Automation Engine 22

Running Scripts in Automation Engine

ESKO⬦

Contents

1. Scripting Concept

Automating Desktop and Other Applications

Although Automation Engine already offers a wide set of tools to create custom workflows, it is still possible that you miss some functionality or maybe to support less standard graphics format that you still use or receive.

This is why Automation Engine also offers custom scripting: you can write scripts (small programs) that typically represent actions that operators do interactively in their graphic desktop applications..

Once you created that script, the *Run Script task* enables to insert that task into your normal workflows. This is also how you can automate your desktop applications.

Some examples:

- To automate actions in desktop applications like Adobe Illustrator, Photoshop, InDesign (Server), etc. (via ExtendScript on macOS and Windows).
- To convert files into a format that Automation Engine tasks support. For example converting non-PDF compatible AI files into PDFs.

> ⚠️ **Attention:** 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.

Supported Script Types

- **AppleScript** (macOS), a scripting language created by Apple that facilitates automated control over script-able Mac applications.

- **ShellScript** (macOS), the shell scripting supported by the Mac Terminal.

- **Batch files** (Windows), a Windows script file that consists of a series of commands to be executed by the command-line interpreter.

- **Windows Script** (Windows), a Microsoft VBScript or JScript.

- **ExtendScript** (macOS and Windows), a type of JavaScript enriched with Adobe extensions that can be executed by certain Adobe applications.

  > ⚠️ **Attention:** When scripting Adobe applications, the Script Runner tool on Windows no longer supports 32 bit versions of those applications.

- **PowerShell** (macOS and Windows), a cross-platform task automation solution made up of a command-line shell, a scripting language, and a configuration management framework.

  > ⚠️ **Attention:** When installing PowerShell 7.x on Windows, make sure the option '**Add PowerShell to Path Environment Variable**' is checked.

  > 💡 **Tip:** Using native PowerShell scripts makes it easier to capture an error status of your script than when you run them via a Windows Batch file (a script error will end up on the error output pin of the script task).

The Script Runner Tool

Executing such scripts starts in the *Run Script task*. One of the settings in that task is to choose if the script will be executed by a separate **Script Runner** tool (that you installed earlier on a Mac or Windows client computer), or if the script can run on the Automation Engine server itself (in the 'on-board Script Runner').

When, for example, your script is a Windows batch file, it will be possible to run it on the Automation Engine server itself. But when the script needs to interact with a Mac (Adobe) application, it will require help from a standalone Script Runner that you also installed on that Mac.

Learn more in *The Script Runner Tool*

Extra Information and Tips available as KB articles

Because the area of scripting is often about customization, the examples in this documentation do not cover all use cases.

We therefor advise to also check *Esko's Knowledge Base* for articles on this topic.
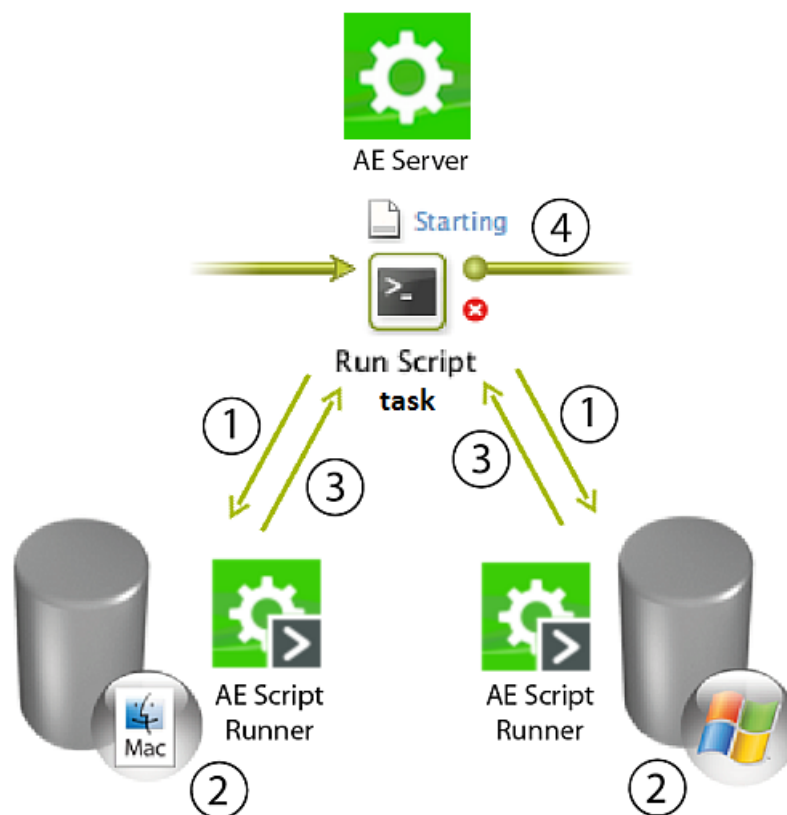
2. The Script Runner Tool

The Automation Engine server itself does not actually run your custom script: the AE task communicates with a tool called **Automation Engine Script Runner**.

> ⚠ **Attention:** As mentioned in *Scripting Concept*, the *'Run Script' task* can run the script on the AE server itself (a Windows Script or batch file) or it could use a standalone Script Runner tool on a Mac or Windows client.

Here 's an overview of the workflow when a standalone AE Script Runner is used:

1. The **Run Script** task sends a request to run the selected script on its input file(s). The task can communicate optional script parameters and also defines the output folder for the resulting file(s).
2. The selected **Script Runner** processes the request and runs the script.
3. The **Script Runner** sends the results back to the Run Script task.
4. The workflow continues with the outputs from this Run Script task.

3. Installing Script Runner

> ⚠️ **Important:** As mentioned in *the chapter on SaaS*, installing and using external Script Runner tools is not supported when your AE is a SaaS setup.

Follow these steps to install the Script Runner tool.

These client applications and other tools can be downloaded:

1. Like the main client applications, also the Script Runner tool can be installed from an AE browser client. Go to `http://<AEserverName>:9000/#/downloads`). In its home page, go to the tab **Apps & Tools** > **Tools** and **Download** Script Runner.

2. After downloading, double-click the 'dmg' file (Mac) or the 'exe' file (Windows) to start the installation. Follow the instructions in the Assistant (Mac) or Installshield Wizard (Windows).

3. Check the tool's preferences to make sure it is running on that computer:

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

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

   - **Start / Stop** the Script Runner (or just check if it is running).

     > 📄 **Note:** On Windows, this will run the Script Runner as a 'Service'. Learn about (the need for) other methods in *Adobe Applications on Windows: Run Script Runner as an Application (Not as a Service)* on page 34.

   - Enable / disable **Start at login**

   - View / change the **Port** that the Script Runner is communicating with

   - View / change the default folders for (types of) scripts.

   Example (on Windows):
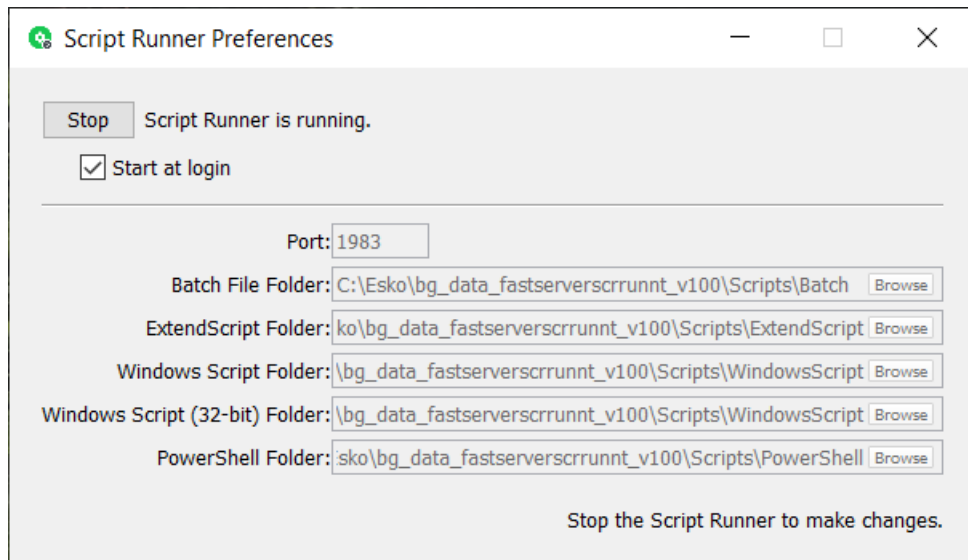
Attention: The above shown folders are local, on the Script Runner computer. Learn how you can (also) store them centrally in *Configuring Script Runner* on page 8.

4. Configuring Script Runner

The Automation Engine server needs to know where your installed your **Script Runner**(s). You could for example have 2 Macs dedicated to run scripts and also have the Automation Engine server itself have some Windows scripts or batch files running.

1. Install **Script Runner** on the client computers that will run scripts. Learn more in *Installing Script Runner* on page 6.

2. In a **Pilot**, go to **Tools** > **Configure**.

3. Go to the category **Scripts**. (As mentioned earlier, this configuration topic does not appear in a SaaS setup)

4. Now define which computers will run the scripts. Press **Insert** or choose **File** > **New** to add one.
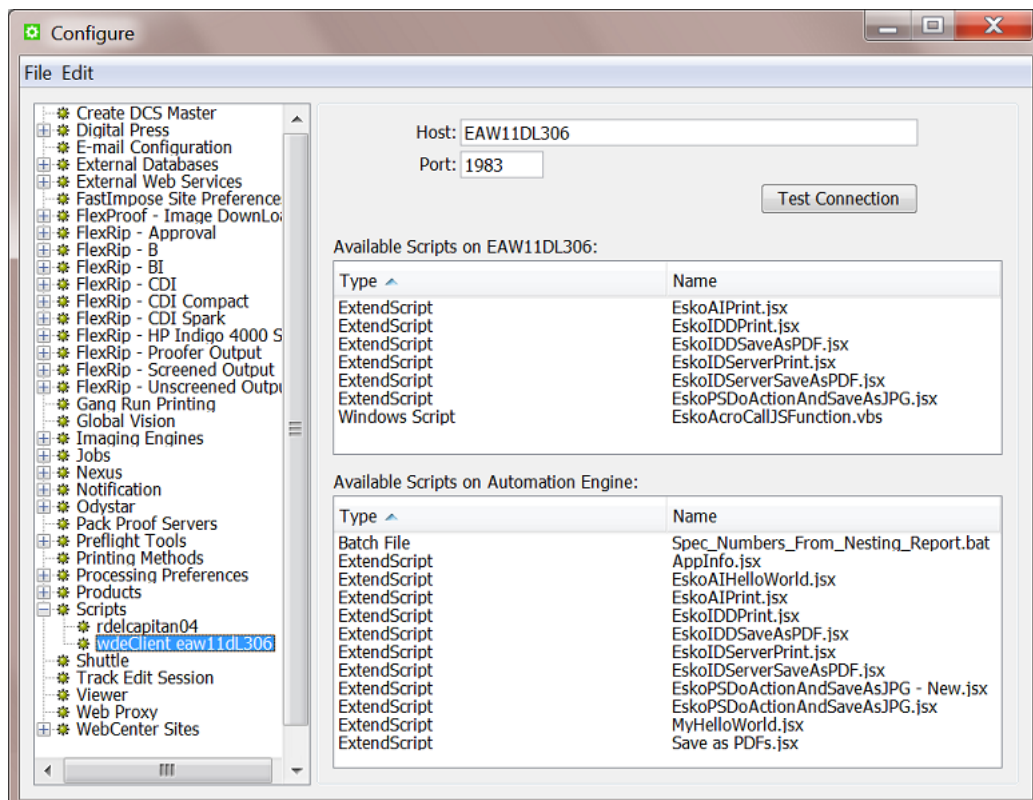
5. Give a suitable name to that Script Runner. Press **F2** or choose **File** > **Rename**.

6. In the **Host** field, enter the computer name or IP address of the Script Runner computer.

7. Enter the **Port** used to connect this computer to your Automation Engine server.

   By default, this is `1983`.

8. Click **Test Connection**. You should now see a list of all available scripts on that computer or see a message '`No scripts available`'.

   These are the scripts that the *Run Script Task* will let you choose from when you have selected that Script Runner configuration in the ticket.

You can store scripts on a local client (Script Runner) or centrally on the Automation Engine server:

- **Local**: You can store scripts on your local client computer on the folder specific for script type. But note that Automation Engine does not back up these local scripts. It can be useful however to only have them local when you are still writing and testing a script or when you do not intend to give access to other Script Runners in your network.

  > **Note:** The default location on a client computer is (macOS) `/Library/Scripts/ Esko` and (Windows) `C:\Esko\bg_data_fastserverscrrunnt_v100\Scripts`.

- **Central**: Alternatively, you can store your scripts in a central Automation Engine system folder. This option is suitable when you want to make your scripts available to **all** configured Script Runners. They will then also be part of your Automation Engine backup.

  > **Note:** The default location on the AE server is `C:\Esko \bg_data_fastserver_v100\Scripts`. The subfolder names indicate the type of script.

9. Choose **Save** (from the menu of the Configure panel).
10. Select the Configure panel's category **Scripts** (not an item you added, but one level higher). On this level you see a dialog where you can choose **Download Scripts** to download all scripts from all configured Script Runner tool(s) and centralize them onto the Automation Engine server. This is done to have a central library that will also be part of your *server configuration backup*.

    > **Note:** When local scripts would overwrite central ones, you will be asked to confirm.
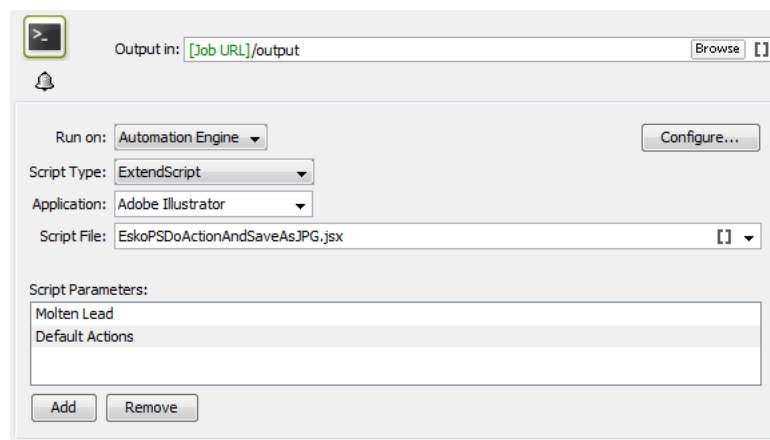
5. The Run Script Task

> **Note:** Learn about the concept and the setup of custom scripts in *Scripting Concept* on page 3.

This task contacts a *Script Runner tool* to run a script on the input file(s).

> **Note:** Since AE 22.03, this task can also be started on an empty token (for example without input files).

1. **Run on:** Choose the **Script Runner** tool that you *configured in the Configure panel*.

2. Select the **Script Type**. The types are different, depending if the selected Script Runner runs on a macOS or a Windows computer:

   a) Script Runner on macOS: You can choose from

   • **AppleScript**

   • **Shell Script**

   • **ExtendScript**.

   • **PowerShell Script**

   > **Note:** On macOS, PowerShell has to be installed. As long as PowerShell is not installed on a Mac script runner node, it will not show up in the list of script types.
   >
   > Installing PowerShell on macOS is documented on Microsoft web pages, for example *these*.

   b) Script Runner on Windows: You can choose from

   • **Batch File**

   • **ExtendScript**

- **Windows Script**

- **Windows Script (32-bit)**.

  > **Note:** Since AE 16.1, you can only install the Script Runner tool on a 64 bit host, but you can still run 32 bit Windows scripts by specifying the script type as `Windows Script (32 bit)`.

- **PowerShell Script**

  > **Note:** On Windows, PowerShell comes with the OS. In most cases you get an old version like v5.1, but you can manually update to a recent version (for example v7.2)

3. Select the **Application**. This field helps to find the right application, especially when you have multiple versions installed. The drop down list is a default list that you can edit.

   - On macOS, this is the (original) name of the .app file of that application
   - On Windows, this is the "Target location" that appears when you ask the "Properties" of that application. An example:

> **Tip:** It is possible that you do not have to enter the full name, but minimally the part that will help the task to recognize which of the multiple versions you want. For example when you both have a CS and a CC version, then the year and the bit version do not have to be mentioned extra (because those are only mentioned after the 'CS' or 'CC' part).

4. Select the **Script File**. You can select one from the drop-down list, enter a path or/and use SmartNames.

5. When the script needs one or more optional parameter(s), enter them in **Script Parameters**. Click **Add** and type the parameter.

> **Note:** Learn more details and see some examples in the next pages.

6. Script Runner on macOS - Examples

> ⚠️ **Warning:** These 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.

6.1. AppleScript - Example 1

AppleScript is a scripting language that enables direct control of script-able applications and of many parts of the macOS. An AppleScript-able 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 macOS
- it is supported by a lot of third party applications
- it very accessible to scripting beginners.

1. Open the AppleScript Editor and add following code:
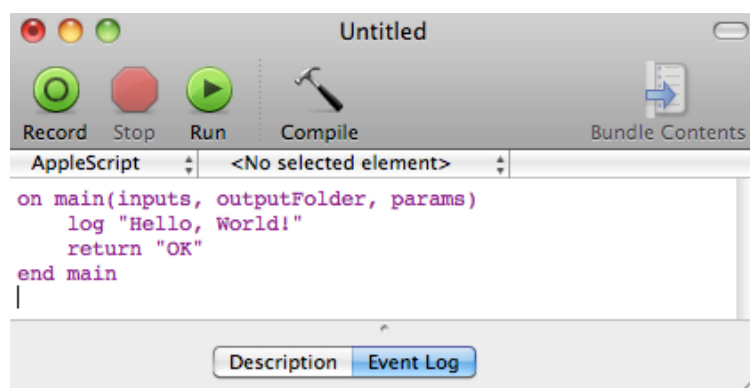
| Option	Description
main	This function will be called by the Script Runner. Only the code in this main function gets executed.
inputs	The first argument of the main function: a list of input file paths (type: list of strings).

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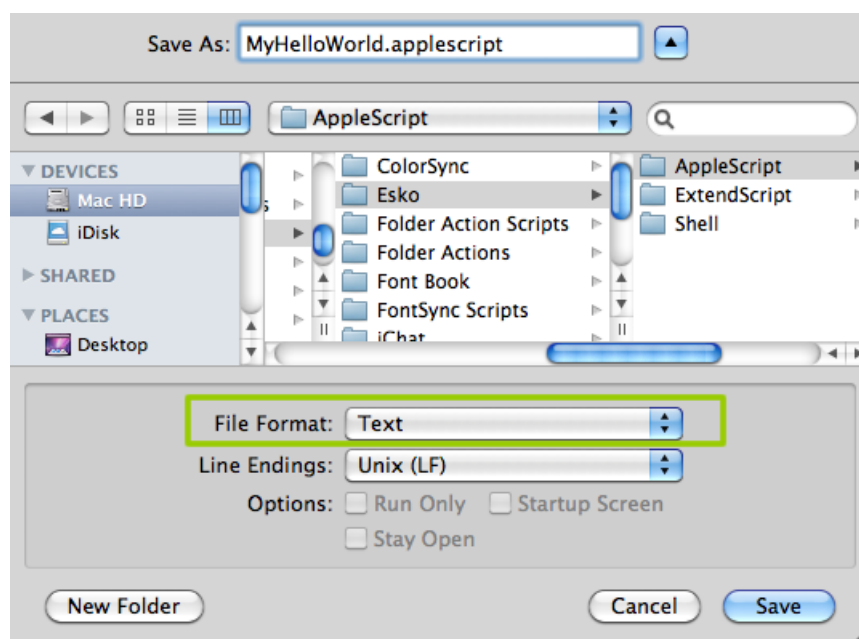
Note: Script Runner supports 'Text' format. Therefore it is essential to change the file format to 'Text'.

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.
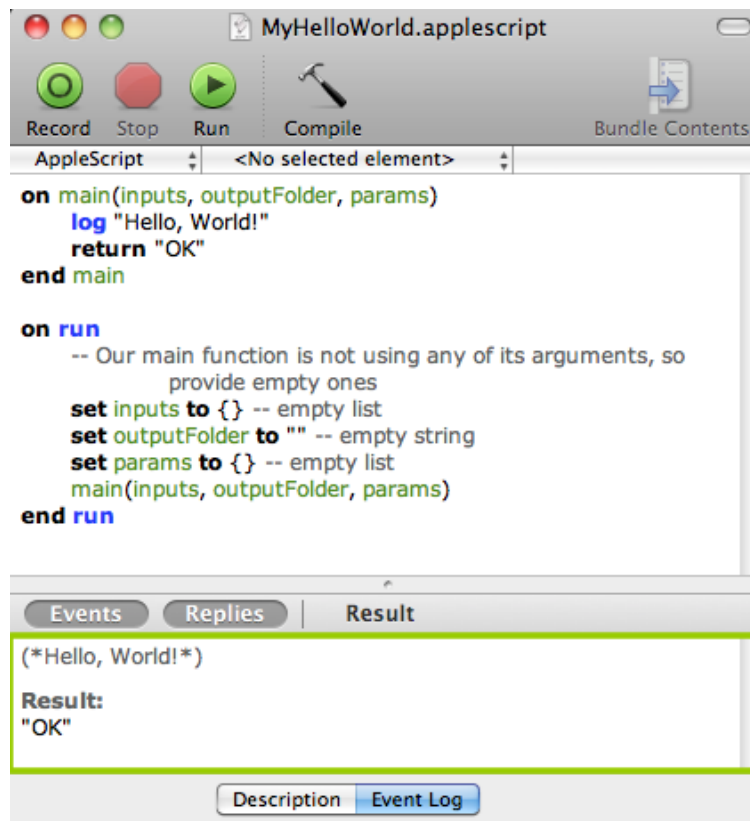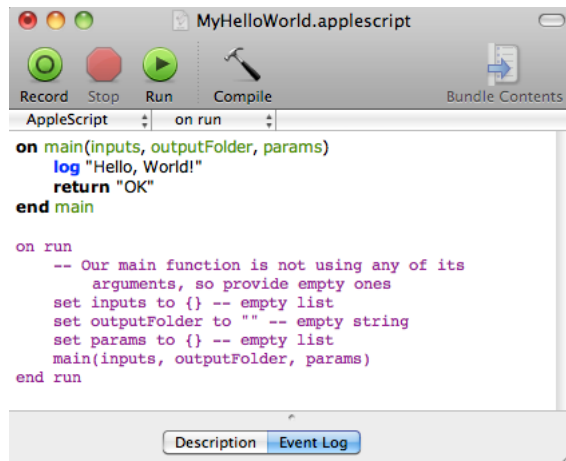
Notice the 'Hello, World!' and 'OK' result in the event log. The Script Runner does not interpret the test code in your script. It will execute the contents of the main function and ignore the rest. You can keep any test code for future local testing.

4. In the 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.
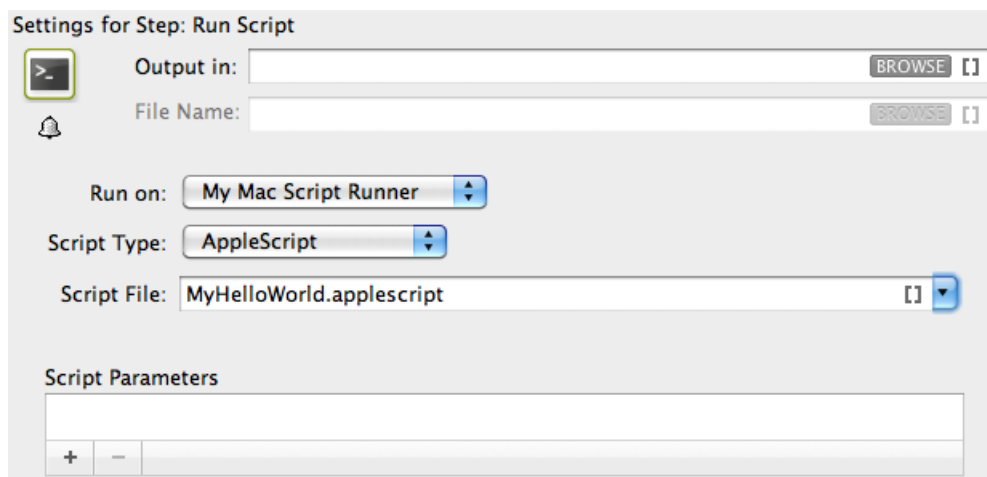
Learn more in *The Run Script Task* on page 10.

Note that the 'Hello, World!' in the task details and 'OK' state correspond with `log "Hello, World!"` and `return "OK"` in the script.
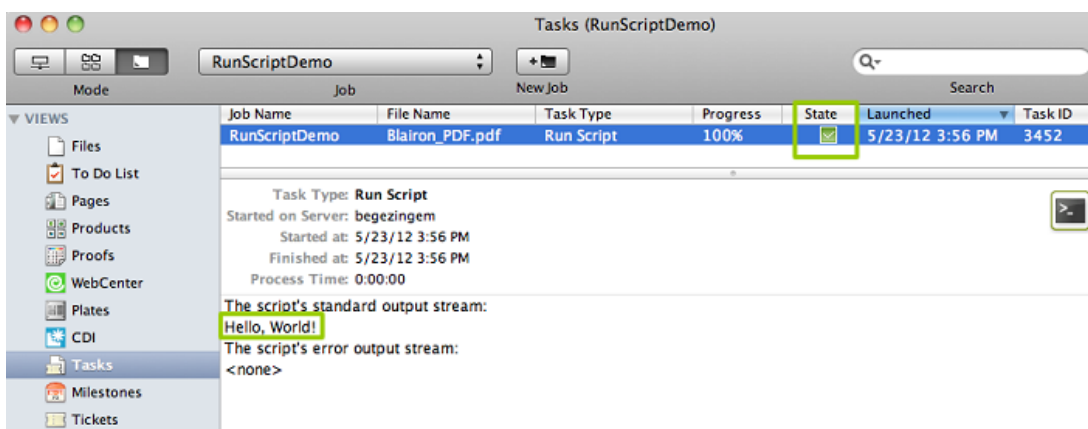
6.2. AppleScript – Example 2

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. To do that, we use `inputs`, `outputFolder` and `params` in the AppleScript.

First, we illustrate how to duplicate files without the size restriction and then we proceed with the actual case.

1. Open the AppleScript Editor and add the below shown 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.
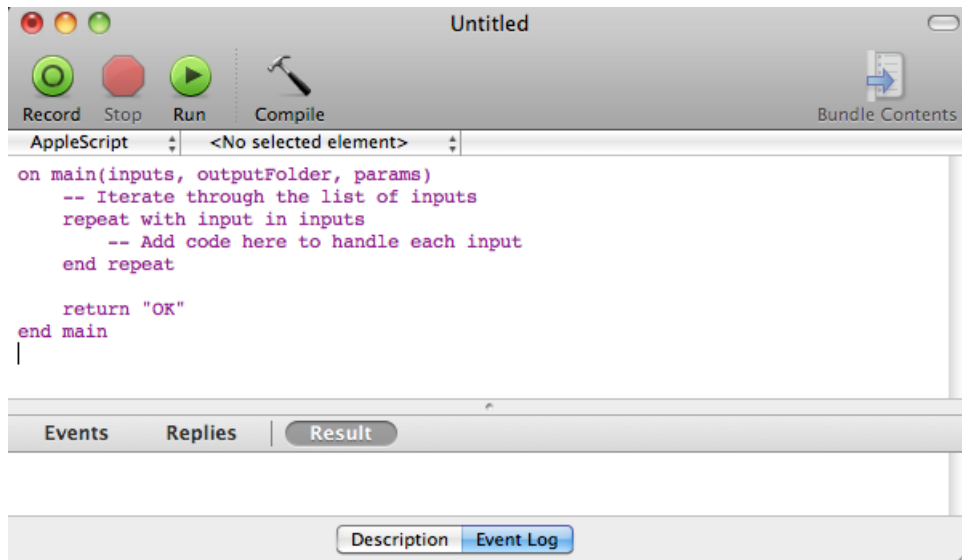
2. You can modify the Script as shown below to duplicate the files to a specified output folder without size restrictions. Save this code as an AppleScript text file in the default AppleScript folder of Script Runner (default: **/Library/Scripts/Esko/AppleScript**) or in the Automation Engine AppleScript folder.
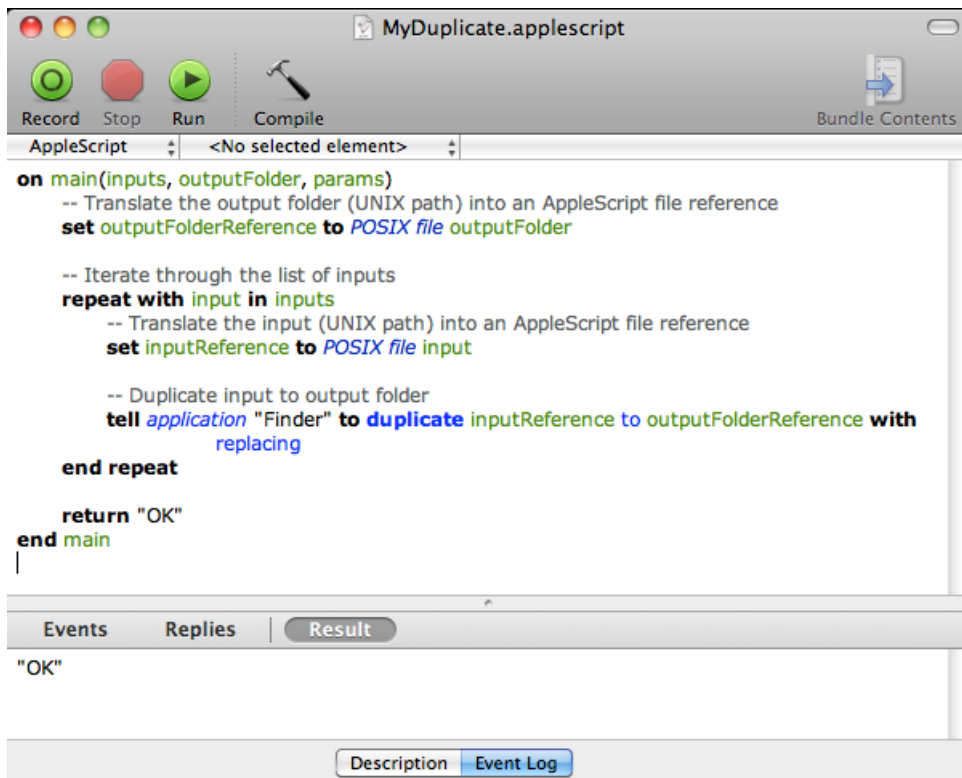
3. Add the file size check in the code as shown below. This will duplicate the file when the input file size is smaller than the maximum size from the script parameters. If this condition is not met, it will add an entry in the log and there will be "Warning". Save the file.

```
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
```

4. In the Pilot, go to **Files** view, 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 10 MB to the current job's Script Output folder. In this example, we executed this task for two files (BigBox.pdf: 22 MB and SmallLabel.pdf: <1 MB).
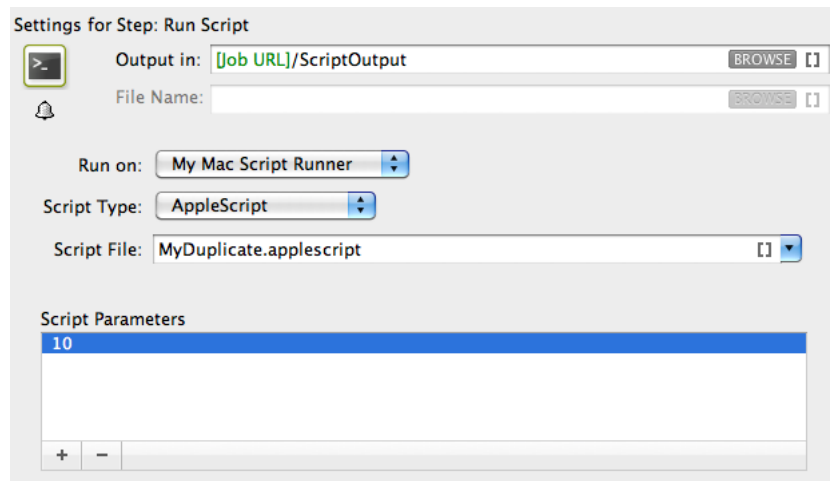
'SmallLabel.pdf' is duplicated into the job's 'ScriptOutput' folder. 'BigBox.pdf' was too big to duplicate (> 10 MB). Therefore, the task ended in a 'Warning' state and added an entry in the task details.
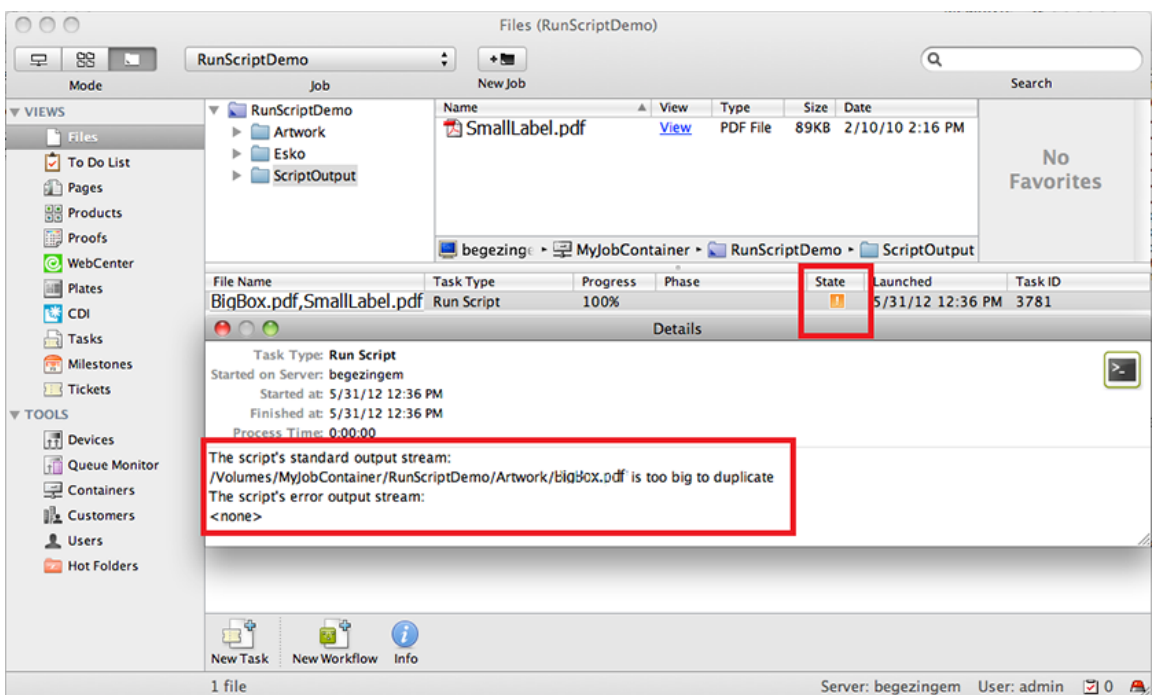
6.3. Shell Script – Example

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 the below shown 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.

```
#!/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[@]}

# Iterate through the list of inputs
for (( i=0; i<${inputCount}; i++ ));
do
    # Add code here to handle each input
done

exit 0
```

2. Write the code as shown 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.

```bash
#!/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
```
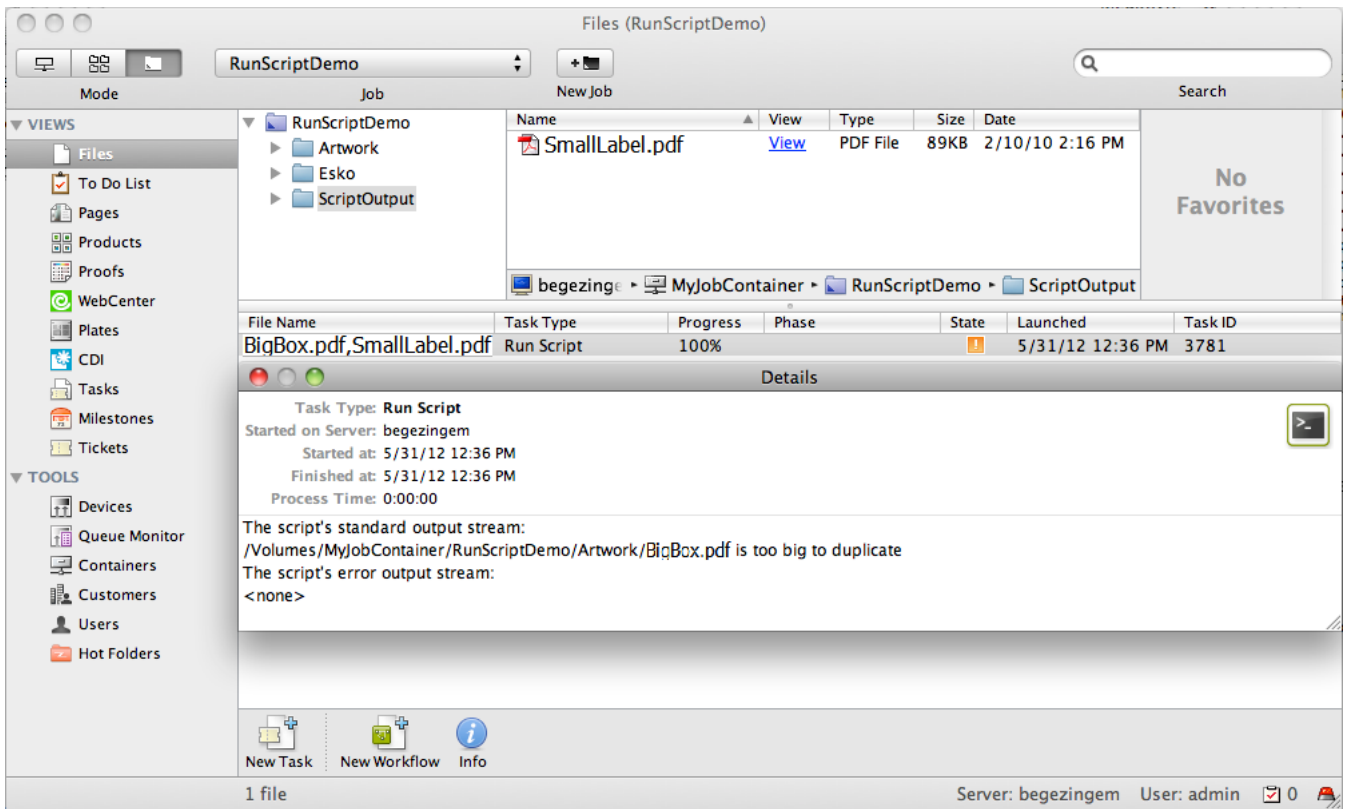
$1	First Shell Script argument: the Run Script task's inputs. A string of input file paths, separated by ':'.
$2	Second Shell Script argument or output folder: This is the folder where Automation Engine expects the script's result files. AE will continue the flow with the files you write in this folder. 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 task.

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

3. In the Pilot, go to **Files** view, 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 10 MB to the current job's Script Output folder. In this example, we executed this task for two files (BigBox.pdf: 22 MB and SmallLabel.pdf: <1 MB).

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

7. Script Runner on Windows - Examples

⚠️ **Warning:** These 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.

7.1. Windows Script Example 1

1. Open a text editor and add the below shown 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 communicates to 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 code shown below. 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 interpret the test code in your script. It will execute the contents of the main function and ignore the rest. You can keep your test code for future local testing.

3. In a Pilot, go to **Files** view, select a file and open a **New Task**. Choose the **Run Script** task, modify its settings and launch the task.
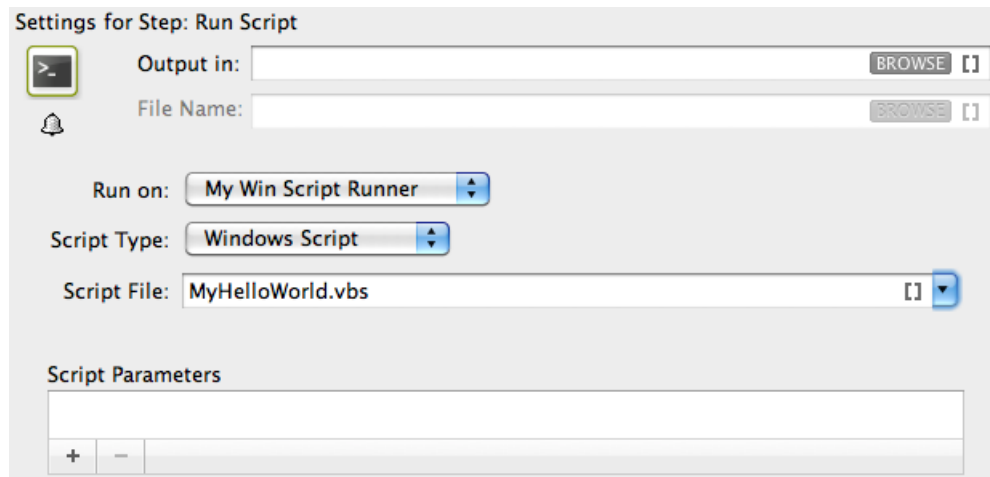
Note that the 'Hello, World!' in the task details and 'OK' state are corresponding with `WScript.Echo "Hello, World!"` and `Main = "OK"` in the script.
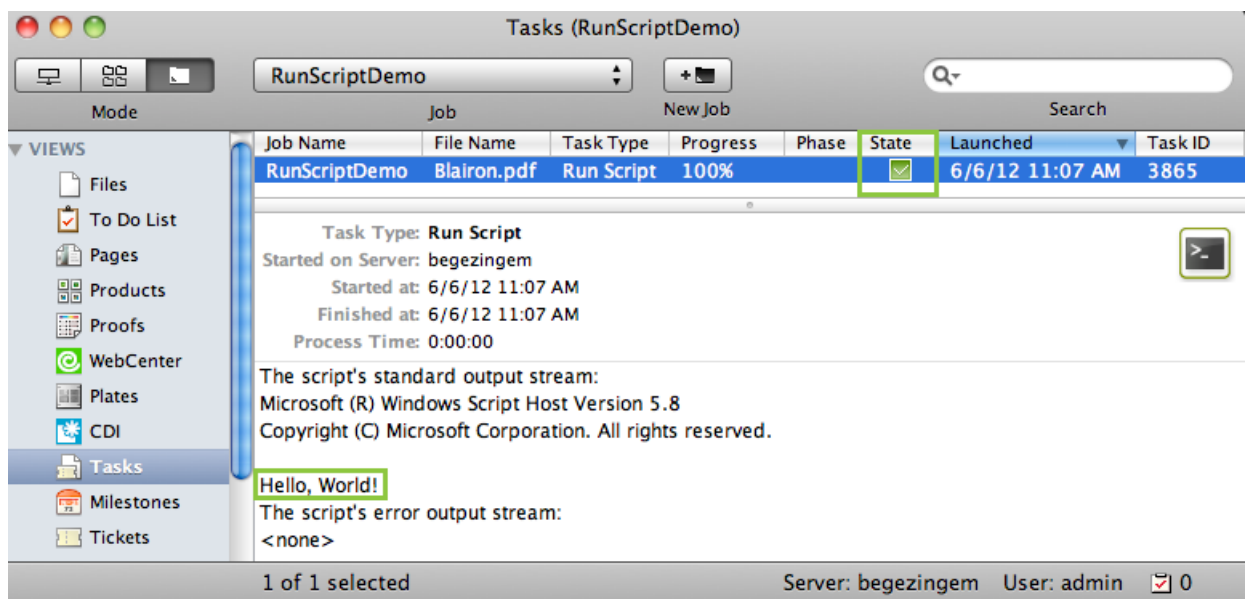
7.2. Windows Script Example 2

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 use `inputs`, `outputFolder` and `params` in the script to achieve our objective.

First, we illustrate how to duplicate files without the size restriction and then we proceed with the actual example.

1. Open a text editor and add the below shown 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.

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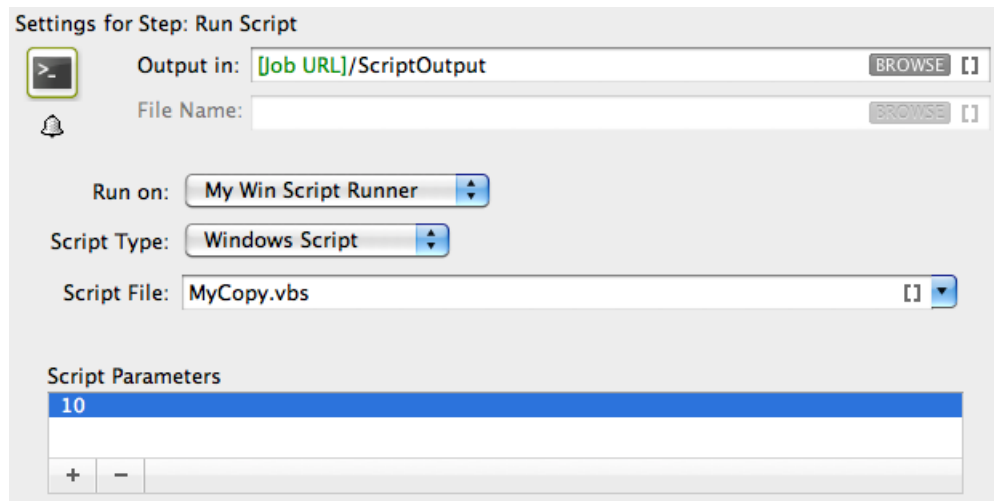
3. Add the file size check in the code as shown below. This will duplicate the file when the input file size is smaller than the maximum size from the script parameters. If this condition is not met it will add an entry in the log and there will be "Warning". Save the file.

```vbs
1
2    Function Main(inputs, outputFolder, params)
3        Dim returnValue
4        returnValue = "OK"
5
6        ' Get the maximum size from the script parameters
7        ' and convert from megabyte to bytes.
8        Dim maxSize
9        maxSize = params(0) * 1000000
10
11       ' Create the file system object that is going to do the copying.
12       Dim filesys
13       Set filesys = CreateObject("Scripting.FileSystemObject")
14       WScript.Echo "File system object created"
15
16       ' Iterate through the list of inputs.
17       For Each input In inputs
18           ' Turn the input path into a file object.
19           Dim inputReference
20           Set inputReference = filesys.GetFile(input)
21
22           ' Check wether the input is OK to copy.
23           If inputReference.Size < maxSize Then
24               ' Copy input to output folder.
25               Dim outputFile
26               outputFile = outputFolder & "\" & filesys.GetFileName(input)
27               filesys.CopyFile input, outputFile
28           Else
29               WScript.Echo input & " is too big to copy"
30               returnValue = "Warning"
31           End If
32       Next
33
34       ' Set the function's return value.
35       Main = returnValue
36   End Function
37
```

4. In a Pilot, go to **Files** view, 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 10 MB to the current job's 'ScriptOutput' folder. In this example, we executed this task for two files (BigBox.pdf: 22 MB and SmallLabel.pdf: <1 MB).

'SmallLabel.pdf' is duplicated into the job's 'ScriptOutput' folder. 'BigBox.pdf' was too big to duplicate (> 10 MB). Therefore, the task ended in 'Warning' state and added an entry in the task details.
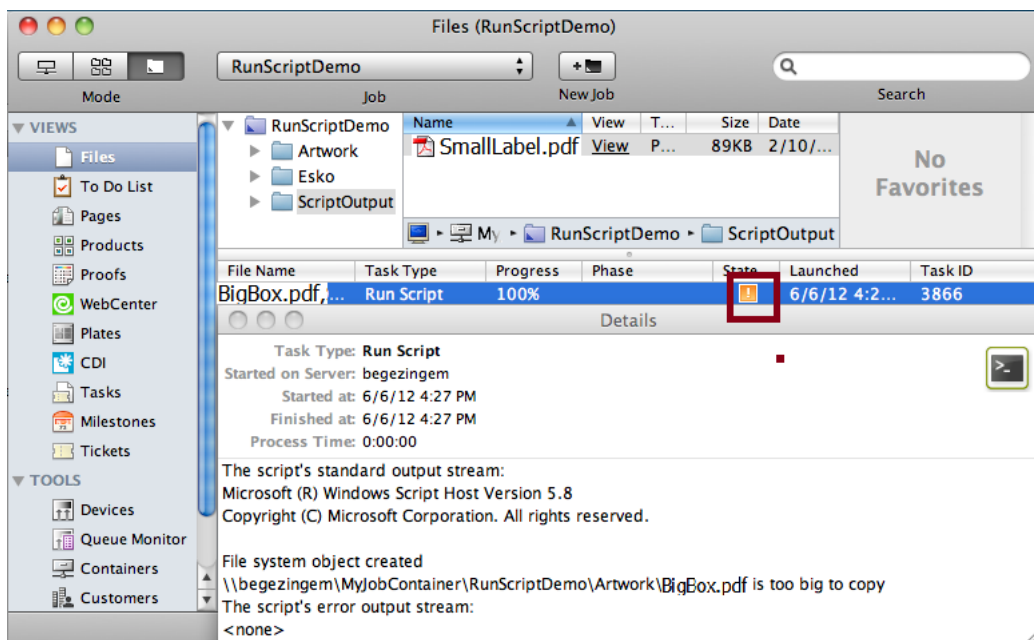
7.3. Batch File Example

In this example, we use a 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 below shown code. 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 ';'.

```
@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

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
echo Add code here to handle each input
goto :eof

:EXIT
exit %exitValue%
```

2. You can modify the Script as shown 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++

1   @echo off
2   rem First script argument contains the input file paths
3   set inputs=%1
4   rem Trim surrounding quotes
5   for /f "usebackq tokens=*" %%a in ('%inputs%') do set inputs=%%~a
6
7   rem Get the output folder
8   set outputFolder=%2
9
10  rem Get the maximum size from the script parameters
11  rem and convert from megabyte to bytes
12  set /a maxSize=%3*1000000
13
14  set exitValue=0
15
16  rem Iterate through the input file paths which are separated by ';'
17  :ITERATE_INPUTS
18  if "%inputs%"=="" goto EXIT
19  for /f "tokens=1 delims=;" %%a in ("%inputs%") do call :HANDLE_INPUT "%%a"
20  for /f "tokens=1* delims=;" %%a in ("%inputs%") do set inputs=%%b
21  goto ITERATE_INPUTS
22
23  :HANDLE_INPUT
24  rem Get the size of the input
25  for %%? in (%1) do set inputSize=%%~z?
26  rem Check wether the input is OK to copy
27  if %inputSize% lss %maxSize% (
28      rem Copy input to output folder
29      copy /y %1 %outputFolder%
30  ) else (
31      echo %1 is too big to copy
32      set exitValue=1
33  )
34  goto :eof
35
36  :EXIT
37  exit %exitValue%
38
```

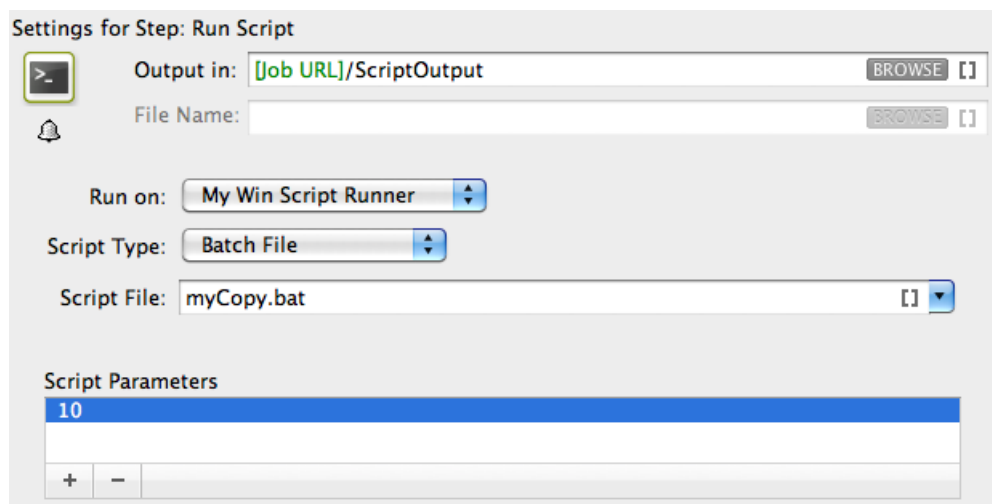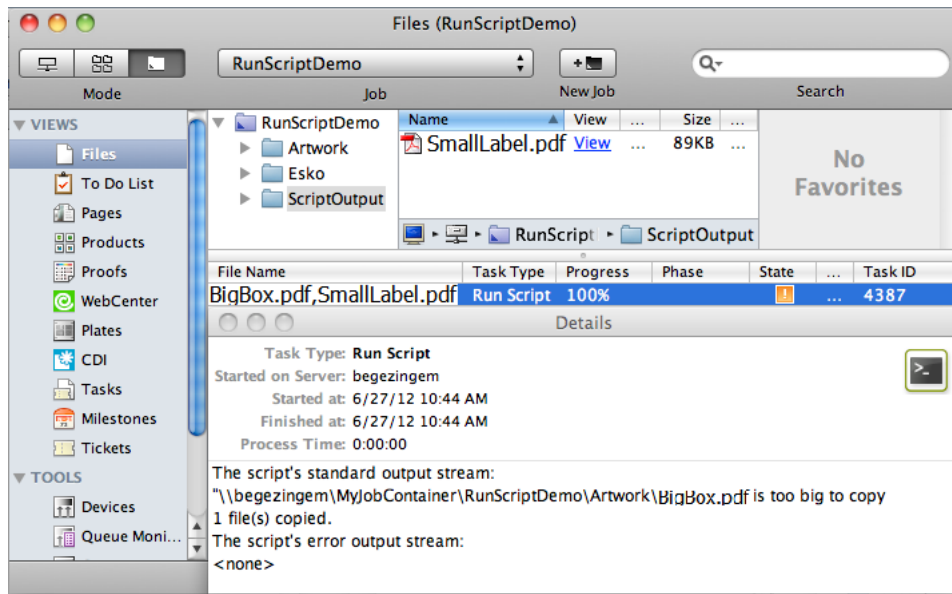| %1	First batch file argument: the Run Script task's inputs. A string of input file paths, separated by ';'.
%2	Second batch file argument or output folder: the folder where AE expects the script's result files. AE will continue the flow with the files you write in this folder. If you leave this folder 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** task.

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

3. In a Pilot, go to **Files** view, 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 10 MB to the current job's Script Output folder. In this example, we executed this task for two files (BigBox.pdf: 22 MB and SmallLabel.pdf: <1 MB).

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

8. Using ExtendScript (macOS & Windows)

8.1. Adobe Applications on Windows: Run Script Runner as an Application (Not as a Service)

When you use ExtendScript on Windows, you can avoid problems 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 the Adobe settings)**.

Follow these steps to do this:

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

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

- Start it as a console application by double clicking its executable in the Script Runner's program folder `>\bin_ix86\egscrrun.exe`. For example `C:\Esko \bg_prog_fastserverscrrunnt_v141\bin_ix86\egscrrun.exe`

8.2. ExtendScript in Adobe Illustrator - Example 1

ExtendScript is JavaScript extended for Adobe CS/CC applications. Adobe provides the ExtendScript Toolkit (ESTK): a complete IDE (integrated development environment) to program ExtendScript. Learn more about Adobe scripting resources in the *Adobe Scripting Center*.

> ⚠ **Attention:** As mentioned in the *introduction*, Adobe (Windows) 32 bit applications are no longer supported on a standalone Script Runner tool.

1. Open the ExtendScript Toolkit and add the below shown 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 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 communicates to 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 below shown code, save and run the script.

As a result, 'Hello, World!' and 'OK' are shown in the Console and the alert box pops up:

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

3. In the Pilot, go to **Files** view, 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.
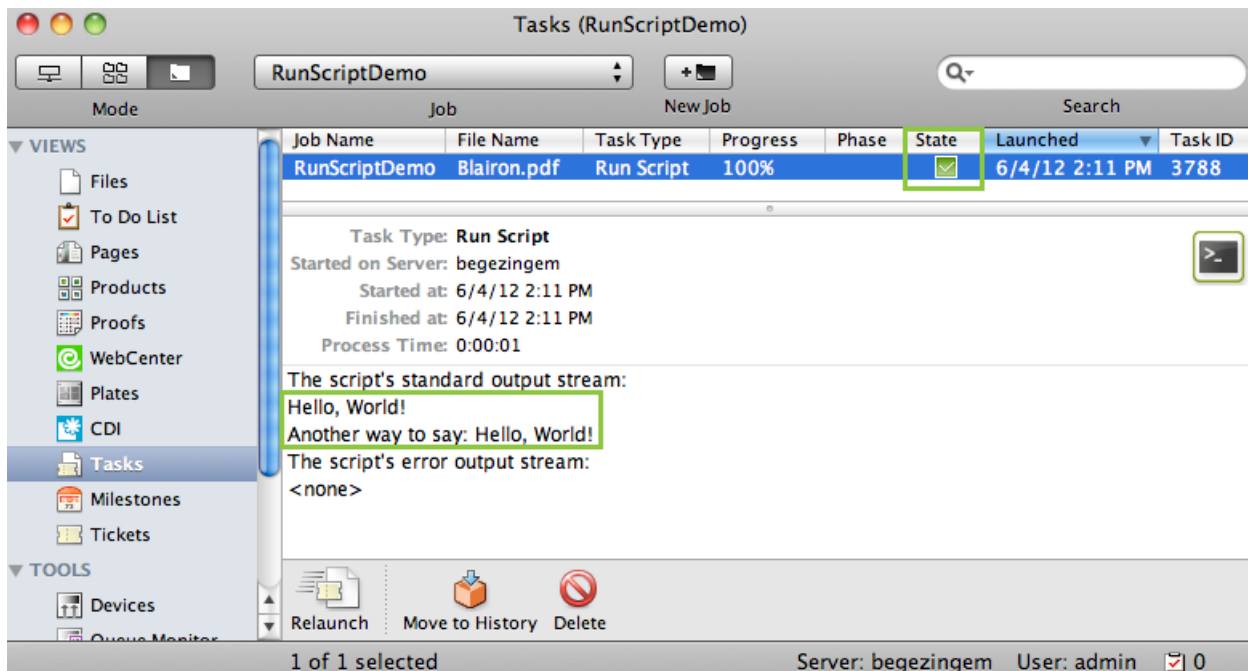
8.3. ExtendScript in Adobe Illustrator - Example 2

In this example we use ExtendScript to **Open** a file in Illustrator and print it using a **Print Preset**.
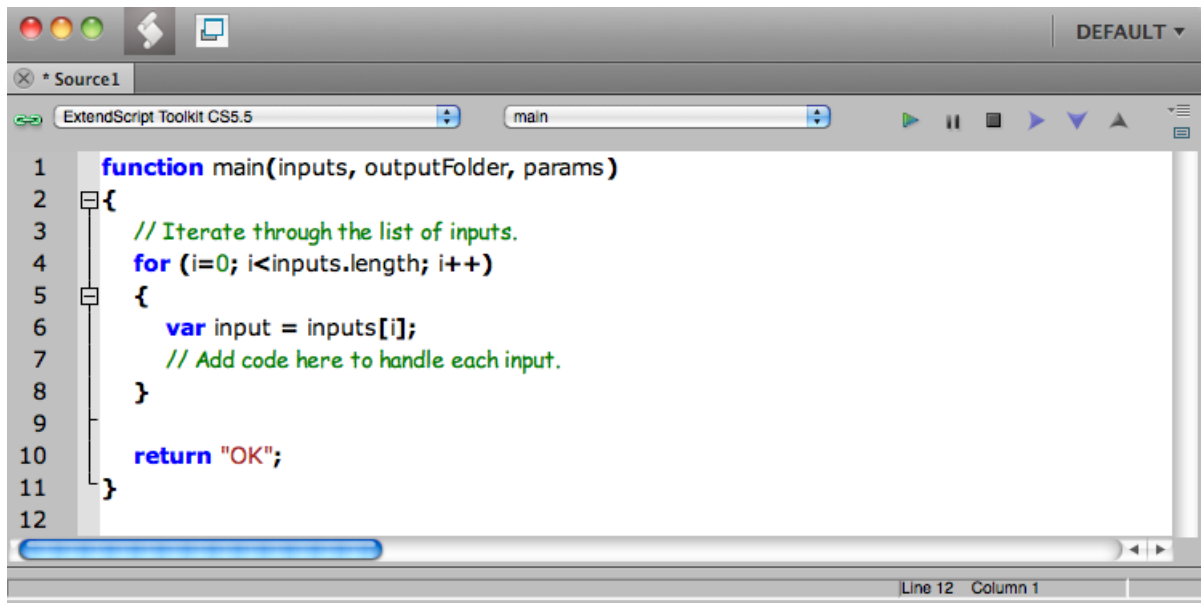
> **Note:** Make sure you read this page first: *Adobe Applications on Windows: Run Script Runner as an Application (Not as a Service)* on page 34.

> **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 the below shown 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 in the **Run Script** task, write the code as shown below. Save this code in the default ExtendScript folder (of Script Runner or of Automation Engine).

```
1   function main(inputs, outputFolder, params)
2   {
3       // First and only script parameter is the print preset.
4       var printPreset = params[0];
5
6       var returnValue = "OK";
7
8       // Iterate through the list of inputs.
9       for (i=0; i<inputs.length; i++)
10      {
11          var input = inputs[i];
12
13          try
14          {
15              // Open the input file.
16              var myDocument = app.open(File(input), DocumentColorSpace.CMYK);
17              // Create print options and set the print preset.
18              var options = new PrintOptions();
19              options.printPreset = printPreset;
20              // Print the document.
21              myDocument.print(options);
22              // Close the document without saving.
23              myDocument.close(SaveOptions.DONOTSAVECHANGES);
24          }
25          catch (e)
26          {
27              // Log problem info and set the return value to "Warning".
28              $.writeln("Problem while printing " + input);
29              $.writeln("Name: " + e.name);
30              $.writeln("Message: " + e.message);
31              returnValue = "Warning";
32          }
33      }
34
35      return returnValue;
36  }
37
```

4. In a Pilot, go to **Files** view, select a file and open a **New Task**. Choose the **Run Script** task, modify its settings and launch the task.
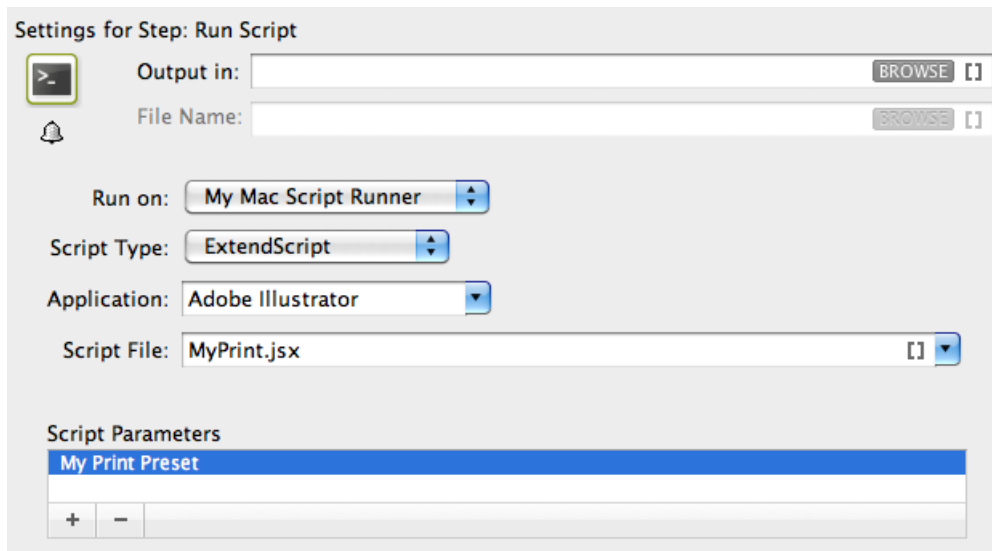
Launching this ticket will print the selected Illustrator files using the **Print Preset** specified as script parameters (`My Print Preset`) in the task.

8.4. ExtendScript in Adobe Photoshop – Example

> **Note:** These sample 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.

In this example, we illustrate a script that performs a Photoshop "action" and then produces a JPEG output. The script 'EskoPSDoActionAndSaveasJPG.jsx' is available from the sample scripts subfolder "ExtendScript" in `/Library/Scripts/Esko` (Mac) or `C:\Esko \bg_data_fastserverscrrunnt_v100\Scripts` (Windows).

1. You have to save the **Set** and **Action** in Photoshop. In this example we are using an action named "`Molten Lead`" which is one of the "`Default Actions`" (folder) as **Set** (variable 2) and **"Molten Lead"** as **Action** (variable 1).

2. You can open the script with ExtendScript Toolkit to edit the script if required. However, in this sample, we do not need to edit the script.

```
ExtendScript Toolkit CS4                    main

/**
 * 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
 */

function main(inputs, outputFolder, params)
{
    for (i=0;i<inputs.length;i++)
    {
        var fileRef = new File(inputs[i]);
        var docRef = app.open(fileRef);

        app.doAction(params[0], params[1]);

        outputFile = new File(outputFolder + "/" + fileRef.name);
        jpgSaveOptions = new JPEGSaveOptions();
        jpgSaveOptions.embedColorProfile = true;
        jpgSaveOptions.formatOptions = FormatOptions.STANDARDBASELINE;
        jpgSaveOptions.matte = MatteType.NONE;
        jpgSaveOptions.quality = 1;
        docRef.saveAs(outputFile, jpgSaveOptions, true, Extension.LOWERCASE);
        docRef.close(SaveOptions.DONOTSAVECHANGES);
    }

    return "OK"
}
```
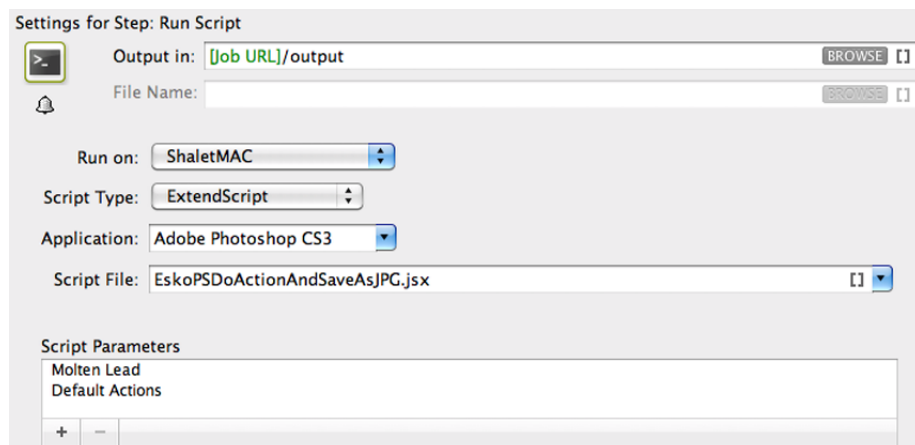
3. Edit the settings of the **Run Script** task as shown below:

a) **Run on:** is the name of the computer where your **Automation Engine ScriptRunner** is installed. The name in this example is ShaletMac. Learn more about configuring and naming a Script Runner in *Configuring Script Runner* on page 8.

b) **Script Type:** choose ExtendScript.

c) **Script File:** choose EskoPSDoActionAndSaveasJPG.jsx.

d) **Script Parameters:** Add the name of your Photoshop **Action** name followed by your **Set**. In this example, Molten Lead; Default Actions

e) Save the task ticket.

4. Launch the task:

a) Select a Photoshop file.

b) Right-click the file and browse for the ticket.

c) Click **Launch.**

You will see that while the script starts to run on the Automation Engine server, Photoshop opens, performs the actions, closes and continues the script. The result of this sample script is an adjusted JPEG file.

9. PowerShell Examples (macOS & Windows)

My First PowerShell

This one only returns the message between "". See it appear in the task details (and log file).

```
Write-Host "Congratulations! Your script executed successfully"
```

Write Local Time to File

This one writes the local time of the executing computer in the task details and also in a .txt file on the specified path.

```
Get-CimInstance -ClassName Win32_LocalTime | Out-File -FilePath C:\temp\localtime.txt
```

Some specifics for an AE task

This one shows some specifics that help in the context of an AE task:

• `CmdletBinding` to capture the task's standard and custom parameters.

• `-split ':'` to split the list of input files (where more than one).

• Returning a task exit status: `0` = OK, `1` = warning, `2` = error .

```
[CmdletBinding()]
 Param
    (
     # Param1 help description
    [Parameter(Position=0)][string]$inputfiles,
 [Parameter(Position=1)][string]$outputFolder,
 [Parameter(Position=2)][string]$Parameter1,
 [Parameter(Position=3)][string]$Parameter2,
 [Parameter(Position=4)][string]$Parameter3
    )

# checks if variables are declared
Set-StrictMode -Version 2

#begin of main program

#$hostname = get-content env:computername
$hostname = hostname

Write-Host "Starting script example.ps1 on: ", $hostname

#show PowerShell version
Write-Host "PowerShell version: ", $PSVersionTable.PSVersion

Get-Date

Write-Host "Received input files", $inputfiles

$inputArray = $inputfiles -split ':'

$retval = 0
Write-Host "Number of input files: " $inputArray.length

$outstring = ""

foreach ($inputfile in $inputArray)
{
 Write-Host "Handling input file: " $inputfile
```

```
 $lastWrite = Get-Item $inputfile | select LastWriteTime
 $outstring = $outstring + $inputfile + "  " + $lastWrite +"`r`n"
}

Write-Host "Pstsmeter 1: " $Parameter1
Write-Host "Pstsmeter 2: " $Parameter2
Write-Host "Pstsmeter 3: " $Parameter3


$outputname = $hostname + ".lst"
try
{
New-Item -Path $outputFolder -Name $outputname -ItemType "file" -Value $outstring -
ErrorAction Stop
}
catch {
 Write-Output "An error occured while writing the output file"
    #Write-Output $_
 $errormessage = $_.Exception.Message
 Write-Host $errormessage

 $retval = 2
}
exit $retval
```

10. Script Samples

Click *this link* to download a PDF containing some samples of scripts.