Automation Engine

Scripting in Automation Engine

ESKO

Contents

1. Scripting

You can write Scripts (small programs) to automate the execution of certain tasks during a workflow. Writing and using such scripts is called scripting. You need the Script Runner application to link scripting with your Automation Engine workflows. You can automate actions from Adobe applications (e.g. Illustrator, Photoshop, InDesign) and third party tools (e.g. Alwan) using Scripts.

Some of the instances where scripting is useful are given below:

- to ensure file format integrity (standardization) for workflow inputs.
- to use standardized PDF as input during the Preflight process.
- to automate Adobe Illustrator, Photoshop, InDesign and InDesign Server via ExtendScript on Mac Os and Windows.

Supported Script Types

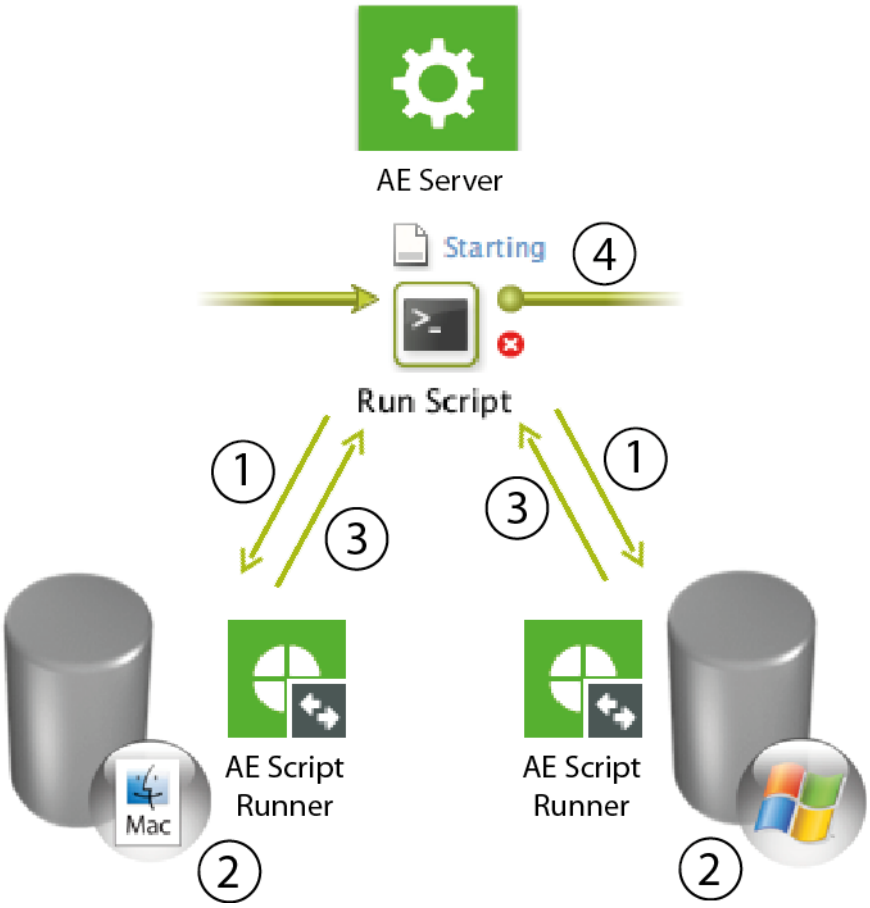
- AppleScript (on Mac OS)
- ShellScript (on Mac OS)
- Batch files (on Windows)
- Windows Script (VBScript, JScript) (on Windows)
- ExtendScript (on Mac OS and Windows)

More info on Scripting :*Getting Started with Scripting*

More info on Scripting in workflows:*Use case: Scripting*

2. Automation Engine Script Runner

The Automation Engine(AE) Script Runner is a standalone AE server component which runs scripts on behalf of AE. You can add customization to your workflow by adding a **Run Script** task. This task will run on the Automation Engine server while the execution of the script will be done on the Script Runner application which can be installed on Windows or Macintosh. When you launch the workflow, the following will take place:

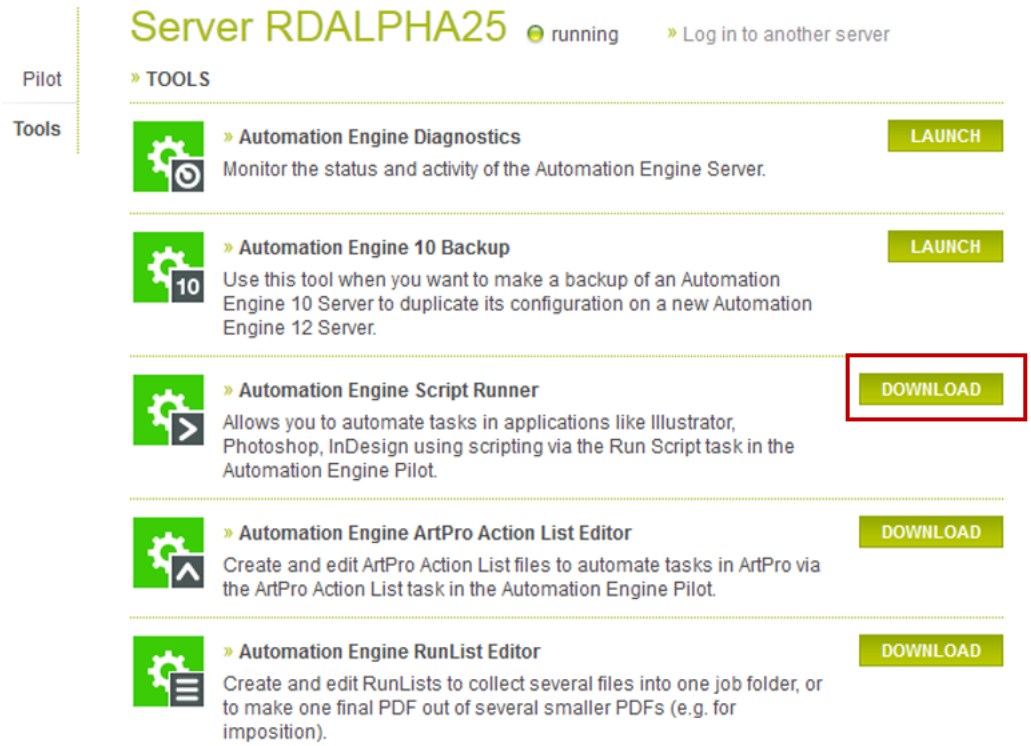


1. The **Run Script** task sends a request to run the specified script.
2. The Script Runner processes the request accordingly and runs the script.
3. The Script Runner sends the results back to the server.
4. The workflow will continue with the outputs from this task.

3. Installing Script Runner

Scripting helps to customize some steps in your work flows. You can achieve this by adding a **Run Script** task to the work flow where you want customization. The Automation Engine will use Script Runner to run scripts which are stored either locally or on a server. When you launch the workflow in Automation Engine, the Script Runner runs the script which contains the main function incorporating the inputs from the Run Script task, output folder and some optional script parameters from the **Run Script** ticket. The workflow in Automation Engine continues with the contents of the output folder which contains the outputs from the script. To achieve this, you need to do the following steps :

1. Download Automation Engine Script runner. You can download it via the web access to the Automation Engine Server. **Client Apps** > **Tools** .



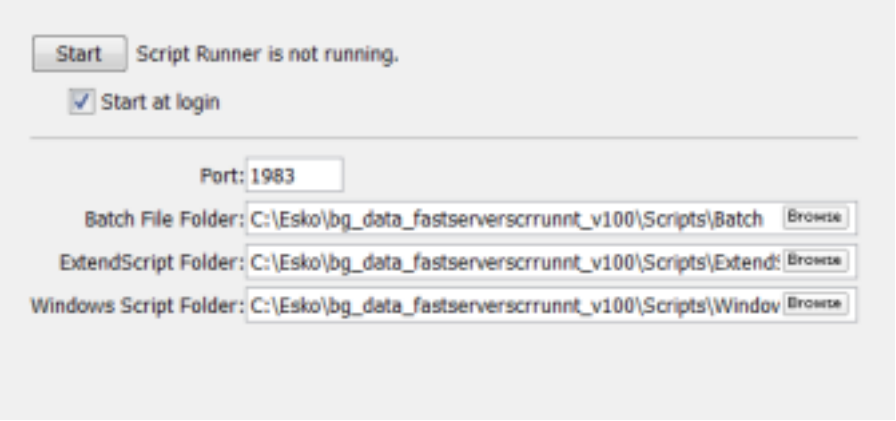
   - On Mac OS, open the ‘dmg’ file after downloading the Script Runner which contains the installer package. Double-click to start the installation.
   - On Windows, double-click the downloaded installer to start the installation.

2. Install the Script Runner by following the instructions in the Installshield Wizard/Assistant and make sure it is running in your computer.

   - On Windows, open **Start** > **All Programs** > **Esko** > **Automation Engine Script Runner** > **Preferences** .
   - On Mac OS, open **Applications** > **Automation Engine Script Runner** > **Esko** > **Automation Engine Script Runner** > **Script Runner Preferences** .

In the Script Runner **Preferences** window, you can:

- check if the Script Runner is actually running
- start / stop the Script Runner
- enable(disable) Start at login
- view and change the port the Script Runner is communicating with
- view and change the default folders for scripts

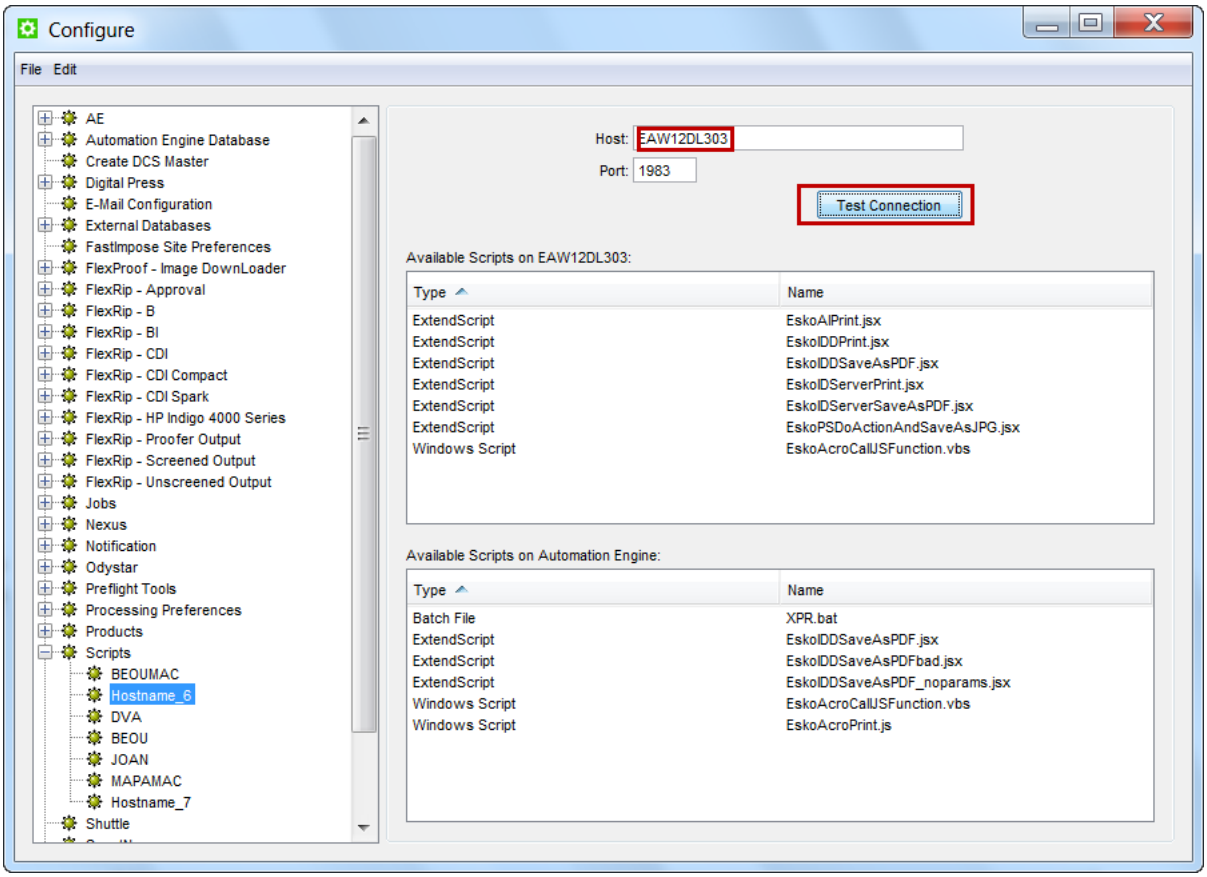


4. Configuring Script Runner

You can configure your Script Runner to your Automation Engine server. This computer must have the Script Runner running.

For more information on installing a Script Runner, refer to *Getting Started with Scripting*

1. Choose **Tools** > **Configure** .
2. Choose **Scripts** from the categories on the left. Choose **File** > **New** .



3. Give a suitable name to the Script Runner by choosing **File** > **Rename**
4. Enter the computer name or IP address of the Script Runner computer in the **Host** field.
5. Enter the **Port** used to connect this computer to your Automation Engine server.

   By default, this is 1983.
6. Click the **Test Connection** button. If you have configured the Scrpt Runner successfully, you will see either all the scripts available on the Script Runner or a message 'No scripts available'.

5. Scripting for Script Runner On Mac OS.

A Script Runner on Mac OS supports AppleScript and Shell Script to automate operations.

Note:

Sample scripts are provided as-is with no warranty of fitness for a particular purpose. These scripts are solely intended to demonstrate techniques for accomplishing common tasks. Additional script logic and error-handling may need to be added to achieve the desired results in your specific environment.

It is up to the user to verify that his intended use of the offered automation functionality is compliant with any third party license agreement and/or other restrictions applicable to any non-Esko products.

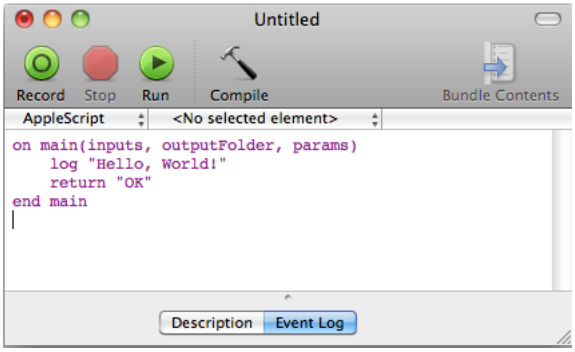
5.1 Using AppleScript on Script Runner

AppleScript is a scripting language that makes direct control of scriptable applications and of many parts of the Mac OS possible. An AppleScriptable application is one that makes its operations and data available in response to AppleScript messages, called Apple events.

We recommend using AppleScript because:

- it is highly integrated into the Mac OS
- it is supported by a lot of third party applications
- it has a high level of accessibility for scripting beginners

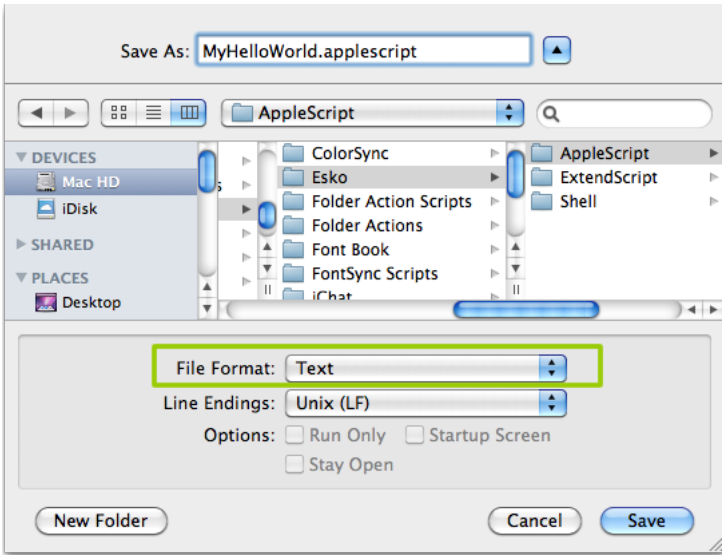
1. Open the AppleScript Editor and add following code.



Options	Description
`main`	This function will be called by the Script Runner. Only the code in this main function gets executed.
`inputs`	The first argument of the main function: a list of input file paths (type: list of strings).

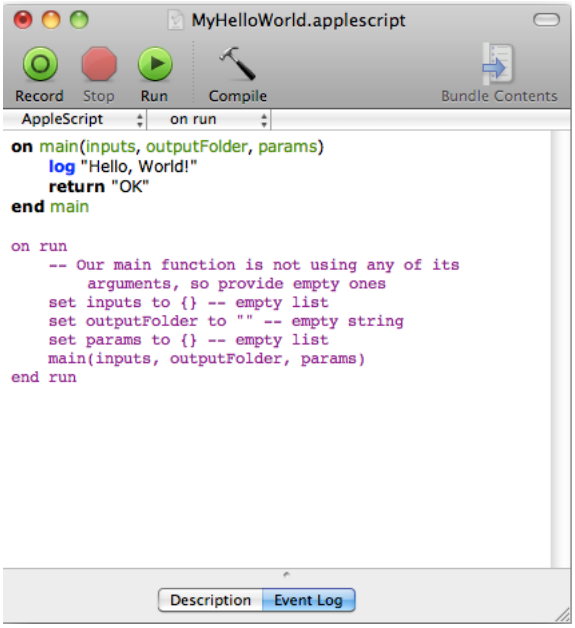
Options	Description
`outputFolder`	Second argument of the main function: the folder where AE expects the script's result files. AE will continue the flow with the files you write in this folder. If you leave this folder empty, AE will continue the flow with the inputs of the Run Script task (type: string).
`params`	Third argument of the main function: additional script parameters injected into the script via the Run Script ticket (type: list of strings).
`log`	Extra log information in the Run Script task details
`return "OK"`	This will communicate with the **Run Script** task that everything went fine. Other possibilities are `return "Warning"` and `return "Error"`.

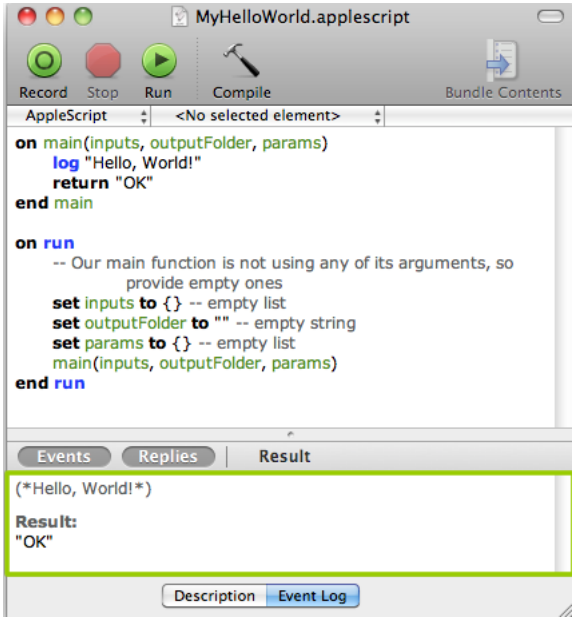
2. Save this code as an AppleScript text file in the Script Runner's AppleScript folder (**default: /Library/Scripts/Esko/AppleScript**) or in the Automation Engine AppleScript folder.



Note: Script Runner supports 'Text' format. Therefore it is essential to change the file format to 'Text'.

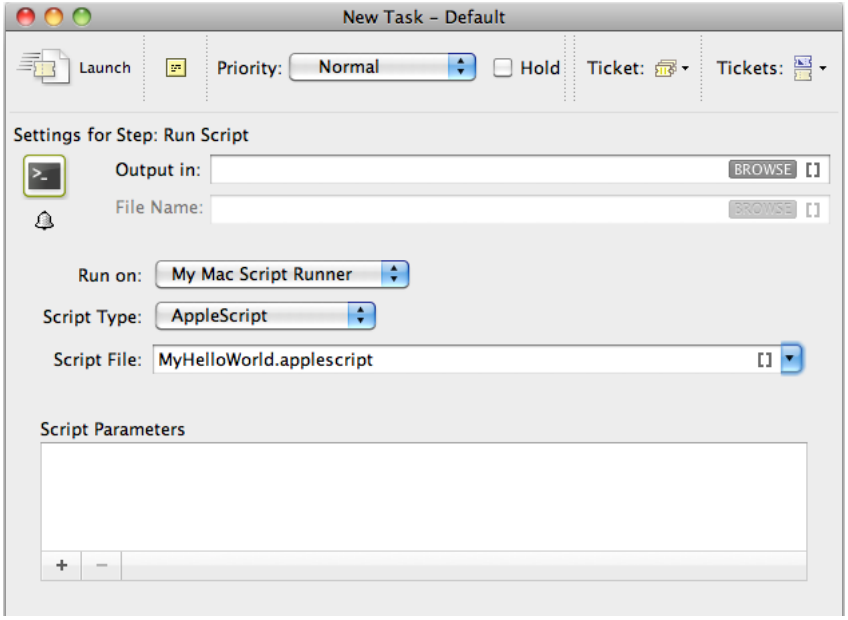
3. You can add following code to test this script locally in the AppleScript Editor. Save the file and click Run to execute the script.





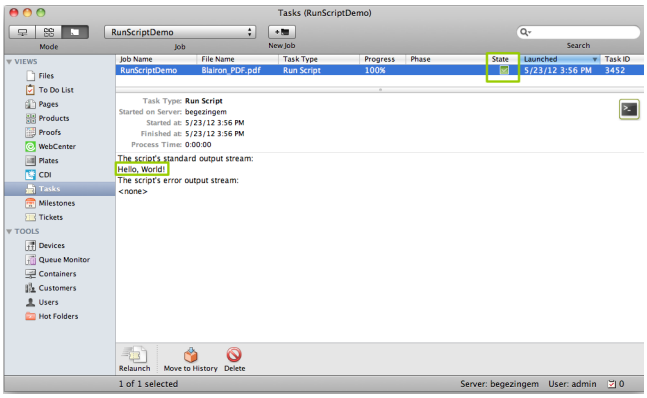
Notice the ‘Hello, World!’ and ‘OK’ result in the event log. The Script Runner does not attend to the test code in your script. It will execute the contents of the main function and ignore the rest. Therefore, you can keep your test code for future local testing.

4. Open the Automation Engine Pilot. Go to **Files** view where you can select a file and open a **New Task**. Choose the **Run Script** task, modify its settings and launch the task.



Read more in *Run Script*

Note that the 'Hello, World!' in the task details and 'OK' state are corresponding with `log "Hello, World!"'` and `'return "OK"'` in the script.

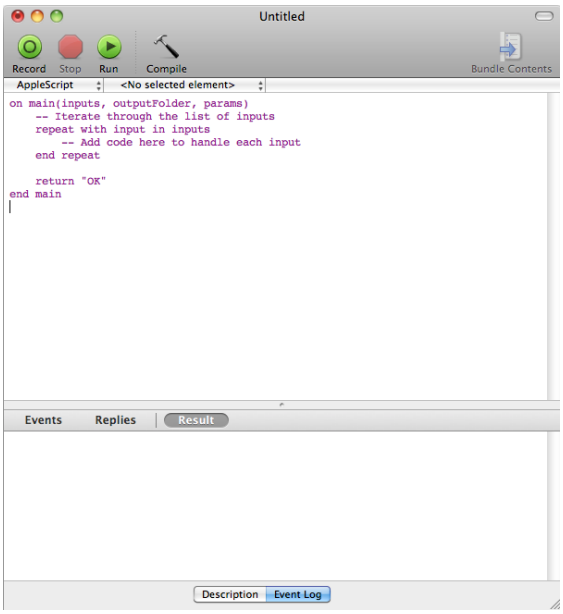


5.1.1 Using AppleScript On Script Runner: A Sample Case

In this example, we use AppleScript to copy every input file with a size smaller than the size specified in the script parameters to the output folder.

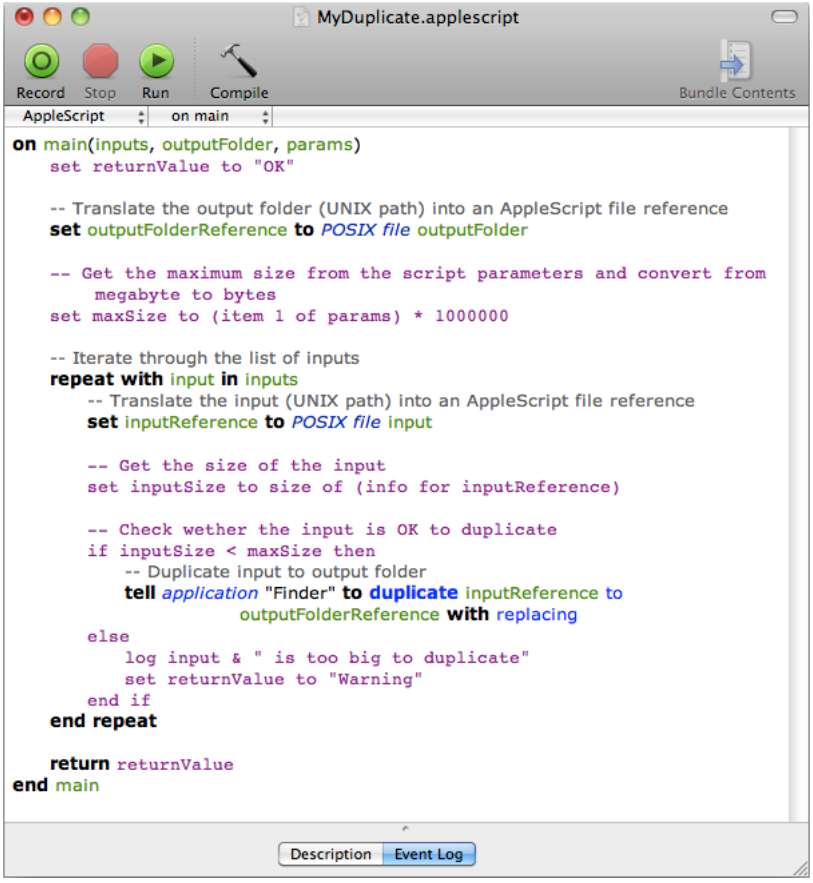
We can use `inputs`, `outputFolder` and `params` in the AppleScript to achieve our objective. First, we demonstrate how to duplicate files without the size restriction and then proceed with the actual case.

1. Open the AppleScript Editor and add the code given below. This code is aimed to iterate through the list of inputs. It enables you to handle the inputs one by one, via the 'input' variable.

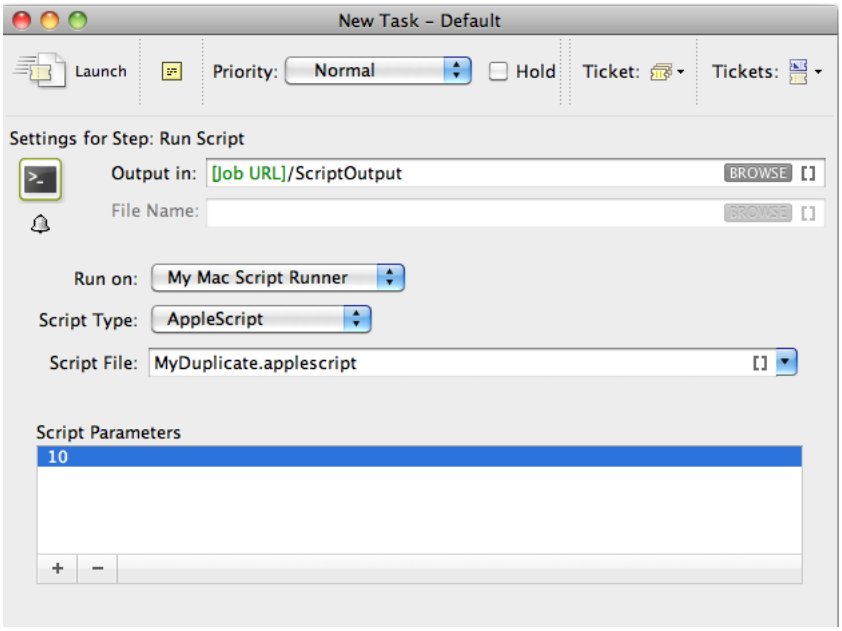


2. You can modify the Script as given below to duplicate the files to a specified output folder without size restrictions. Save this code as an AppleScript text file in the default AppleScript folder of Script Runner(default: **/Library/Scripts/Esko/AppleScript**) or in the Automation Engine AppleScript folder.

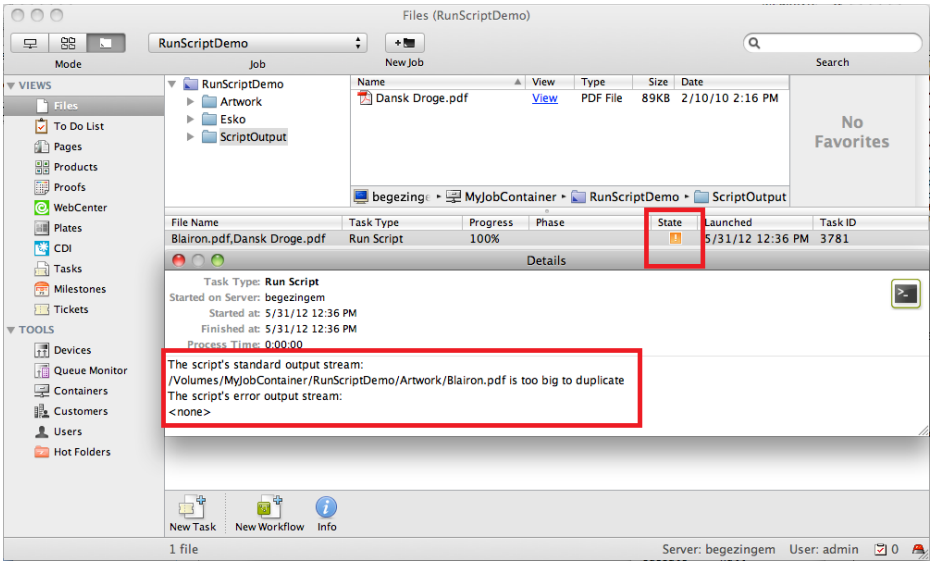
3. Add the file size check in the code as given below. This will duplicate the file when the input file size is smaller than the maximum size from the script parameters. If this condition is not met it will add an entry in the log and there will be "Warning". Save the file.



4. Open the Automation Engine Pilot. Go to **Files** view where you can select the files to be copied and open a **New Task**. Choose the **Run Script** task, modify its settings and launch. This modified ticket will duplicate every selected file which is smaller than 10MB to the current job's Script Output folder. In this example, we excuted this task for two files(Blairon.pdf: 22MB and Dansk Droge.pdf: <1MB).



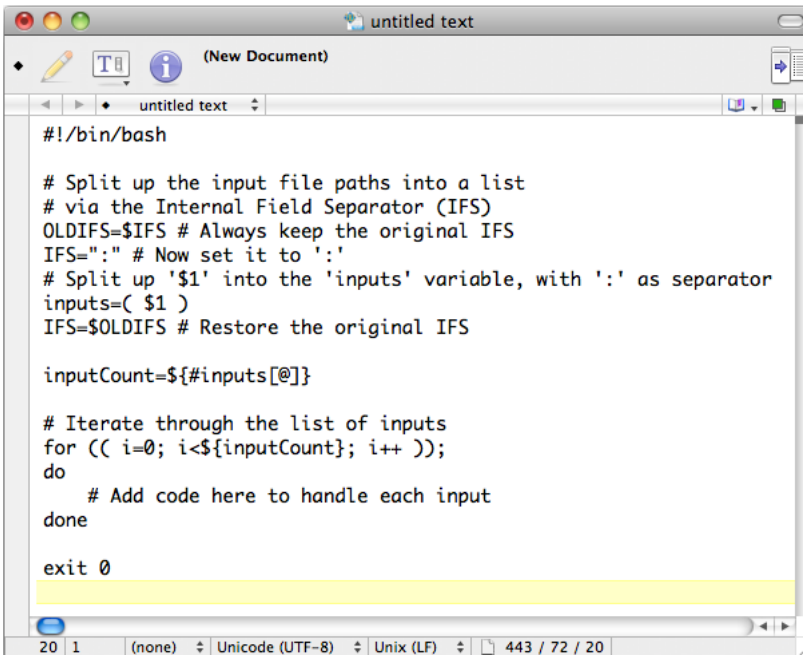
'Dansk Droge.pdf' is duplicated into the job's Script Output folder. 'Blairon.pdf' was too big to duplicate (> 10MB). Therefore, the task ended in 'Warning' state and added an entry in the task details.



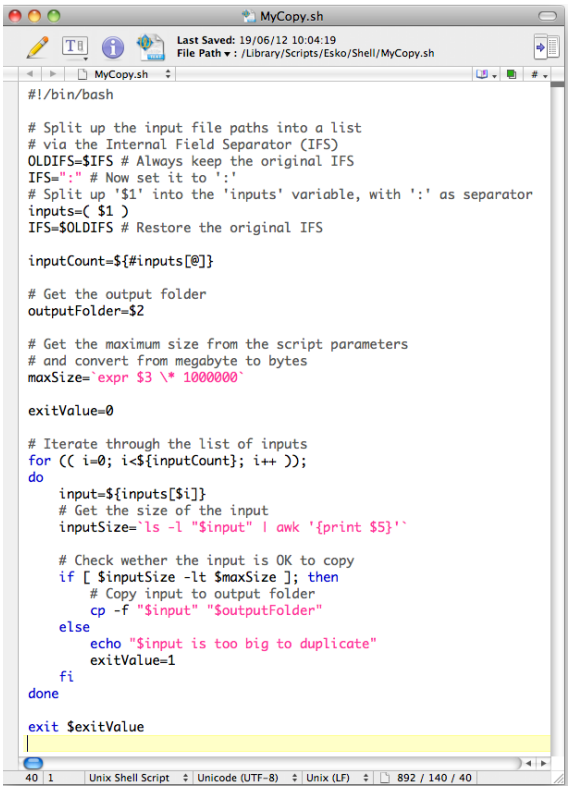
5.2 Using Shell Script on Script Runner

In this example, we use a Shell Script to copy every input file with a size smaller than the specified size in the script parameters to the output folder.

1. Open a text editor and add following code. When the Script Runner executes this code, $1 (the script's first argument) will contain a string of input file paths separated by `:`. The code splits up the concatenated file paths into a real list. This helps to iterate through the list and handle the **Run Script** task's inputs one by one.



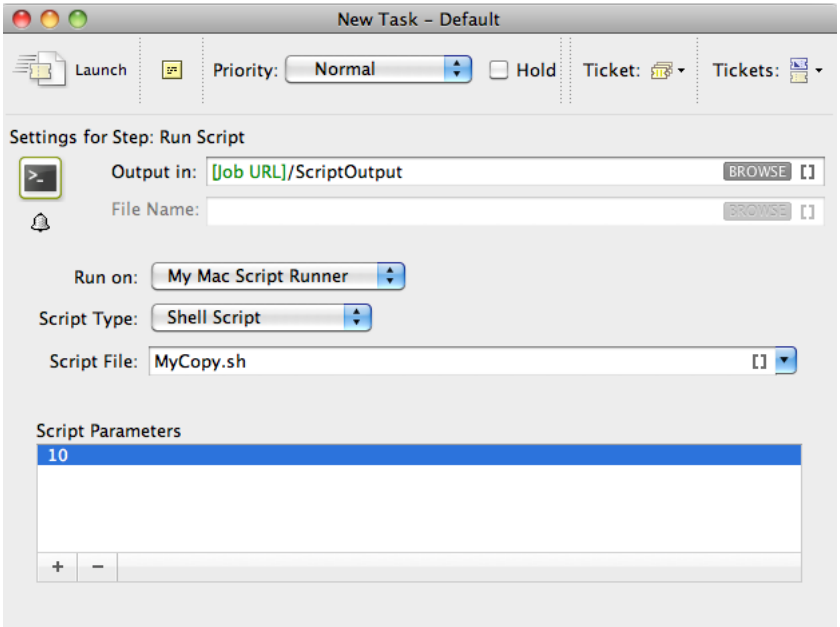
2. Write and save the code as below. This script copies the input to the output folder if the input's file size is smaller than the maximum size from the script parameters. If the size of the file is bigger, it adds an entry in the log and makes sure the task ends in 'Warning' state (via exit value '1'). Save this code as a text file to the Script Runner's Shell folder (default: **/Library/Scripts/Esko/Shell**) or to the Automation Engine Shell folder.



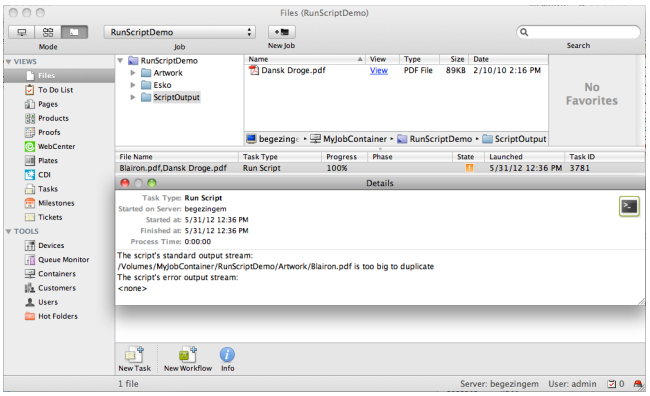
`$1`	First shell script argument: the Run Script task's inputs. A string of input file paths, separated by ':'.
`$2`	Second shell script argument or output folder: This is the folder where Automation Engine expects the script's result files. AE will continue the flow with the files you write in this folder. If you leave this folder empty, AE will continue the flow with the inputs of the Run Script task.
`$3, $4, $5, …`	Remaining shell script arguments: additional script parameters which you can inject into the script via the Run Script ticket.

`exitValue`	Ending Status of the task
`0`	OK
`1`	Warning
`2`	Error

3. Open the Automation Engine Pilot. Go to **Files** view where you can select a file and open a **New Task**. Choose the **Run Script** task, modify its settings and launch. This modified ticket will duplicate every selected file which is smaller than 10MB to the current job's Script Output folder. In this example, we excuted this task for two files(Blairon.pdf: 22MB and Dansk Droge.pdf: <1MB).



'Dansk Droge.pdf' is duplicated into the job's Script Output folder. 'Blairon.pdf' was too big to duplicate (> 10MB). Therefore, the task ended in 'Warning' state (due to `exitValue=1` in the code) and added an entry in the task details.

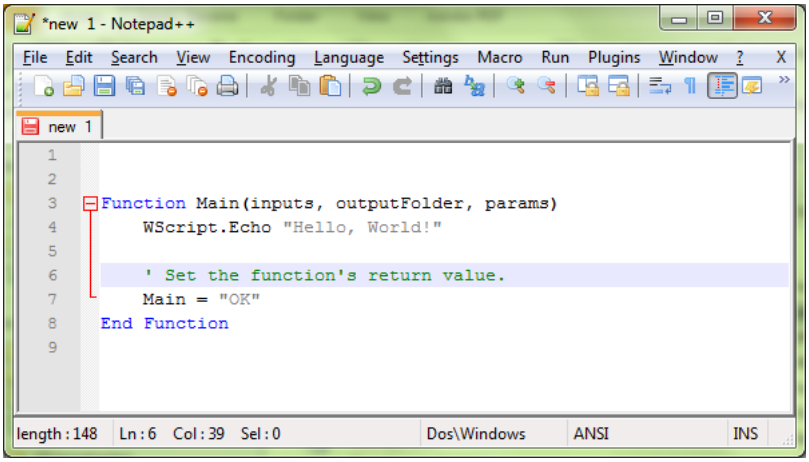

Files (RunScriptDemo)
RunScriptDemo
Mode
Job
New Job
Search
VIEWS
Files
To Do List
Pages
Products
Proofs
WebCenter
Plates
CDI
Tasks
Milestones
Tickets
TOOLS
Devices
Queue Monitor
Containers
Customers
Users
Hot Folders
RunScriptDemo
Artwork
Esko
ScriptOutput
Name
View
Type
Size
Date
Dansk Droge.pdf
View
PDF File
89KB
2/10/10 2:16 PM
No Favorites
begezing
MyJobContainer
RunScriptDemo
ScriptOutput
File Name
Task Type
Progress
Phase
State
Launched
Task ID
Blairon.pdf,Dansk Droge.pdf
Run Script
100%
5/31/12 12:36 PM
3781
Details
Task Type: Run Script
Started on Server: begezingem
Started at: 5/31/12 12:36 PM
Finished at: 5/31/12 12:36 PM
Process Time: 0:00:00
The script's standard output stream:
/Volumes/MyJobContainer/RunScriptDemo/Artwork/Blairon.pdf is too big to duplicate
The script's error output stream:
<none>
New Task
New Workflow
Info
1 file
Server: begezingem
User: admin

6. Scripting for Script Runner on Windows

A Script Runner on Windows supports Windows Script and Batch File. We recommend Windows Script for its scripting abilities comparable to batch files, its wider range of supported features and the simpler syntax. Windows Script is plain-text VBScript or JScript which is interpreted and run by the Windows Script Host.

6.1 Using Windows Script on Script Runner

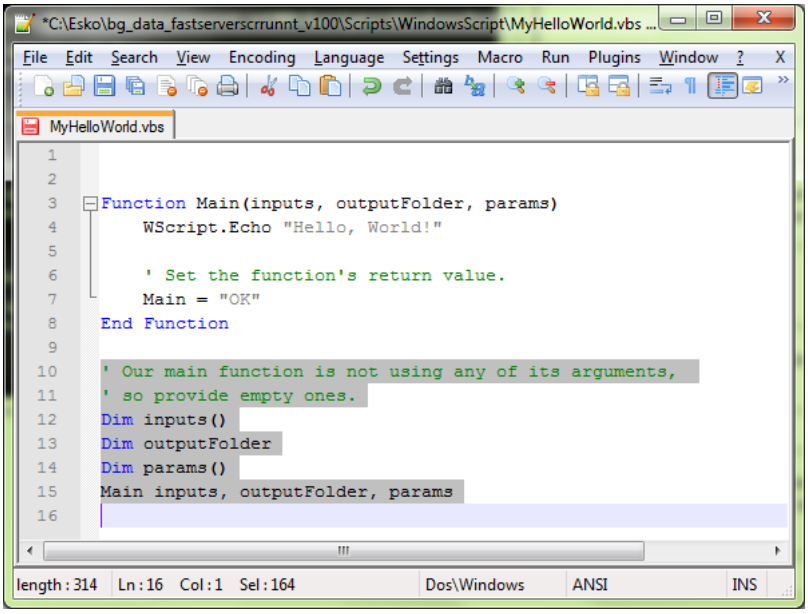
1. Open a text editor and add the following code:



`Function Main`	The function that will be called by the Script Runner. Script Runner executes only the code in this main function.
`inputs`	First argument of main function: a list of input file paths(type: list of strings).
`outputFolder`	Second argument of the main function: the folder where AE expects the script's result files. AE will continue the flow with the files you write in this folder. If you leave this folder empty, AE will continue the flow with the inputs of the Run Script task (type: string).
`params`	Third argument of the main function: additional script parameters injected into the script via the Run Script ticket (type: list of strings).
`WScript.Echo`	This puts some extra log info in the Run Script task details and log. This call prints text to the Console and adds a newline character without Script Runner context.

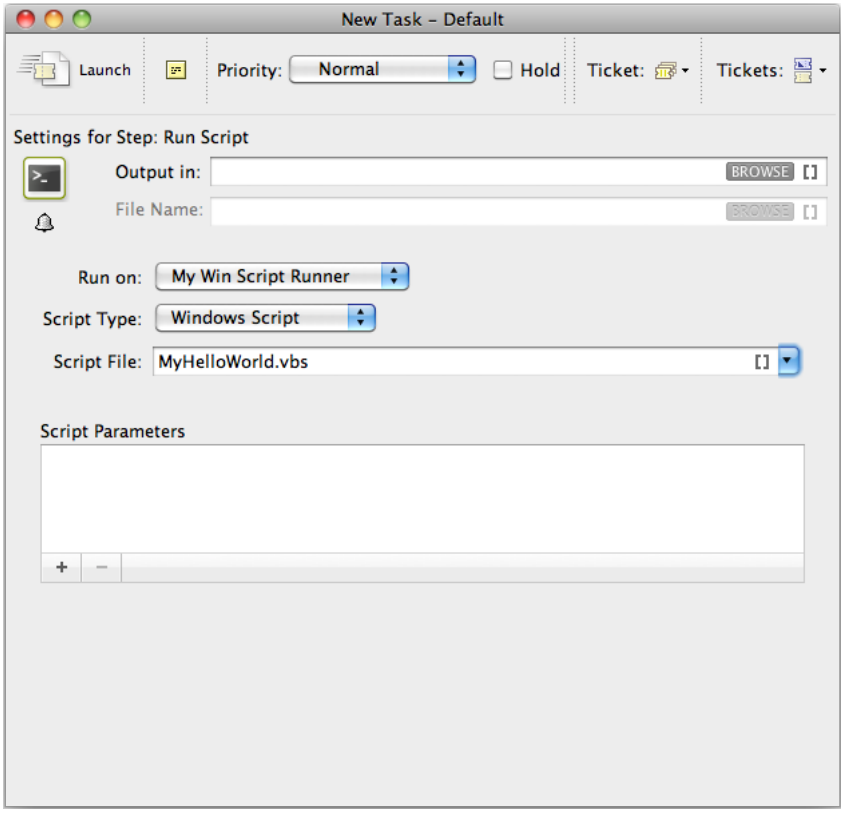
`Main = “OK”`	This will communicate with the **Run Script** task that everything went fine. Other possibilities are `Main = “Warning”` and `Main = “Error”`.

2. You can test this script locally by adding the following code: Save this file. Open `command prompt`. Change the directory to the script’s parent directory. Run command ‘cscript MyHelloWorld.vbs’.

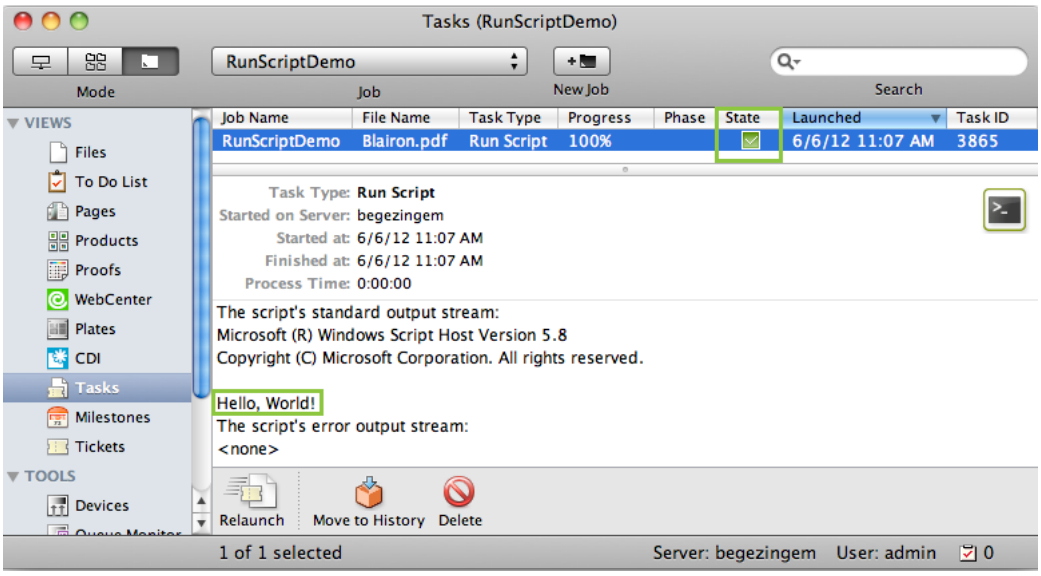


This will produce the output ‘Hello, World!’ to the console. The Script Runner does not attend to the test code in your script. It will execute the contents of the main function and ignore the rest. Therefore, you can keep your test code for future local testing.

3. Open the Automation Engine Pilot. Go to **Files** view where you can select a file and open a **New Task**. Choose the **Run Script** task, modify its settings and launch the task.



Note that the 'Hello, World!' in the task details and 'OK' state are corresponding with `WScript.Echo "Hello, World!"` and `Main = "OK"` in the script.

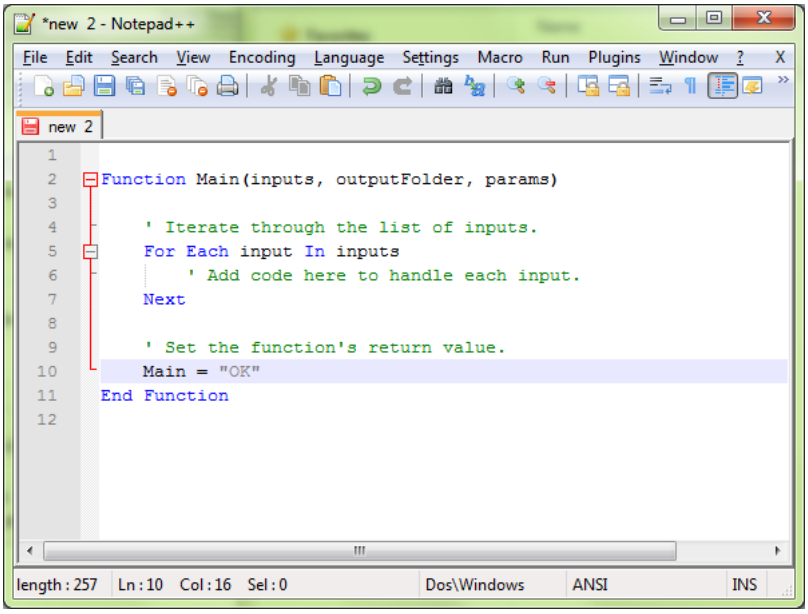


6.1.1 Using Windows Script On Script Runner: A Sample Case

In this example, we use Windows Script to copy every input file with a size smaller than the size specified in the script parameters to the output folder.

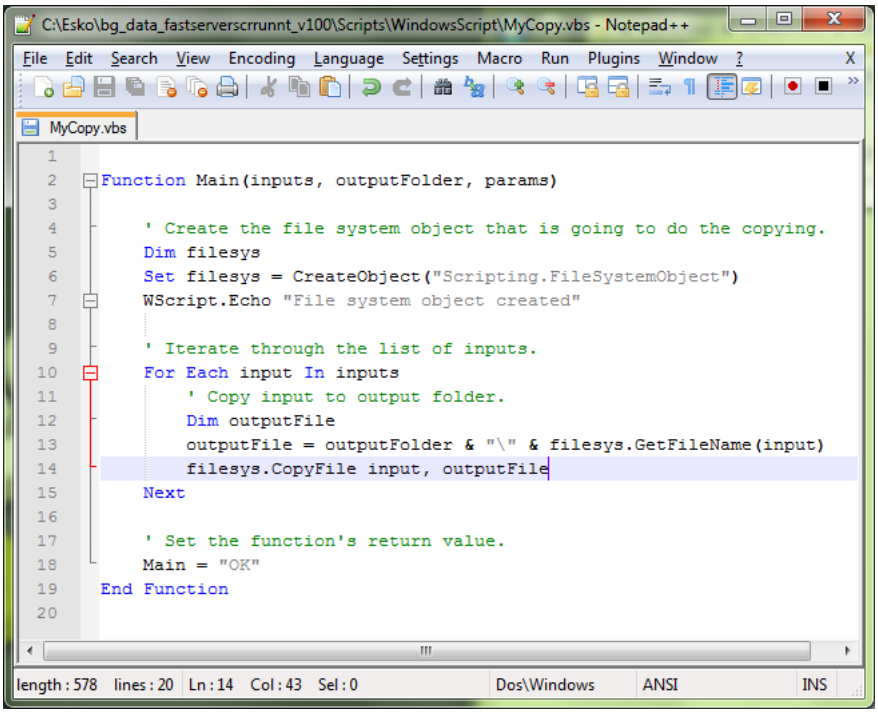
We can use `inputs`, `outputFolder` and `params` in the AppleScript to achieve our objective. First, we demonstrate how to duplicate files without the size restriction and then proceed with the actual use case.

1. Open a text editor and add the code given below. This code is aimed to iterate through the list of inputs. It enables you to handle the inputs one by one, via the 'input' variable.



2. You can modify the Script as given below to duplicate the files to a specified output folder without size restrictions. Save this code as a text file with '.vbs' extension (VBScript) in the Windows

Script folder of Script Runner (default: **C:\Esko\bg_data_fastserverscrrunnt_v100\Scripts\WindowsScript**) or in the Automation Engine WindowsScript folder.



3. Add the file size check in the code as given below. This will duplicate the file when the input file size is smaller than the maximum size from the script parameters. If this condition is not met it will add an entry in the log and there will be "Warning". Save the file.

*C:\Esko\bg_data_fastserverscrrunnt_v100\Scripts\WindowsScript\MyCopy.vbs - Notepad++

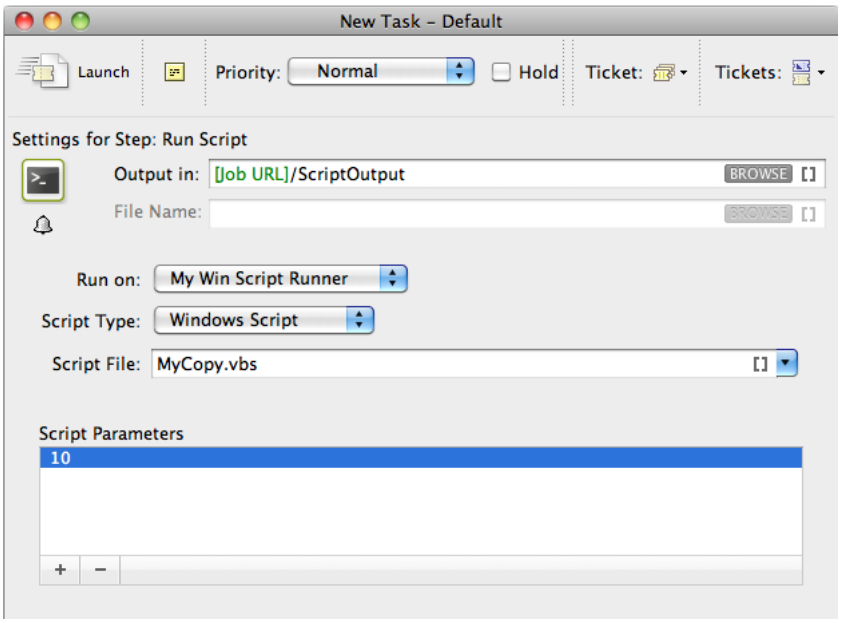
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

MyCopy.vbs

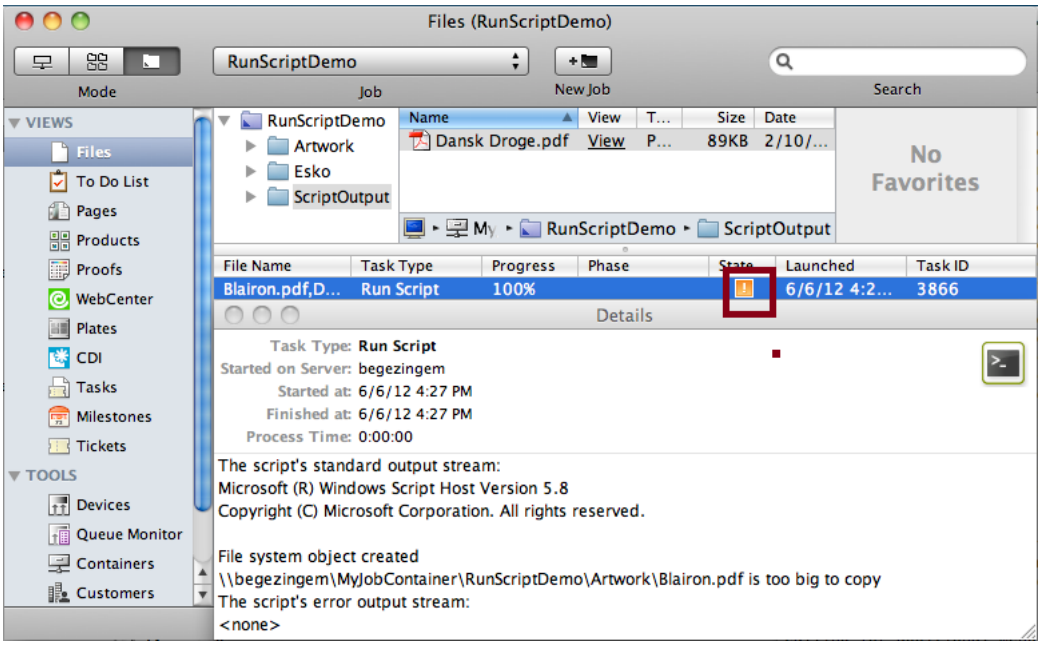
```
Function Main(inputs, outputFolder, params)
    Dim returnValue
    returnValue = "OK"

    ' Get the maximum size from the script parameters
    ' and convert from megabyte to bytes.
    Dim maxSize
    maxSize = params(0) * 1000000

    ' Create the file system object that is going to do the copying.
    Dim filesys
    Set filesys = CreateObject("Scripting.FileSystemObject")
    WScript.Echo "File system object created"

    ' Iterate through the list of inputs.
    For Each input In inputs
        ' Turn the input path into a file object.
        Dim inputReference
        Set inputReference = filesys.GetFile(input)

        ' Check wether the input is OK to copy.
        If inputReference.Size < maxSize Then
            ' Copy input to output folder.
            Dim outputFile
            outputFile = outputFolder & "\" & filesys.GetFileName(input)
            filesys.CopyFile input, outputFile
        Else
            WScript.Echo input & " is too big to copy"
            returnValue = "Warning"
        End If
    Next

    ' Set the function's return value.
    Main = returnValue
End Function
```

length : 1046 lines : 37 Ln : 31 Col : 15 Sel : 0 Dos\Windows ANSI INS

4. Open the Automation Engine Pilot. Go to **Files** view where you can select the files to be copied and open a **New Task**. Choose the **Run Script** task, modify its settings and launch. This modified ticket will duplicate every selected file which is smaller than 10MB to the current job's Script Output folder. In this example, we excuted this task for two files(Blairon.pdf: 22MB and Dansk Droge.pdf: <1MB).



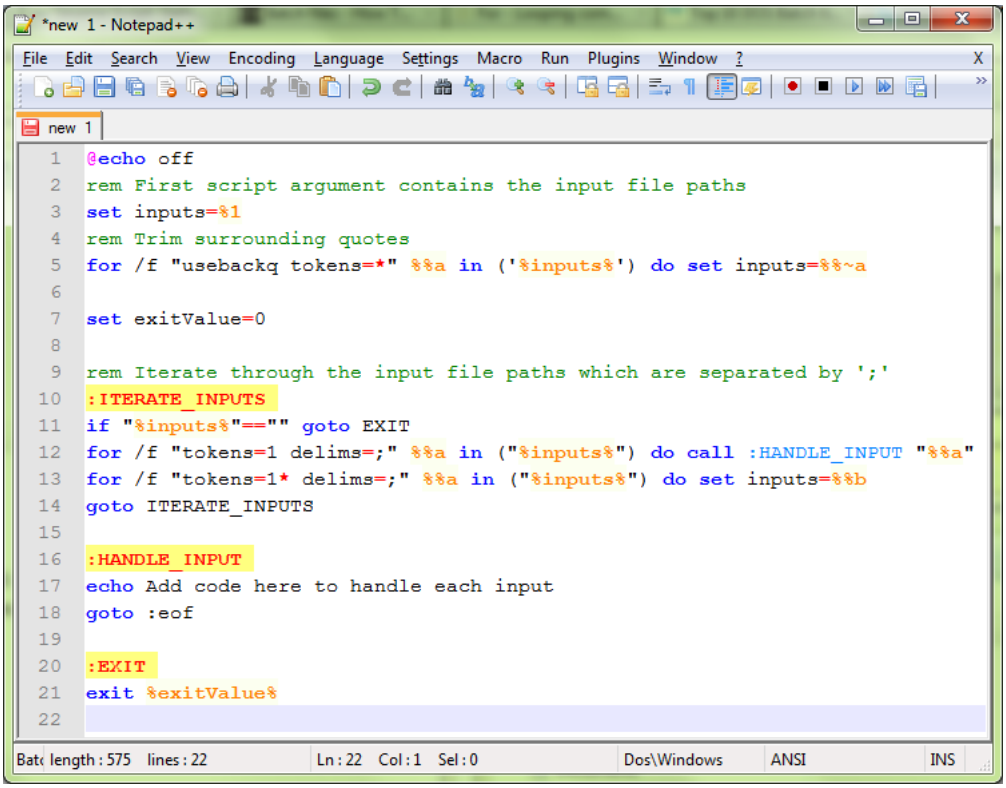
'Dansk Droge.pdf' is duplicated into the job's Script Output folder. 'Blairon.pdf' was too big to duplicate (> 10MB). Therefore, the task ended in 'Warning' state and added an entry in the task details.



6.2 Using Batch File for Scripting on Windows

In this example, we use Batch File to copy every input file with a size smaller than the size specified in the script parameters to the output folder.

1. Open a text editor and add the code given below. This code is aimed to iterate through the list of inputs. It enables you to handle the inputs one by one, via the command %1 (the script's first argument) will contain a string of input file paths, separated by '**;**'.



2. You can modify the Script as given below to duplicate the files to a specified output folder with a size check. Save this code as a text file with '. bat ' extension in the Script Runner Batch File Folder (default: **C:\Esko\bg_data_fastserverscrrunnt_v100\Scripts\BatchFile**) or in the Automation Engine BatchFile folder.

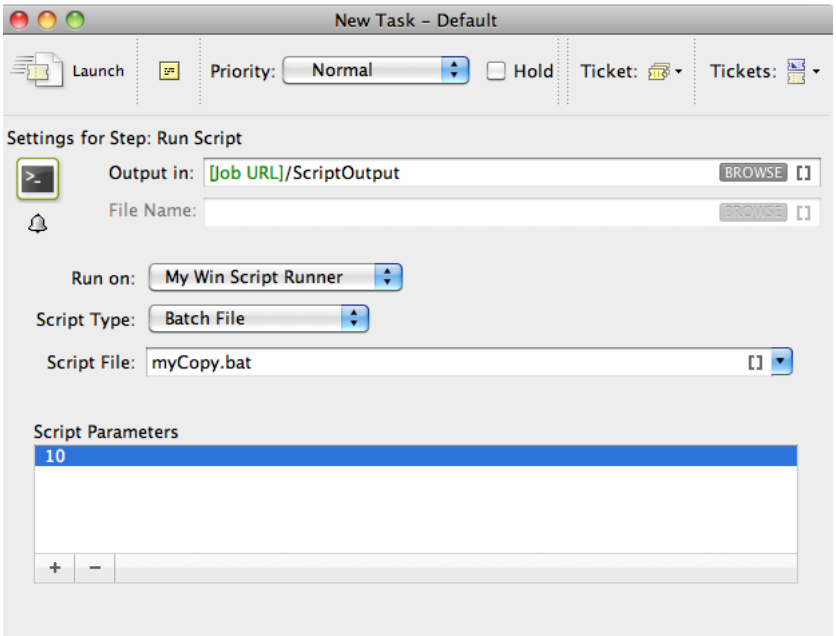
C:\Esko\bg_data_fastserverscrrunnt_v100\Scripts\Batch\myCopy.bat - Notepad++

```
@echo off
rem First script argument contains the input file paths
set inputs=%1
rem Trim surrounding quotes
for /f "usebackq tokens=*" %%a in ('%inputs%') do set inputs=%%~a

rem Get the output folder
set outputFolder=%2

rem Get the maximum size from the script parameters
rem and convert from megabyte to bytes
set /a maxSize=%3*1000000

set exitValue=0

rem Iterate through the input file paths which are separated by ';'
:ITERATE_INPUTS
if "%inputs%"==="" goto EXIT
for /f "tokens=1 delims=;" %%a in ("%inputs%") do call :HANDLE_INPUT "%%a"
for /f "tokens=1* delims=;" %%a in ("%inputs%") do set inputs=%%b
goto ITERATE_INPUTS

:HANDLE_INPUT
rem Get the size of the input
for %%? in (%1) do set inputSize=%%~z?
rem Check wether the input is OK to copy
if %inputSize% lss %maxSize% (
    rem Copy input to output folder
    copy /y %1 %outputFolder%
) else (
    echo %1 is too big to copy
    set exitValue=1
)
goto :eof

:EXIT
exit %exitValue%
```

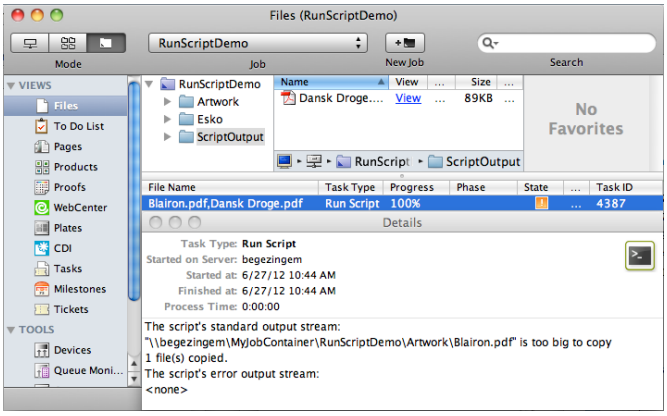
%1	First batch file argument: the Run Script task's inputs. A string of input file paths, separated by ';'.
%2	Second batch file argument or output folder: the folder where AE expects the script's result files. AE will continue the flow with the files you write in this folder. If you leave this folder empty, AE will continue the flow with the inputs of the Run Script task.
%3, %4,%5...	Remaining batch file arguments: additional script parameters, injected into the script via the **Run Script** ticket.

Exit value	Ending Status of the task
0	OK
1	Warning
2	Error

3. Open the Automation Engine Pilot. Go to **Files** view where you can select the files to be copied and open a **New Task**. Choose the **Run Script** task, modify its settings and launch. This modified ticket will duplicate every selected file which is smaller than 10MB to the current job's Script Output folder. In this example, we excuted this task for two files(Blairon.pdf: 22MB and Dansk Droge.pdf: <1MB).



'Dansk Droge.pdf' is duplicated into the job's Script Output folder. 'Blairon.pdf' was too big to duplicate (> 10MB). Therefore, the task ended in 'Warning' state and added an entry in the task details.

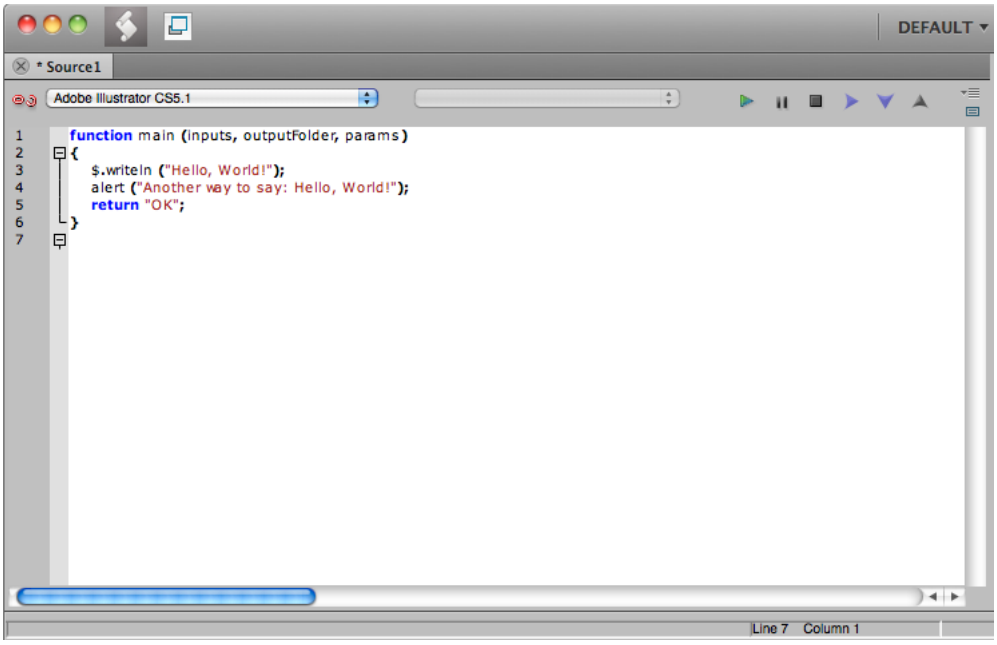


7. Using ExtendScript on Script Runner(On MacOS/Windows)

A Script Runner supports direct interpretation of ExtendScript by the Adobe Creative Suite (CS) application selected in the **Run Script** ticket.

ExtendScript is JavaScript extended for Adobe CS applications. Adobe provides a complete integrated development environment (IDE) for programming ExtendScript which is the ExtendScript Toolkit (ESTK). Latest versions of the ESTK are available with the Creative Suite. For more info and Adobe scripting resources visit the Adobe Scripting Center.

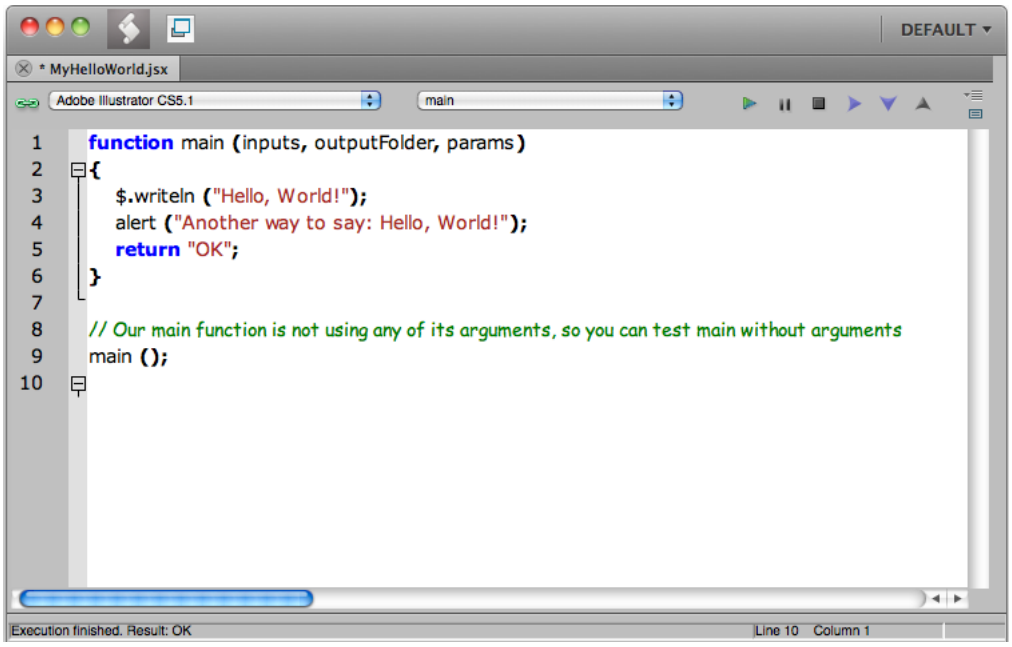
1. Open the ExtendScript Toolkit and add following code. Save this code in the Script Runner's ExtendScript folder. The default location is : **/Library/Scripts/Esko/ExtendScript** for a Script Runner on Mac OS or **C:\Esko\bg_data_fastserverscrrunnt_v100\Scripts\ExtendScript** on Windows. Alternatively, you can save them in the ExtendScript folder of Automation Engine.



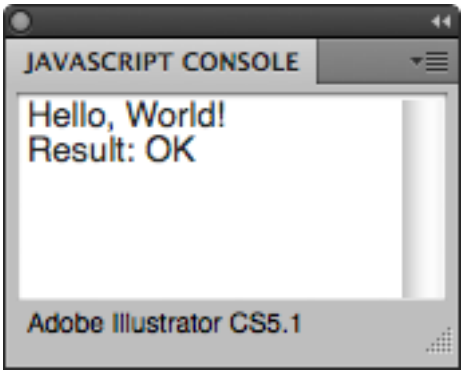
`Main`	This function will be called by the Script Runner. Only the code in this main function gets executed.
`Inputs`	First argument of main function: a list of input file paths (type: list of strings).
`outputFolder`	Second argument of the main function: the folder where AE expects the script's result files. AE will continue the flow with the files you write in this folder. If you leave this folder empty, AE will continue the flow with the inputs of the Run Script task (type: string).

`params`	Third argument of main function: additional script parameters, injected into the script via the **Run Script** ticket. (type: list of strings)
`$.writeln`	This writes extra log information in the Run Script task details and log. Without Script Runner context this call prints text to the Console, and adds a newline character.
`alert`	This registers some extra log info in the **Run Script** task details and log. Without Script Runner context this call displays an alert box.
`return "OK";`	This will communicate with the **Run Script** task that everything went fine. Other possibilities are `Return = "Warning"` and `Return = "Error"`.

2. To test the script locally in the ExtendScript Toolkit, add following code, save and run.

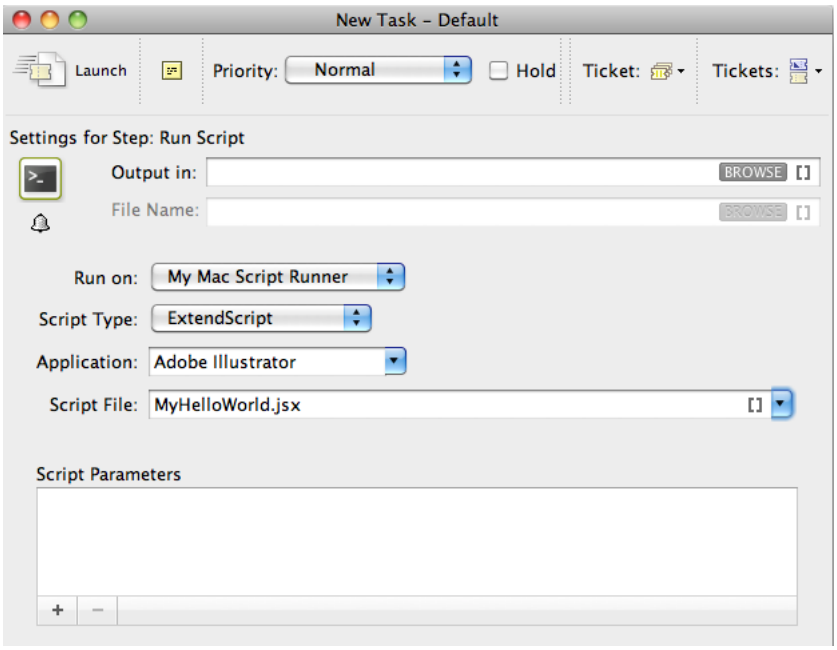


Note that the alert box pops up and the 'Hello, World!' and 'OK' result in the Console.

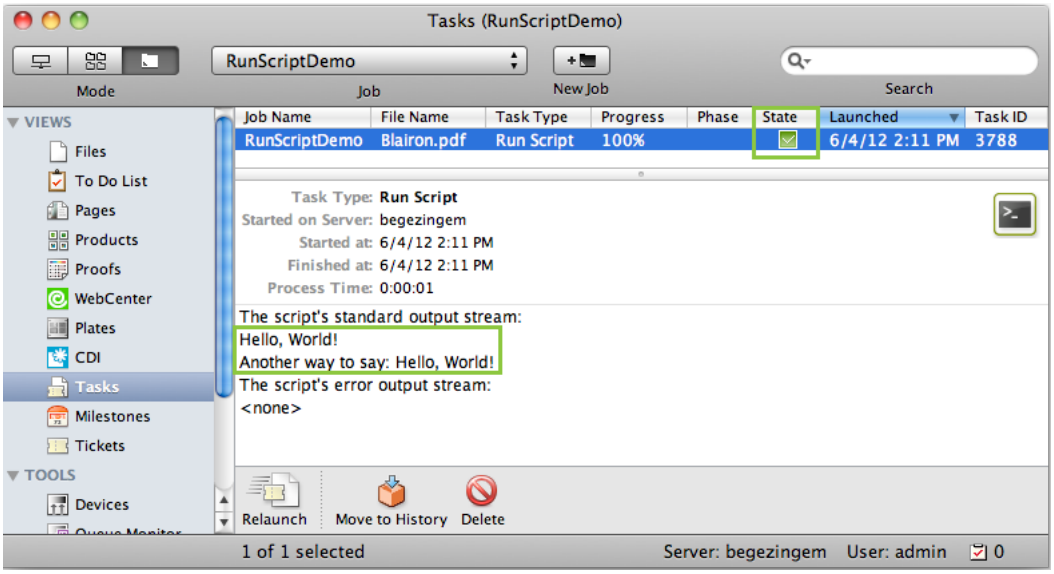


The Script Runner does not attend to the test code in your script. It will execute only the contents of the main function and ignore the rest. Therefore, you can keep your test code for future local testing.

3. Open the Automation Engine Pilot. Go to **Files** view where you can select a file and open a **New Task**. Choose the **Run Script** task, modify its settings and launch the task.



Note that the 'Hello, World!' in the task details and 'OK' state are corresponding with `$.writeln("Hello World!")` and `Return = "OK"` in the script.



7.1 Using ExtendScript on Script Runner: A Sample Case.

In the following example we are going to use ExtendScript to **Open** a file in Illustrator and print it using a **Print Preset**.

Note: When you use ExtendScript on Windows, you can avoid troubles while accessing your user specific settings such as Adobe applications' **Presets** , **Actions** etc by stopping the Script Runner service and running it as an application for the logged in user (who also defined Adobe settings).

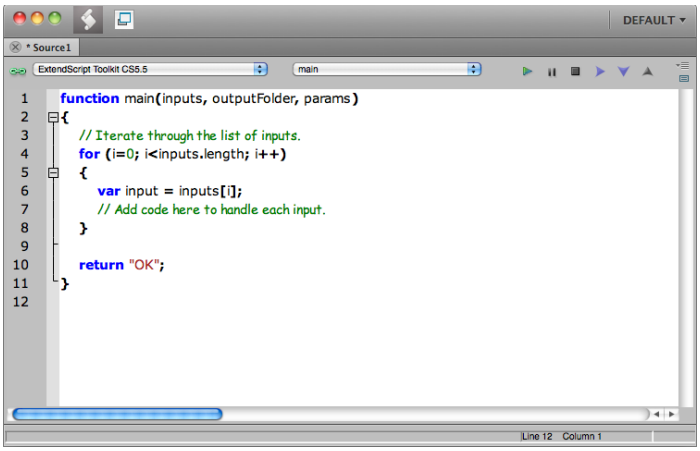
Open **Start** > **All Programs** > **Esko** > **Automation Engine Script Runner** > **Preferences** .

Stop the Script Runner and uncheck **Start at login** (which actually means 'Start as service') and Close **Preferences**.

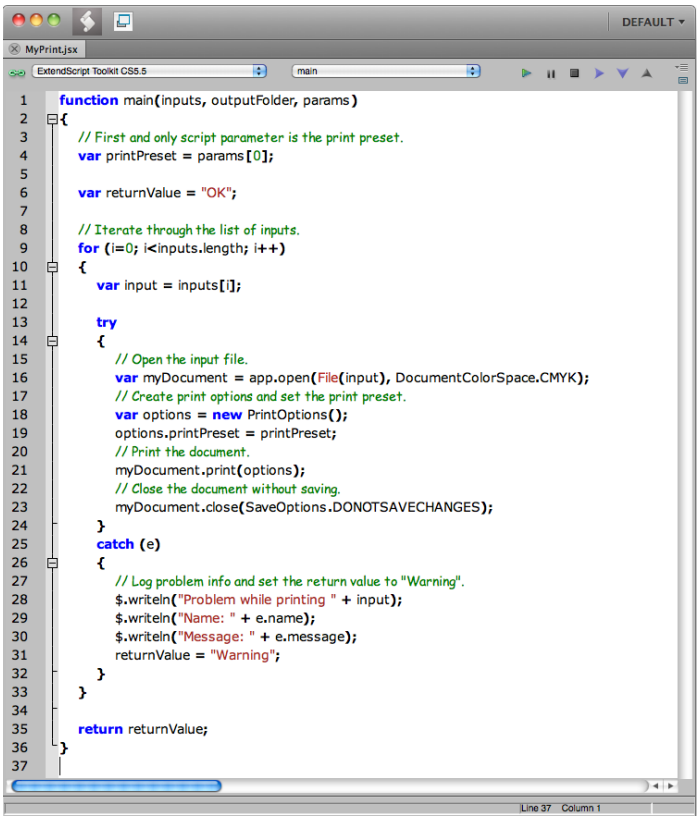
Start as a console application by double clicking its executable: Script Runner's program folder> \bin_ix86\egscrrun.exe (e.g.C:\Esko\bg_prog_fastserverscrrunnt_v120\bin_ix86\egscrrun.exe)

Note: To prevent having to start up the Script Runner every time you log in, add its executable to your user's/system's Startup Items.

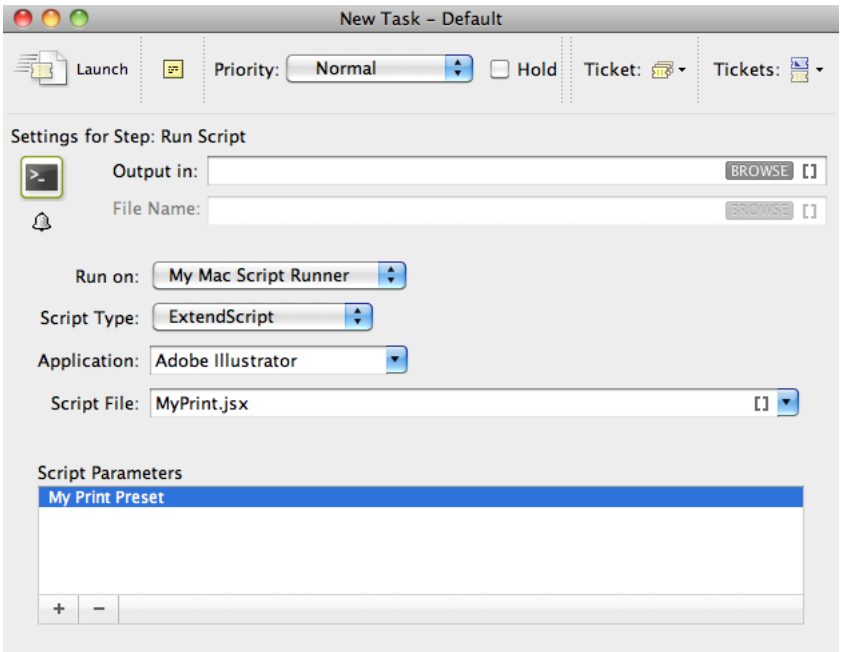
1. Define a **My Print Preset** in your Adobe Illustrator application.
2. Open the ExtendScript Toolkit and add following code. This code is aimed to iterate through the list of inputs. It enables you to handle the inputs one by one, via the `input` variable.



3. To print every input file using a **Print Preset** from the script parameters of the **Run Script** task, write the code as below. Save this code in the default ExtendScript folder(of Script Runner or Automation Engine).



4. Open the Automation Engine Pilot. Go to **Files** view where you can select a file and open a **New Task**. Choose the **Run Script** task, modify its settings and launch the task.



Launching this ticket will result in the selected Illustrator files being printed using the **Print Preset** specified as script parameters (`My Print Preset`) in the task.

8. Appendix : Script Samples

Note:

Sample scripts are provided as-is with no warranty of fitness for a particular purpose. These scripts are solely intended to demonstrate techniques for accomplishing common tasks. Additional script logic and error-handling may need to be added to achieve the desired results in your specific environment.

It is up to the user to verify that his intended use of the offered automation functionality is compliant with any third party license agreement and/or other restrictions applicable to any non-Esko products.

8.1 AppleScript Code Samples

MyHelloWorld.applescript

```
on main(inputs, outputFolder, params)
 log "Hello, World!"
 return "OK"
end main
on run
 -- Our main function is not using any of its arguments, so provide empty ones
 set inputs to {} -- empty list
 set outputFolder to "" -- empty string
 set params to {} -- empty list
 main(inputs, outputFolder, params)
end run
```

MyDuplicate.applescript

```
on main(inputs, outputFolder, params)
 set returnValue to "OK"

 -- Translate the output folder (UNIX path) into an AppleScript file
 reference
 set outputFolderReference to POSIX file outputFolder

 -- Get the maximum size from the script parameters and convert from megabyte
 to bytes
 set maxSize to (item 1 of params) * 1000000

 -- Iterate through the list of inputs
 repeat with input in inputs
  -- Translate the input (UNIX path) into an AppleScript file reference
  set inputReference to POSIX file input

  -- Get the size of the input
  set inputSize to size of (info for inputReference)

  -- Check wether the input is OK to duplicate
```

```
  if inputSize < maxSize then
   -- Duplicate input to output folder
        tell  application  "Finder"  to  duplicate  inputReference  to
 outputFolderReference with replacing
  else
   log input & " is too big to duplicate"
   set returnValue to "Warning"
  end if
 end repeat

 return returnValue
end main
```

8.2 Shell Script Code Sample

MyCopy.sh

```
#!/bin/bash

# Split up the input file paths into a list
# via the Internal Field Separator (IFS)
OLDIFS=$IFS # Always keep the original IFS
IFS=":" # Now set it to ':'
# Split up '$1' into the 'inputs' variable, with ':' as separator
inputs=( $1 )
IFS=$OLDIFS # Restore the original IFS

inputCount=${#inputs[@]}

# Get the output folder
outputFolder=$2

# Get the maximum size from the script parameters
# and convert from megabyte to bytes
maxSize=`expr $3 \* 1000000`

exitValue=0

# Iterate through the list of inputs
for (( i=0; i<${inputCount}; i++ ));
do
 input=${inputs[$i]}
 # Get the size of the input
 inputSize=`ls -l "$input" | awk '{print $5}'`

 # Check wether the input is OK to copy
 if [ $inputSize -lt $maxSize ]; then
  # Copy input to output folder
  cp -f "$input" "$outputFolder"
 else
  echo "$input is too big to duplicate"
  exitValue=1
 fi
done

exit $exitValue
```

8.3 Windows Script Code Samples

MyHelloWorld.vbs

```
Function Main(inputs, outputFolder, params)
 WScript.Echo "Hello, World!"
 ' Set the function's return value.
 Main = "OK"
End Function

' Our main function is not using any of its arguments,
' so provide empty ones.
Dim inputs()
Dim outputFolder
Dim params()
Main inputs, outputFolder, params
```

MyCopy.vbs

```
Function Main(inputs, outputFolder, params)
 Dim returnValue
 returnValue = "OK"

 ' Get the maximum size from the script parameters
 ' and convert from megabyte to bytes.
 Dim maxSize
 maxSize = params(0) * 1000000

 ' Create the file system object that is going to do the copying.
 Dim filesys
 Set filesys = CreateObject("Scripting.FileSystemObject")
 WScript.Echo "File system object created"

 ' Iterate through the list of inputs.
 For Each input In inputs
  ' Turn the input path into a file object.
  Dim inputReference
  Set inputReference = filesys.GetFile(input)

  ' Check wether the input is OK to copy.
  If inputReference.Size < maxSize Then
   ' Copy input to output folder.
   Dim outputFile
   outputFile = outputFolder & "\" & filesys.GetFileName(input)
   filesys.CopyFile input, outputFile
  Else
   WScript.Echo input & " is too big to copy"
   returnValue = "Warning"
  End If
 Next

 ' Set the function's return value.
 Main = returnValue
End Function
```

8.4 Batch File Code Sample

MyCopy.bat

```
@echo off
rem First script argument contains the input file paths
set inputs=%1
rem Trim surrounding quotes
for /f "usebackq tokens=*" %%a in ('%inputs%') do set inputs=%%~a

rem Get the output folder
set outputFolder=%2

rem Get the maximum size from the script parameters
rem and convert from megabyte to bytes
set /a maxSize=%3*1000000

set exitValue=0

rem Iterate through the input file paths which are separated by ';'
:ITERATE_INPUTS
if "%inputs%"==""" goto EXIT
for /f "tokens=1 delims=;" %%a in ("%inputs%") do call :HANDLE_INPUT "%%a"
for /f "tokens=1* delims=;" %%a in ("%inputs%") do set inputs=%%b
goto ITERATE_INPUTS

:HANDLE_INPUT
rem Get the size of the input
for %%? in (%1) do set inputSize=%%~z?
rem Check wether the input is OK to copy
if %inputSize% lss %maxSize% (
 rem Copy input to output folder
 copy /y %1 %outputFolder%
) else (
 echo %1 is too big to copy
 set exitValue=1
)
goto :eof

:EXIT
exit %exitValue%
```

8.5 ExtendScript Code Samples

MyHelloWorld.jsx

```
function main (inputs, outputFolder, params) {
    $.writeln ("Hello, World!");
    alert ("Another way to say: Hello, World!");
    return "OK";
}
// Our main function is not using any of its arguments, so you can test
 main without arguments
main ();
```

MyPrint.jsx

```
function main(inputs, outputFolder, params) {
    // First and only script parameter is the print preset.
    var printPreset = params[0];

    var returnValue = "OK";

    // Iterate through the list of inputs.
    for (i=0; i<inputs.length; i++)
    {
        var input = inputs[i];

        try
        {
            // Open the input file.
          var myDocument = app.open(File(input), DocumentColorSpace.CMYK);
            // Create print options and set the print preset.
            var options = new PrintOptions();
            options.printPreset = printPreset;
            // Print the document.
            myDocument.print(options);
            // Close the document without saving.
            myDocument.close(SaveOptions.DONOTSAVECHANGES);
        }
        catch (e)
        {
            // Log problem info and set the return value to "Warning".
            $.writeln("Problem while printing " + input);
            $.writeln("Name: " + e.name);
            $.writeln("Message: " + e.message);
            returnValue = "Warning";
        }
    }

    return returnValue;
}
```