Schedule Scripts

Overview

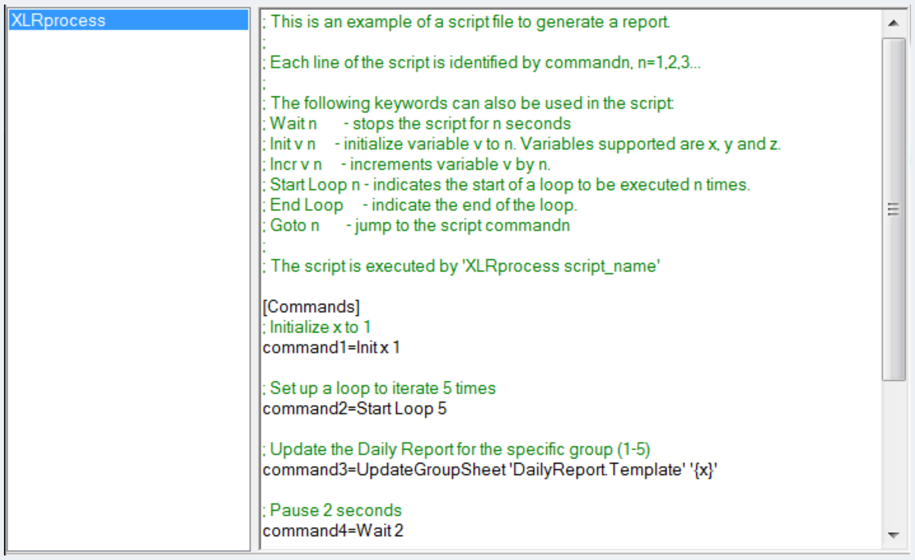
Schedule scripts provide basic programming capability coupled with the ability to trigger one or more schedule actions. For example, a schedule script can be used to monitor the file system and perform specific actions based on file activity.

The following document describes what schedule scripts are, how to configure them and what options are available.

Access Schedule Scripts

Schedule scripts are configured in the **Schedule Designer** (**Project Explorer**, **Project** tab, **Designer** button).

The **Script Editor** is opened in the **Schedule Designer** by selecting **Tools, Script Editor**.



```
; This is an example of a script file to generate a report.
;
; Each line of the script is identified by commandn, n=1,2,3...
;
; The following keywords can also be used in the script:
; Wait n       - stops the script for n seconds
; Init v n     - initialize variable v to n. Variables supported are x, y and z.
; Incr v n     - increments variable v by n.
; Start Loop n - indicates the start of a loop to be executed n times.
; End Loop     - indicate the end of the loop.
; Goto n       - jump to the script commandn
;
; The script is executed by 'XLRprocess script_name'

[Commands]
; Initialize x to 1
command1=Init x 1

; Set up a loop to iterate 5 times
command2=Start Loop 5

; Update the Daily Report for the specific group (1-5)
command3=UpdateGroupSheet 'DailyReport.Template' '{x}'

; Pause 2 seconds
command4=Wait 2
```

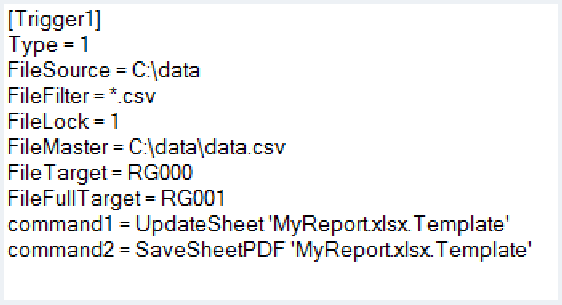
All the scripts configured in the project are listed on the left side. Select a script to view its configuration on the right. By default, every project has the script *XLRprocess*. This script is provided to illustrate the scripting functionality and syntax.

Perform Actions on File Activity

Scripts can be used to perform actions by monitoring folders for file activity.

Configuration Settings

Each trigger consists of a set of configuration settings that determine what to monitor and what to do when files are detected.



```
[Trigger1]
Type = 1
FileSource = C:\data
FileFilter = *.csv
FileLock = 1
FileMaster = C:\data\data.csv
FileTarget = RG000
FileFullTarget = RG001
command1 = UpdateSheet 'MyReport.xlsx.Template'
command2 = SaveSheetPDF 'MyReport.xlsx.Template'
```

The above can be repeated using [Triggerx] where x = 2,3 and so on.

Processing Cycle

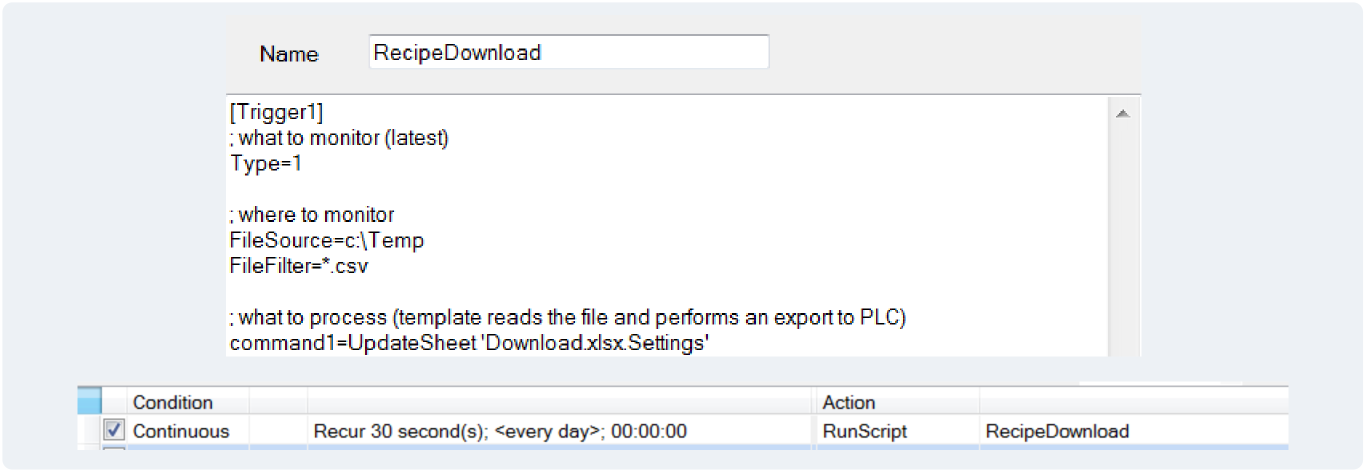
When a script is initiated, usually from the **Scheduler**, it follows the following processing cycle:

- **Where and What to Monitor**
  FileSource and **FileFilter** determines a file set which have a modified date (or creation data see below) newer than the last time the script was initiated.
- **Apply Filter**
  FileLock=1 filters the file set by removing files that do not have full read/write access. In other words, files that are in use will not be considered. Otherwise, the file set is not filtered (**FileLock**=0).
  Type=1 filters the file set to the newest file based on the modified date otherwise the file set is not reduced (**Type**=2).
  If **Type**=1 and **FileLock**=0, the newest file is based on the creation date rather than the modified date. This is most useful when the files are downloaded from a file server because the modified date for all these files are the same but the creation date would reflect the age of the files to determine the newest.
- **What to Process**
  For each file remaining in the file set (after applying the above filter conditions), the following step are performed:
  a. If configured, the file is copied using the name given in **FileMaster**. This is useful because each file is now identified by a fixed name so it is handled easier by other application.
  b. If configured, the following variables are assigned:
     FileTarget with the short name of the file (i.e., name without a path or extension).
     FileFullTarget with the full name of the file (i.e., name with a path and extension).
     Note that these variables can be used in the report templates referenced by the commands of the next step.
  c. **XLReporter** initiates each command starting with **command1**
  d. If configured, the file is moved to the folder given by **FileArchive.** If this setting is omitted when file(s) are processed, the date of the last processed file is retained in the file *_lastprocessed*.ini. Subsequent passes will only consider files after the date retained in this file.
- **When to Process pply Filter**
  The processing cycle is initiated from the **Scheduler** using the **Process Script** action.

Example

The process settings for each recipe, such as set points, are stored in a CSV file named after the recipe name. When a recipe is required by the process, a copy is moved to a folder being monitored by XLReporter which then reads the settings and downloads them to the PLC.

The following script with the schedule (monitoring for the file every 30 secs) provides the result:



Name: RecipeDownload

```
[Trigger1]
; what to monitor (latest)
Type=1

; where to monitor
FileSource=c:\Temp
FileFilter=*.csv

; what to process (template reads the file and performs an export to PLC)
command1=UpdateSheet 'Download.xlsx.Settings'
```

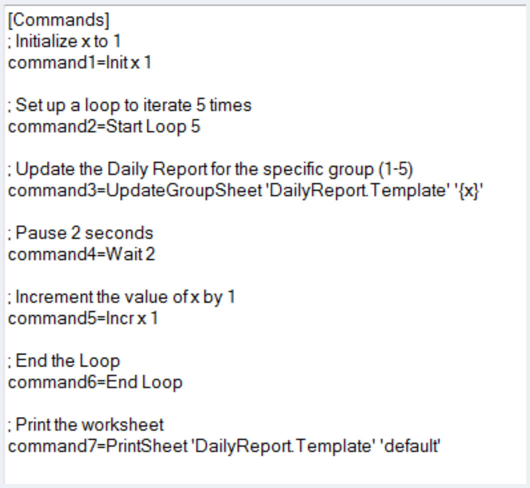
Condition		Action	
✓ Continuous	Recur 30 second(s); <every day>; 00:00:00	RunScript	RecipeDownload

Perform Actions By Rules

Scripts can be used to perform actions according to defined rules.

Configuration Settings

The defined rules are prefixed by **commandx** as follows, where x is an incrementing value starting at 1,



```
[Commands]
; Initialize x to 1
command1=Init x 1

; Set up a loop to iterate 5 times
command2=Start Loop 5

; Update the Daily Report for the specific group (1-5)
command3=UpdateGroupSheet 'DailyReport.Template' '{x}'

; Pause 2 seconds
command4=Wait 2

; Increment the value of x by 1
command5=Incr x 1

; End the Loop
command6=End Loop

; Print the worksheet
command7=PrintSheet 'DailyReport.Template' 'default'
```

Processing Cycle

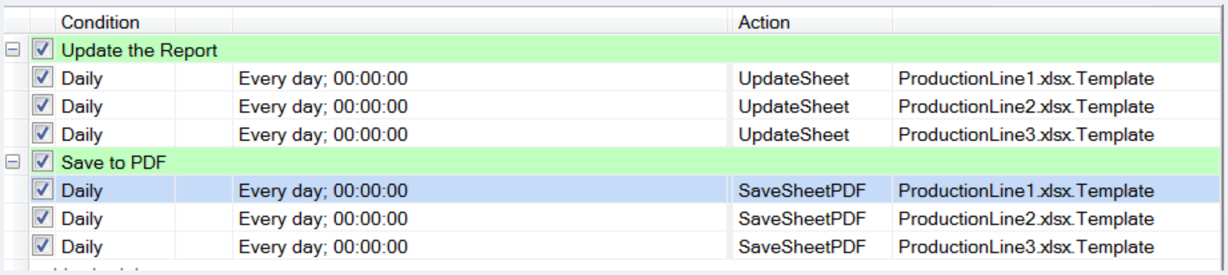
When a script is initiated, usually from the **Scheduler**, it starts at **command1** and sequentially moves through each command. However, if **Start Loop** is encountered then all the commands between the **Start Loop** and **End Loop** are processed for the number of times specified in **Start Loop**. When the loop has been exhausted, the script continues with the next command after the **End Loop**.

There is support for three variables: *x*, *y* and *z*. These can be initialized with **Init** and incremented with **Incr**. The values of the variables can be used in the script by enclosing them in {}, e.g., *{x}*.

Example

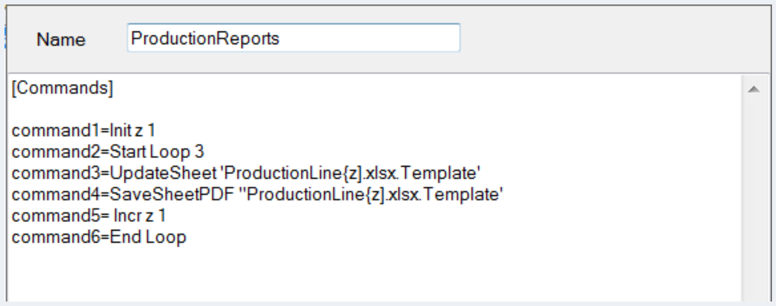
A facility has 3 production lines, each requiring a daily report as a PDF file.

The following schedule would give the desired result:



Condition		Action	
☑ Update the Report			
☑ Daily	Every day; 00:00:00	UpdateSheet	ProductionLine1.xlsx.Template
☑ Daily	Every day; 00:00:00	UpdateSheet	ProductionLine2.xlsx.Template
☑ Daily	Every day; 00:00:00	UpdateSheet	ProductionLine3.xlsx.Template
☑ Save to PDF			
☑ Daily	Every day; 00:00:00	SaveSheetPDF	ProductionLine1.xlsx.Template
☑ Daily	Every day; 00:00:00	SaveSheetPDF	ProductionLine2.xlsx.Template
☑ Daily	Every day; 00:00:00	SaveSheetPDF	ProductionLine3.xlsx.Template

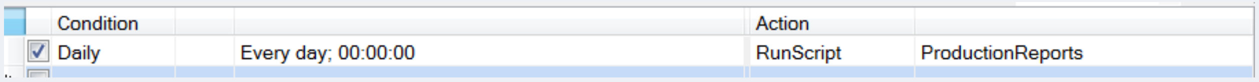
Or the following script and a simpler schedule:



Name: ProductionReports

```
[Commands]

command1=Init z 1
command2=Start Loop 3
command3=UpdateSheet 'ProductionLine{z}.xlsx.Template'
command4=SaveSheetPDF ''ProductionLine{z}.xlsx.Template'
command5= Incr z 1
command6=End Loop
```



Condition		Action	
☑ Daily	Every day; 00:00:00	RunScript	ProductionReports

Combine Multiple Actions

Schedule scripts can be configured to simply combine multiple actions into a single schedule line. However, it is recommended to use the Update Action List action to combine multiple actions together since it provides an easier environment to specify the actions and parameters for those actions.