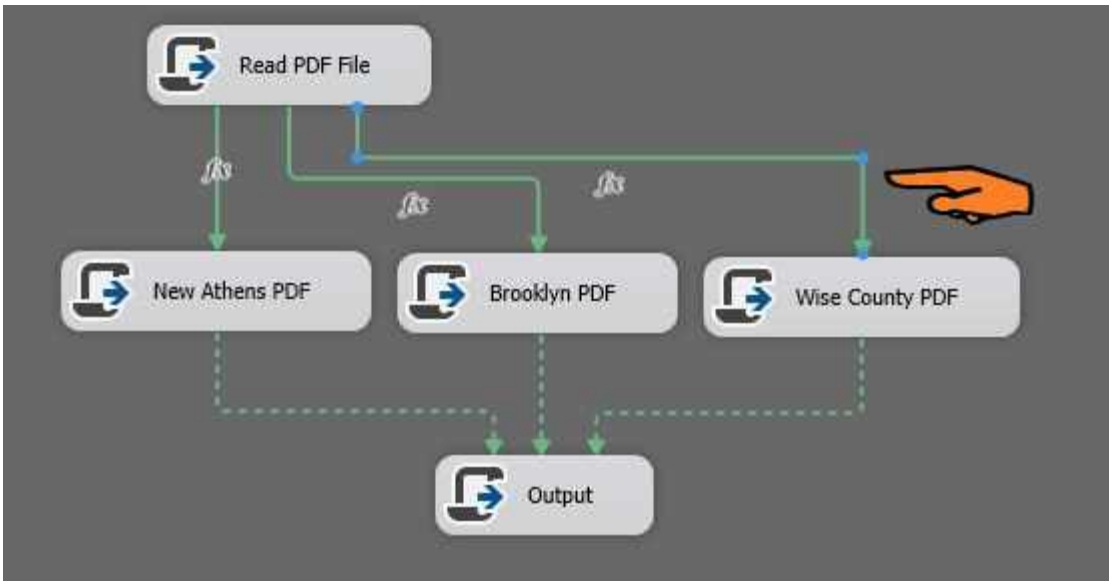

Figure 5. Gaining access to the Precedence Constraint Editor

Two Constraint options must be modified for the process to work: The **Evaluation operation** and the **Expression** itself. These two options are shown below *after* they have been changed.

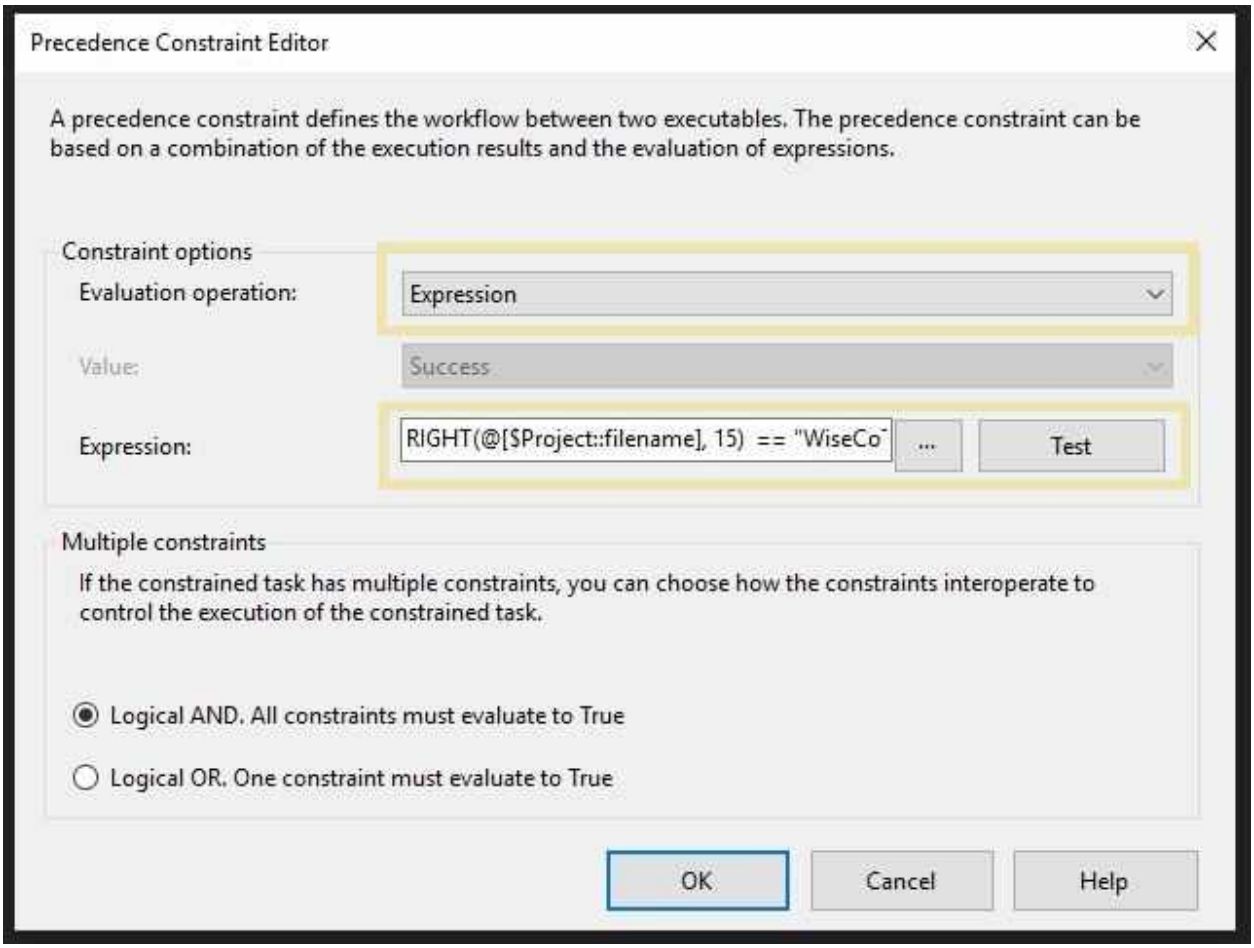

Figure 6. Precedence Constraint Editor showing updated expression information

The Evaluation operation must be changed to: **Expression**. The Expression then must be created and entered. The evaluation I entered merely compares the filename alone and not the complete file name and file path together. The evaluation editor, or Expression Builder, can be accessed by click the "**…**" button.

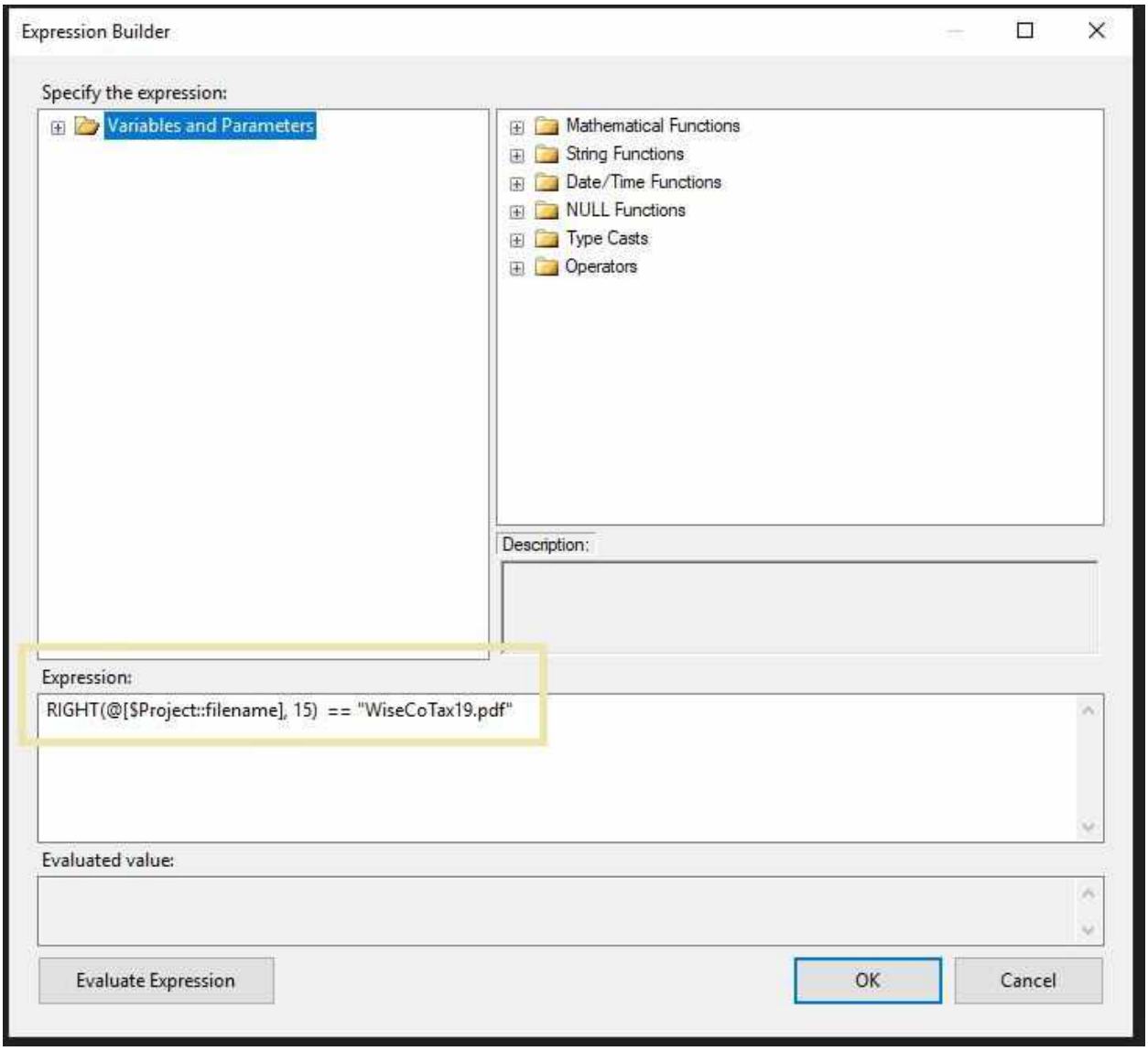

Figure 7. Expression Builder showing completed expression

NOTE: The expression must be modified for each task.

Once the expression has been entered and saved, a small "**fx**" is shown to inform the user that an expression has been entered in the Expression Builder. When the package has been started and this part of the process is evaluated and executed, the individual tasks take over. The PDF text is passed to the correct task by utilizing the **ReadOnlyVariables** option in the Script Task Editor. This window is shown below.

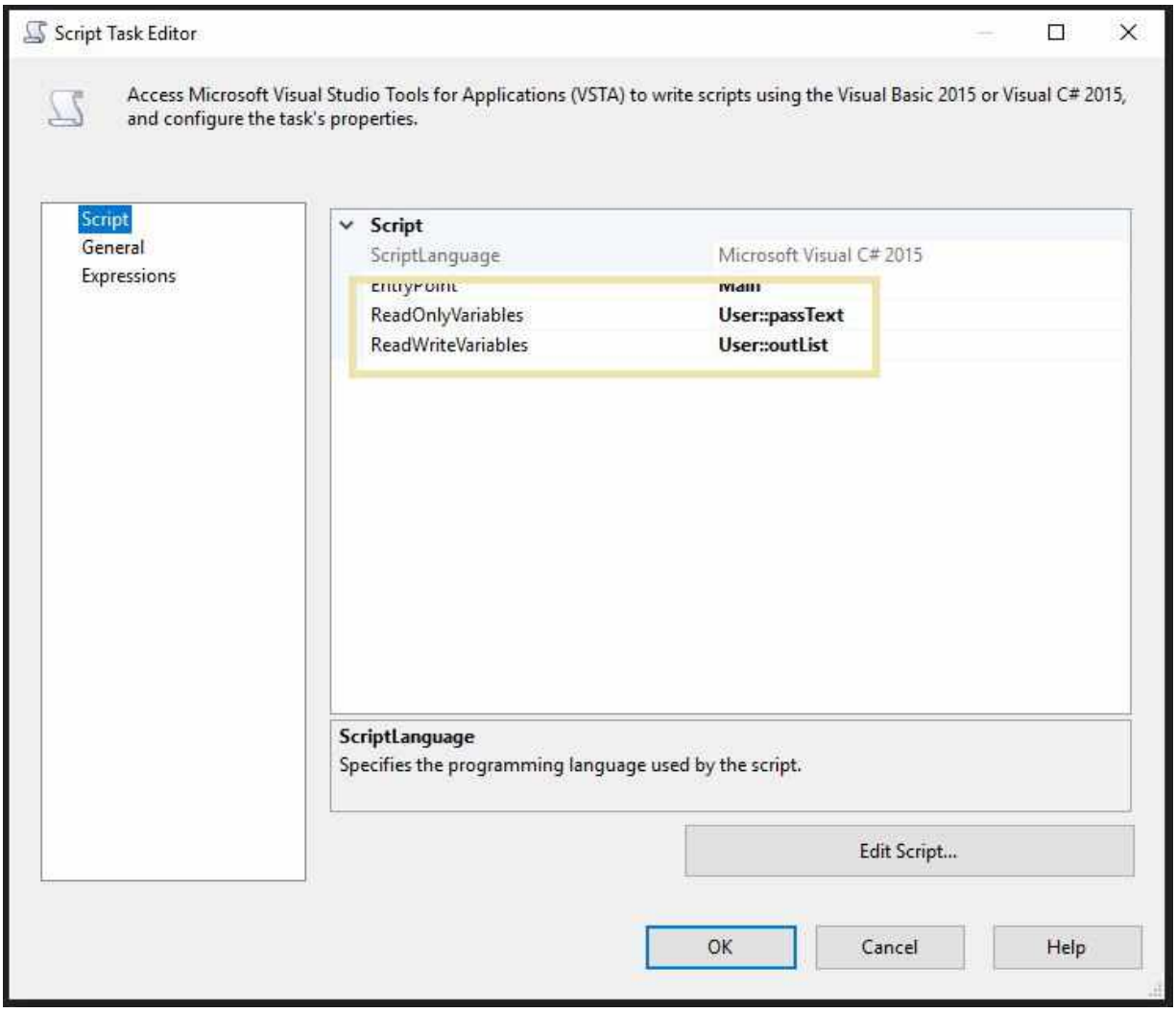

Figure 8. Script Task Editor window showing the variables that are passed between tasks

This is a user defined variable and not a *Project Parameter*. The Variables screen can be accessed by right clicking in an empty area in the Package Design background. The menu is shown below.

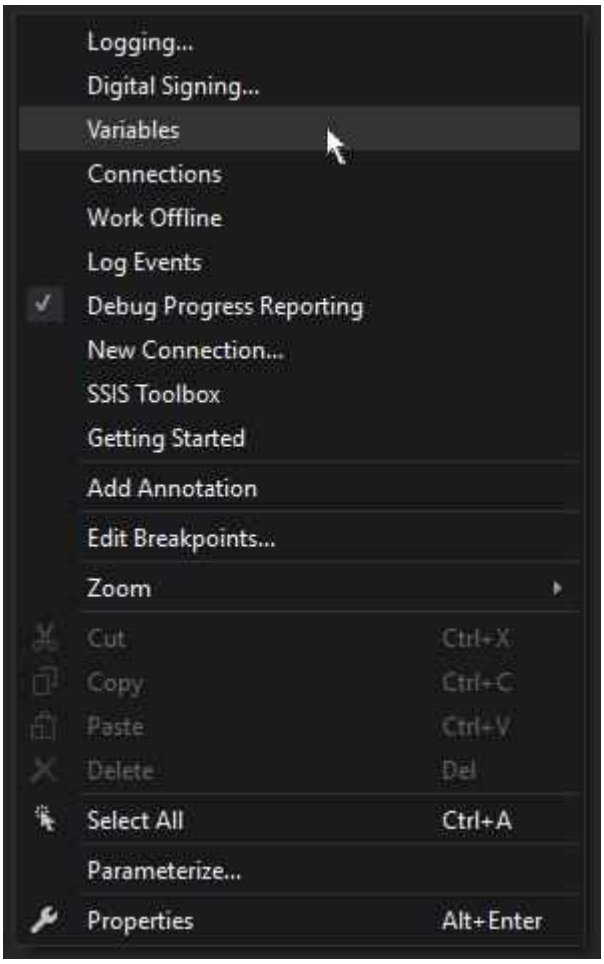

Figure 9. Accessing the Variables List

The first variable I entered was **passText**. This is the text that is read from the PDF and passed to the individual task as a string. The screen is shown in the figure below.

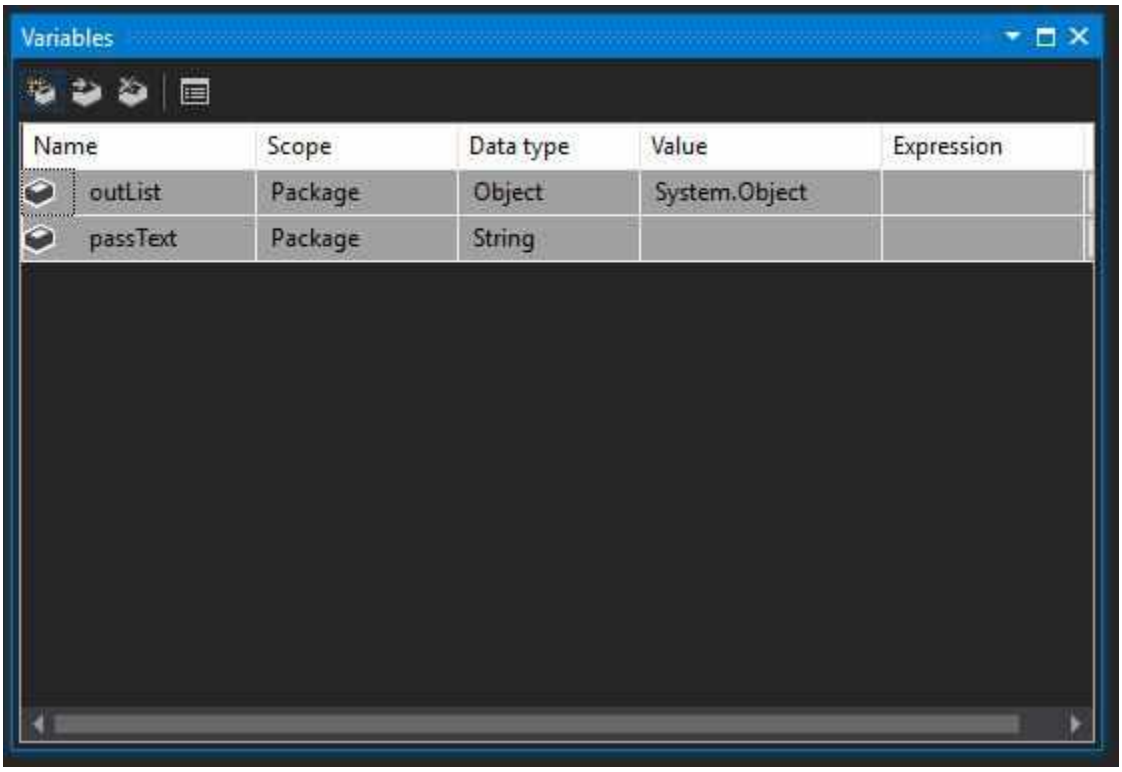

Figure 10. Variables model window

The task now processes the supplied text. Regular expressions, text splitting, and other string functions can be applied here to gather the required results. Two types of variables were my only options when trying to pass this processed information to the final task: A String Array or a String List. String arrays could not be used because when the variable is declared, the size (or number of lines), must be declared at the time of declaration. I went with the String List because this variable type can grow dynamically, and the size of the list does not have to be declared when creating the variable. The list was the only option because the input string would vary, and the length of items could not be known beforehand. Figure 10 also shows the **outList** variable with the Data type as object or **System.Object**. This object is what is passed to the final task: The output task.

Multiple connectors can be made to the output task, but one small change must be made for this to work as programmed or intended: Changing the Precedence Constraint Editor again. As stated above, to gain access to the Precedence Constraint Editor, double click the connection line between tasks or click once and then choose "**Edit…**".

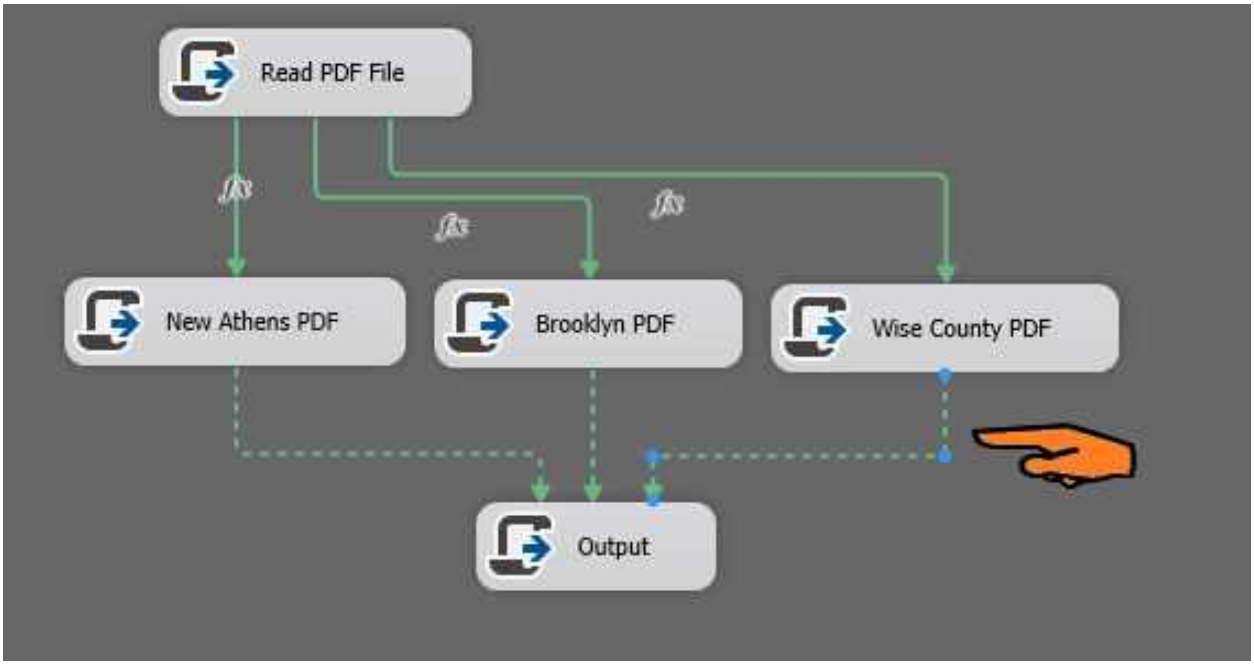

Figure 11. Gaining access to the Precedence Constraint Editor

Only one small change must be made on the Precedence Constraint Editor for the previous task to send output to the output task: Changing the Multiple constraints option. This is shown below in Figure 12.

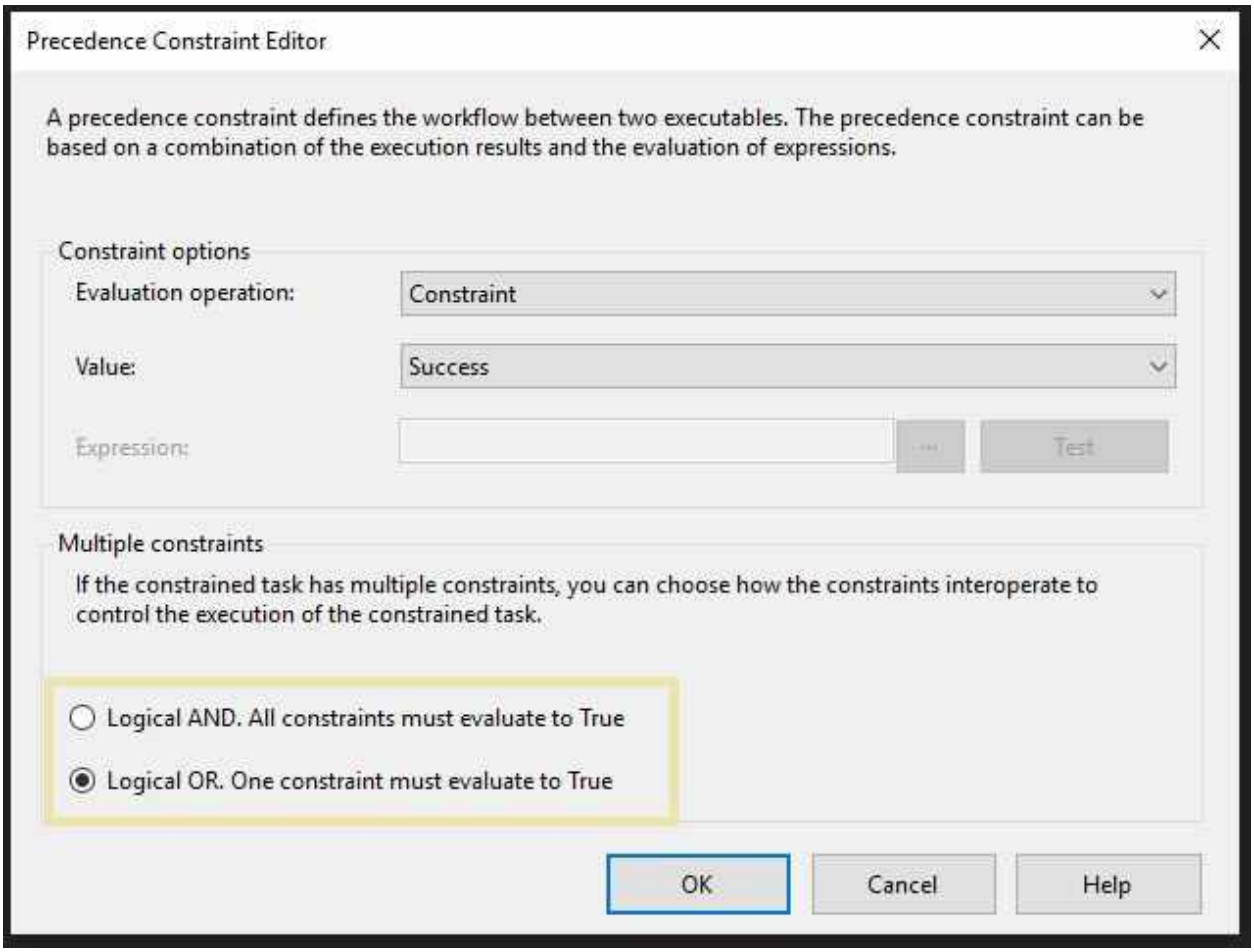

Figure 12. Precedence Constraint Editor modal window

The **Logical AND** (default) option must be changed to **Logical OR**. Only one connector option must be changed for the entire package for the other tasks to send to the output task.

NOTE: Once this changed is made, connector lines change from solid to dashed.

Changes now must be made to the Script Task Editor for the Output task. Only the ReadOnlyVariables option must be modified. The screen shot below shows the change that must be made.

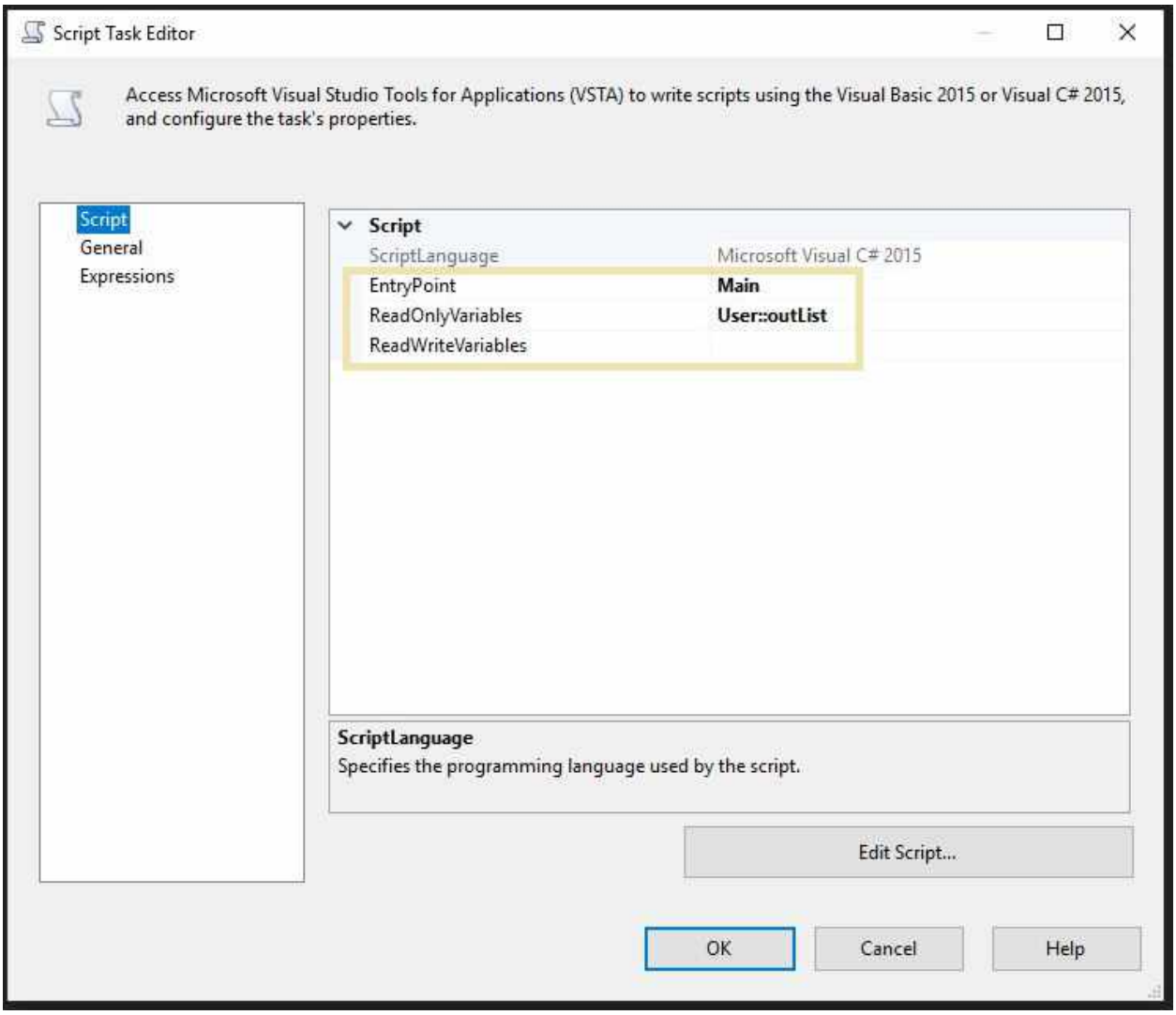

Figure 13. Script Task Editor for the Output Task

The Output task now takes over and processes the information that is passed as the outList object.

As of this writing, the output is merely a MessageBox window showing the lines gathered and that the code is working correctly. This, of course, will be modified in the future to either output a CSV file or inserted in a database table as needed.

Known Issues (Possibly):
The **Read PDF File** task currently reads the PDF information into a single text string. PDF files with many pages would surely stress this variable, run out of memory, and crash the program. One fix would be to read a single PDF page, process this information, read another page, etc. and repeat the process until all the pages in the PDF has been processed. Although now, the PDF files I have tested are small and I have been increasing the page number with each additional task created.

Possible future enhancements:

- Read a folder of similar PDFs and process individually. This would eliminate the need to change the filename for each PDF conversion run. The program could be simply be pointed to a folder and assigned to process each PDF in that folder.
- Read a PDF page at a time for processing, for obvious reasons, as stated above.
- Add a parameter variable or flag to the original filename to designate if the output should be a CSV file or written to a database.
- Add the ability to process multiple PDF files using a flag, such as: the pipe '|' or colon ':' symbol between filenames.