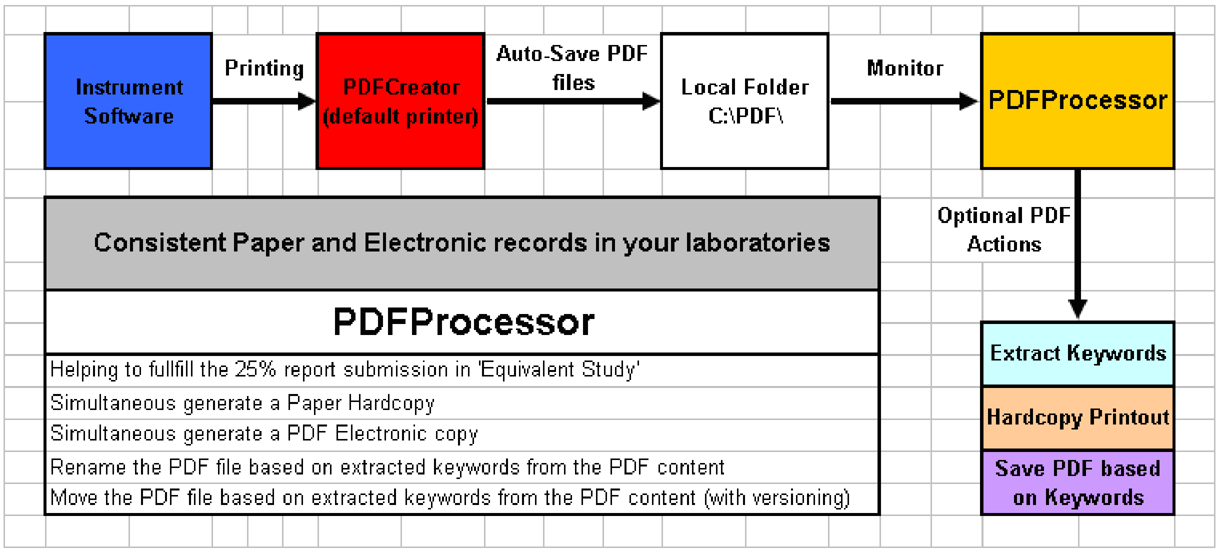
PDFProcessor

http://www.waleson.eu

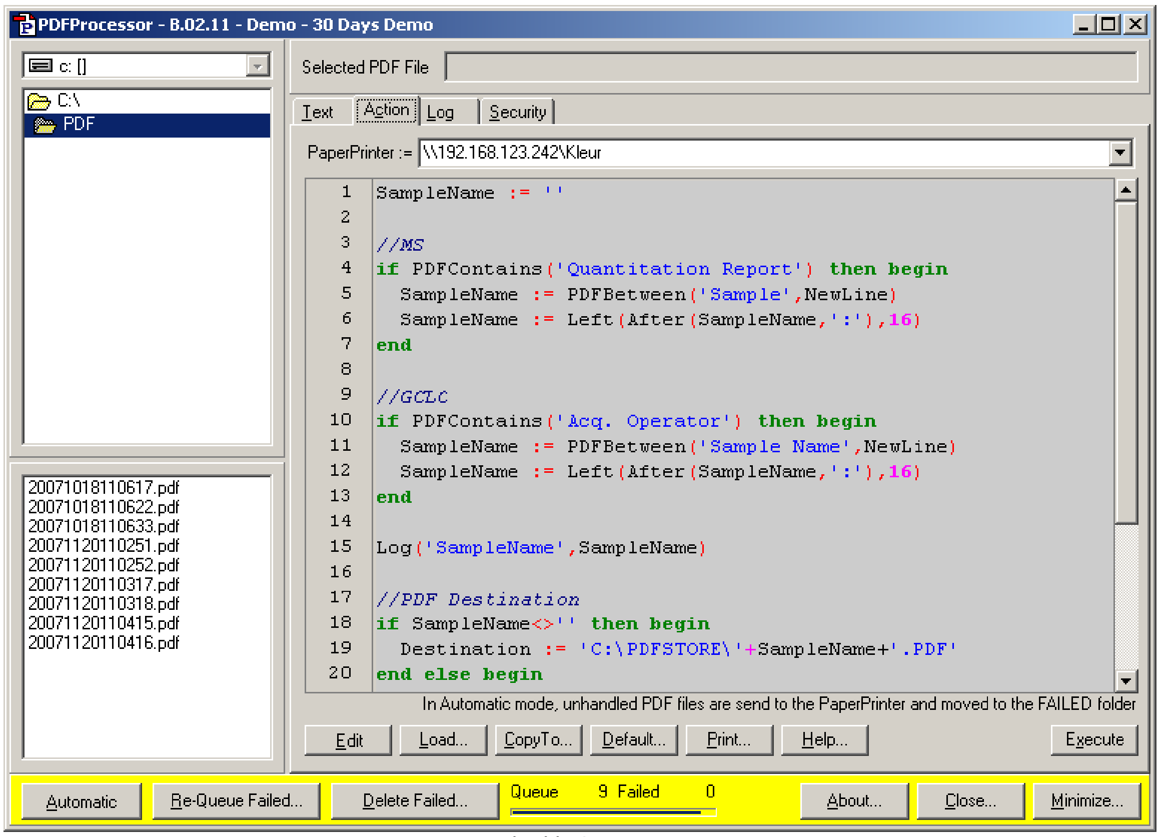
Purpose

PDFProcessor processes PDF files in a folder. The text within the PDF file is processed with a Pascal Script. Many commands are available to extract key values from the text, print a hardcopy of the pdf file, and rename and move the pdf file to a server folder all based on the extracted keywords. PDF files will not be overwritten, but a version number will be appended on collision. A script is included for the standard Agilent ChemStation reports. The script can be modified to fit any report in your laboratory that is produced by any kind or brand of instrument.

Schema


Instrument Software
Printing
PDFCreator (default printer)
Auto-Save PDF files
Local Folder C:\PDF\
Monitor
PDFProcessor
Optional PDF Actions
Extract Keywords
Hardcopy Printout
Save PDF based on Keywords
Consistent Paper and Electronic records in your laboratories
PDFProcessor
Helping to fullfill the 25% report submission in 'Equivalent Study'
Simultaneous generate a Paper Hardcopy
Simultaneous generate a PDF Electronic copy
Rename the PDF file based on extracted keywords from the PDF content
Move the PDF file based on extracted keywords from the PDF content (with versioning)

Screenshot



Selected PDF File

PaperPrinter := \\192.168.123.242\Kleur

```
SampleName := ''

//MS
if PDFContains('Quantitation Report') then begin
  SampleName := PDFBetween('Sample',NewLine)
  SampleName := Left(After(SampleName,':'),16)
end

//GCLC
if PDFContains('Acq. Operator') then begin
  SampleName := PDFBetween('Sample Name',NewLine)
  SampleName := Left(After(SampleName,':'),16)
end

Log('SampleName',SampleName)

//PDF Destination
if SampleName<>'' then begin
  Destination := 'C:\PDFSTORE\'+SampleName+'.PDF'
end else begin
```

In Automatic mode, unhandled PDF files are send to the PaperPrinter and moved to the FAILED folder

Queue 9 Failed 0

```
20071018110617.pdf
20071018110622.pdf
20071018110633.pdf
20071120110251.pdf
20071120110252.pdf
20071120110317.pdf
20071120110318.pdf
20071120110415.pdf
20071120110416.pdf
```

Flexible Scripting

Installation

Download URL

http://www.waleson.nl/products/pdfprocessor

License file

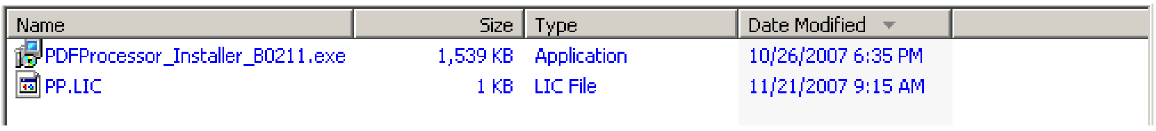
PDFProcessor is protected by a license file. Without license file, PDFProcessor operates in demo mode (confirmation dialogs). Automatic license installation occurs if the license file exists next to the setup program. It is possible to install the license file after the installation.

PDF Virtual Printer

First install a PDF virtual printer. Any PDF virtual printer can be uses that can automatically save PDF files with unique names to C:\PDF. Unfortunately many PDF virtual printers display dialogs during PDF creation, or overwrite existing PDF files. Fortunately PDFCreator exists as a freeware PDF virtual printer with the required features. Download URL http://sourceforge.net/projects/pdfcreator

Installer

Execute the setup program PDFProcessor_Installer_xxxxx.exe

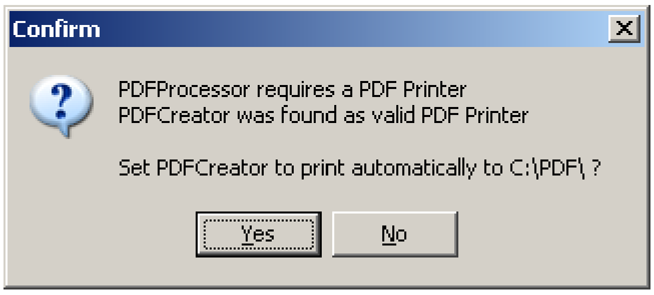


Name	Size	Type	Date Modified
PDFProcessor_Installer_B0211.exe	1,539 KB	Application	10/26/2007 6:35 PM
PP.LIC	1 KB	LIC File	11/21/2007 9:15 AM

A shortcut is added to the windows programs startup group. PDFProcessor automatically starts after user logon to windows.

Configuration

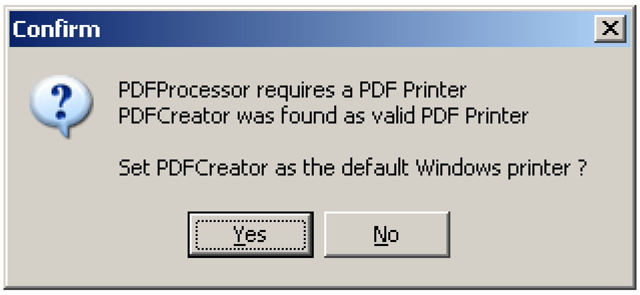
After startup, PDFProcessor checks whether PDFCreator is installed and checks its configuration. PDFProcessor contains specific functions to automatically configure PDFCreator version 0.9.3.



Confirm

PDFProcessor requires a PDF Printer
PDFCreator was found as valid PDF Printer

Set PDFCreator to print automatically to C:\PDF\ ?



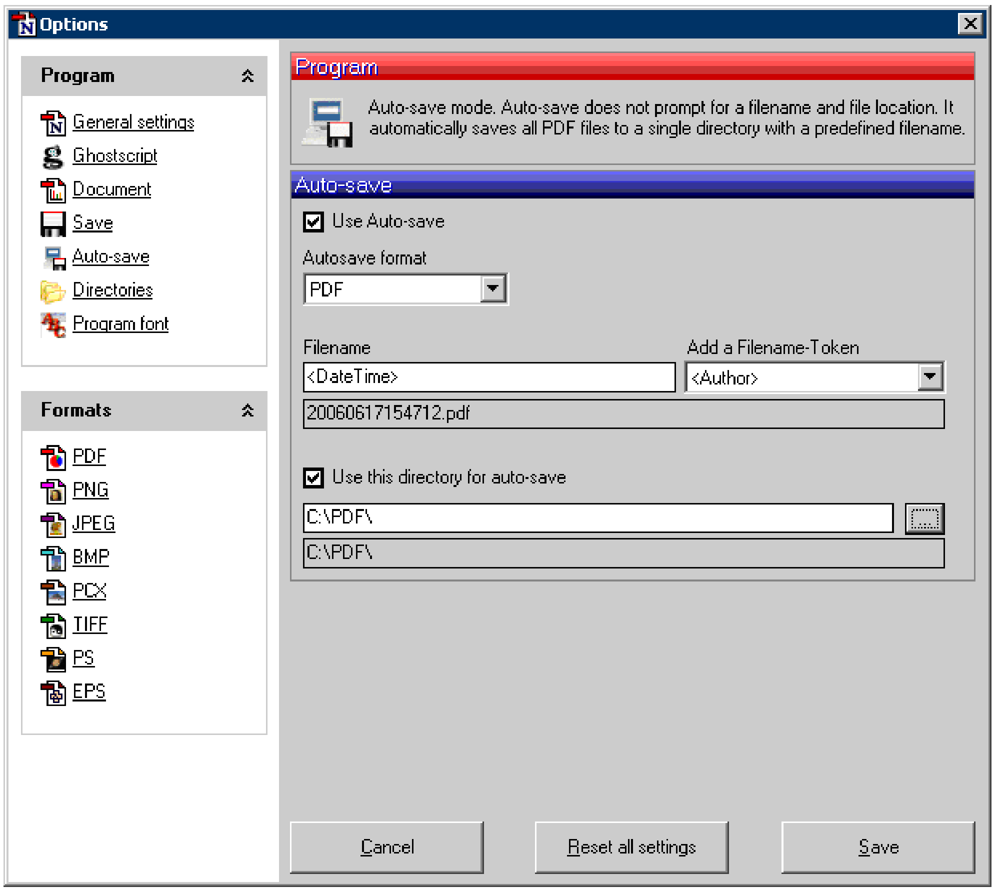
Confirm

PDFProcessor requires a PDF Printer
PDFCreator was found as valid PDF Printer

Set PDFCreator as the default Windows printer ?

Manual setup

Set PDFCreator as the default printer, and configure Auto-Save



Program

Auto-save mode. Auto-save does not prompt for a filename and file location. It automatically saves all PDF files to a single directory with a predefined filename.

Auto-save

☑ Use Auto-save

Autosave format: PDF

Filename: <DateTime>

Add a Filename-Token: <Author>

20060617154712.pdf

☑ Use this directory for auto-save

C:\PDF\

C:\PDF\

Concept

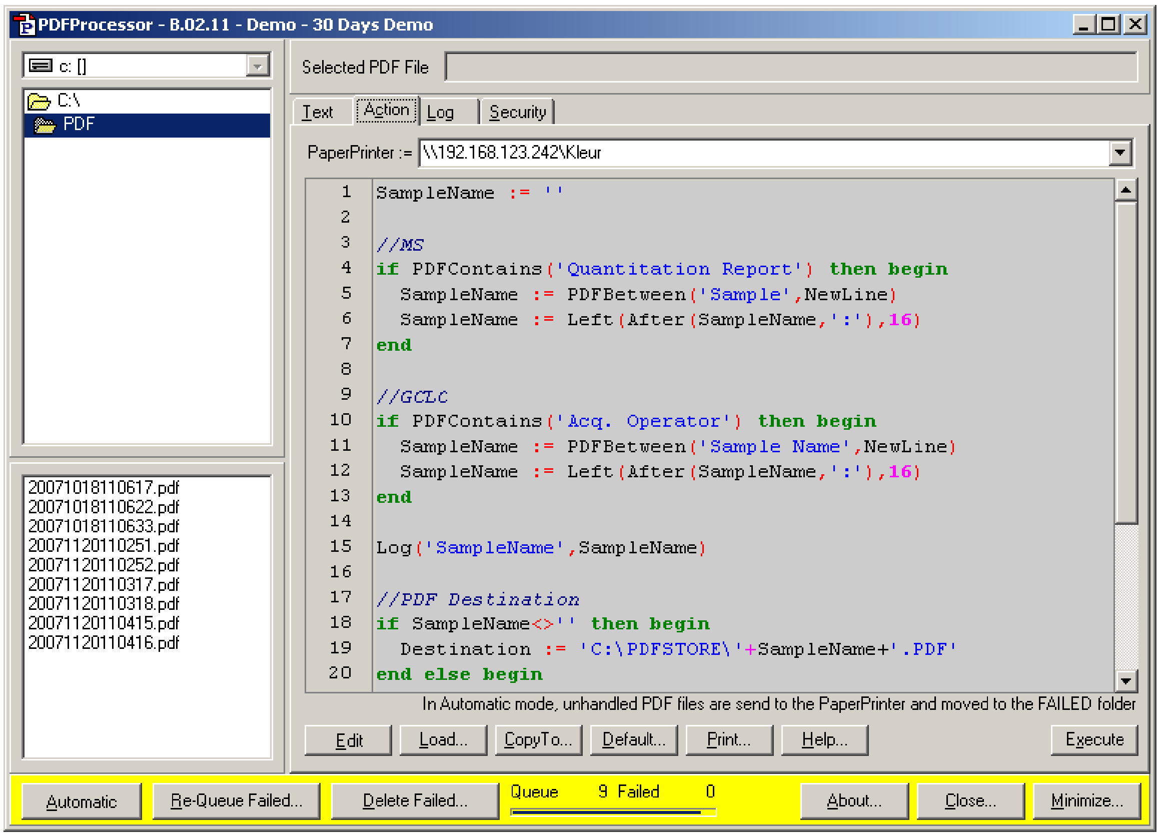
PDFProcessor monitors a selected folder for new PDF files. Each PDF file will be loaded and text extracted. A Pascal script will interpret the text and print or move the PDF file.

PDF Flow

A. Chemstation Prints to Default Printer
B. PDFCreator produces PDF files in C:\PDF
C. PDFProcessor Monitors C:\PDF and interprets the text in each PDF file and can
   1. Print a hardcopy to a selected printer
   2. Rename the PDF file to the Sample Name found in the PDF text
   3. Move the PDF file to a specified server folder

Action Script
The PDFProcessor Action script can be reset to factory defaults. The factory defaults depend on the License File. Customers can ask to include their specific default Action script.

The action script is based on the Pascal script language. Many functions are added to assist PDF text search and PDF file handling. The help file describes the functions.



20071018110617.pdf
20071018110622.pdf
20071018110633.pdf
20071120110251.pdf
20071120110252.pdf
20071120110317.pdf
20071120110318.pdf
20071120110415.pdf
20071120110416.pdf

PaperPrinter := \\192.168.123.242\Kleur

```
SampleName := ''

//MS
if PDFContains('Quantitation Report') then begin
  SampleName := PDFBetween('Sample',NewLine)
  SampleName := Left(After(SampleName,':'),16)
end

//GCLC
if PDFContains('Acq. Operator') then begin
  SampleName := PDFBetween('Sample Name',NewLine)
  SampleName := Left(After(SampleName,':'),16)
end

Log('SampleName',SampleName)

//PDF Destination
if SampleName<>'' then begin
  Destination := 'C:\PDFSTORE\'+SampleName+'.PDF'
end else begin
```

In Automatic mode, unhandled PDF files are send to the PaperPrinter and moved to the FAILED folder

Queue 9 Failed 0

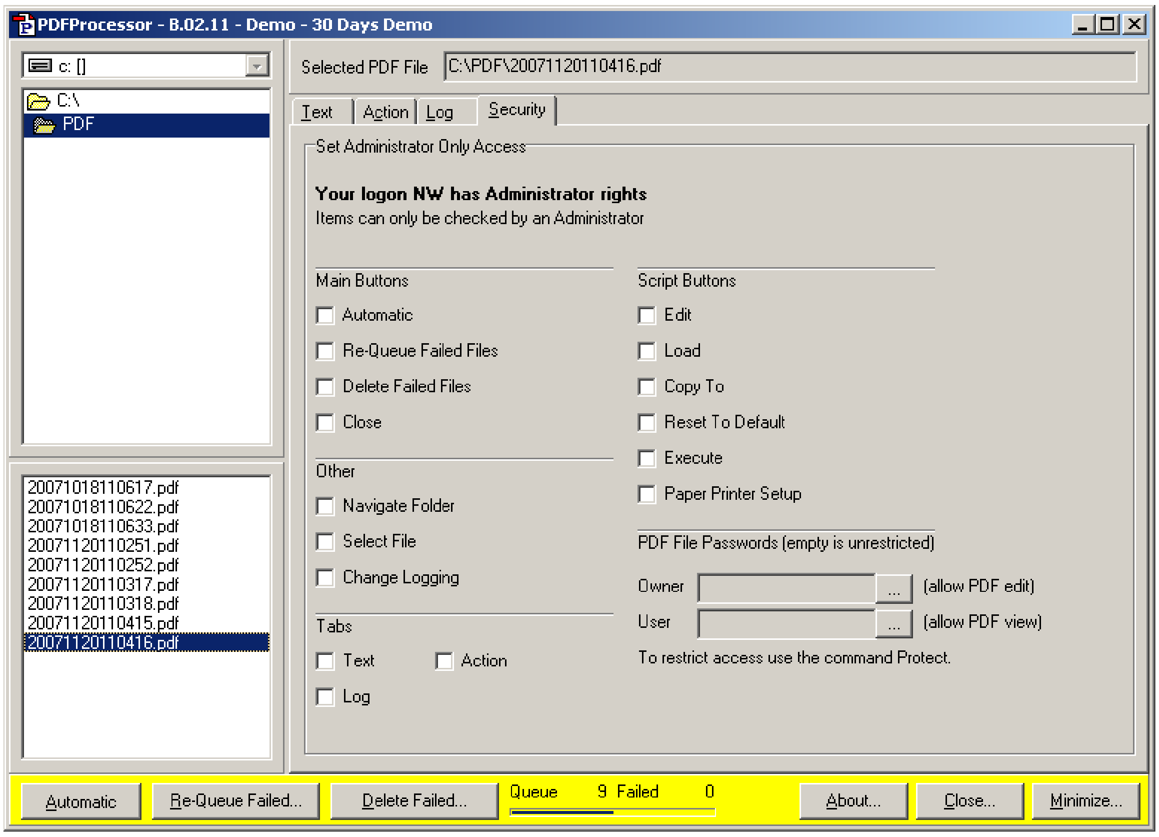
Security

Users that are part of the Administrator group have unlimited access to the PDFProcessor features.
Other users can be restricted by an Administrator by setting specific checkboxes.

PDF files can be password protected with an Owner and or a User password.
The Owner password protects the PDF content for editing.
The User password protects the PDF content for viewing.

If PDFProcessor must act on protected PDF files, the owner or user passwords must be set, to allow PDFProcessor to edit or read the content.

If PDFProcessor must protect PDF files, the Action script function Protect(Filename) should be used.



Selected PDF File C:\PDF\20071120110416.pdf

20071018110617.pdf
20071018110622.pdf
20071018110633.pdf
20071120110251.pdf
20071120110252.pdf
20071120110317.pdf
20071120110318.pdf
20071120110415.pdf
20071120110416.pdf

Set Administrator Only Access

Your logon NW has Administrator rights
Items can only be checked by an Administrator

Main Buttons
- [ ] Automatic
- [ ] Re-Queue Failed Files
- [ ] Delete Failed Files
- [ ] Close

Other
- [ ] Navigate Folder
- [ ] Select File
- [ ] Change Logging

Tabs
- [ ] Text
- [ ] Action
- [ ] Log

Script Buttons
- [ ] Edit
- [ ] Load
- [ ] Copy To
- [ ] Reset To Default
- [ ] Execute
- [ ] Paper Printer Setup

PDF File Passwords (empty is unrestricted)

Owner ... (allow PDF edit)
User ... (allow PDF view)

To restrict access use the command Protect.

Queue 9 Failed 0

Standard Pascal Script Syntax

begin .. **end** constructor

procedure and **function** declarations

if .. **then** .. **else** constructor

for .. **to** .. **do** .. **step** constructor

while .. **do** constructor

repeat .. **until** constructor

try .. **except** and **try** .. **finally** blocks

case statements

array constructors (x:=[1, 2, 3];)

^ , * , / , and , + , - , or , <> , >=, <= , = , > , < , div , mod , xor , shl , shr operators

PDFProcessor Specific Pascal Script Commands

ActivateText1
After this command, all text search function will operate on Text1. Text1 and Text2 are the results of two different methods of extracting text from a PDF file.

ActivateText2
After this command, all text search function will operate on Text2. Text1 and Text2 are the results of two different methods of extracting text from a PDF file.

AppendToTextFile(FileName,Text: **string**)
Append Text to a text file.

Execute(FileName,Parameter: **string**)
Executes or opens the specified filename with the optional specified parameters

Log(Text1,Text2: **string**)
Write to Log (blue)

LogErr(Text1,Text2: **string**)
Write to Log (red)

IsAutomatic
Script was started Automatic

PDFName
PDF File Name, will be update after PDFMoveTo

PDFFile
PDF full File Path and Name, will be update after PDFMoveTo

PDFPath
PDF File Path, will be update after PDFMoveTo

PDFText
PDF Text Content

PDFContains(SubStr: **string**)
Test PDF Content for present SubString, case-insensitive

PDFLeft(Count: integer)
Extract first count characters of the PDF Content

PDFRight(Count: integer)
Extract last count characters of the PDF Content

PDFBetween(SubStr1, SubStr2: **string**)
Extract characters of the PDF Content between SubStr1 and SubStr2, case-insensitive

PDFAfter(SubStr: **string**)
Extract characters of the PDF Content after SubStr, case-insensitive

PDFBefore(SubStr: **string**)
Extract characters of the PDF Content before SubStr, case-insensitive

PDFAfterN(SubStr: **string** Count: integer)
Extract characters of the PDF Content after Count occurrences of SubStr, case-insensitive

PDFBeforeN(SubStr: **string** Count: integer)
Extract characters of the PDF Content before Count occurrences of SubStr, case-insensitive

PDFMid(Start, Length: integer)
Extract Count characters of the PDF Content after Start character

PDFREFind(RegExp: **string** Index: integer)
Regular Expression, extract the match within the indexed brackets
Example PDF Content= 'abcd efgh 1234 5678'
PDFREFind('**(\d+).(\d+)**',2) *returns '5678'*
See Regular Expression information on the Internet

PDFREMatch(RegExp: **string**)
Regular Expression, matches the PDF Content
Example PDF Content= 'abcd efgh 1234 5678'
PDFREMatch('**efgh**') *returns true*
See Regular Expression information on the Internet

PDFREReplace(RegExp, Substitute: **string**)
Regular Expression, replaces all matching SubStrings with Substitute
Example PDF Content= 'abcd efgh 1234 5678'
PDFREReplace('**efgh**','a') *returns 'abcd a 1234 5678'*
See Regular Expression information on the Internet

Protect(FileName: **string**)
Protect a PDF file with passwords from the security panel

WaterMarkPages(FileName, Identifier, Text: **string**)
Puts Text string as watermark on every page that contains the Identifier string.

Repaginate(FileName: **string** X, Y: double)
Finds existing page counter text nearest to the bottom within X, Y mm from right lower corner.
Insert the new page counter text on every page at the same position with the original font.

PDFAppendTo(FileName: **string**)
Appends PDF Content to the specified file

CoverPage(FileName, Header, Text: **string**)
Add a cover page with header and text strings at the start of the PDF Content
Text can contain newline (#10) and tab (#9) characters

DeletePages(FileName: **string**; StartPage, EndPage: integer)
Delete specified pages

PDFPrintTo(PrinterName: **string**)
Print to the specified printer name. Available printer names are displayed in the Log at startup

AppendTo(Source, Destination: **string**)
Appends source file content to the specified destination file

AppendToPDF(FileName: **string**)
Appends source file to the PDF Content

PrintTo(FileName, PrinterName: **string**)
Print to the specified PDF file to the specified printer name. Available printer names are displayed in the Log at startup

DateTimeNow(Format: **string**)
Format Now, Format characters: yy yyyy m mm mmm mmmm d dd ddd dddd h hh n nn s ss z zzz

DateTime(Time, Format: **string**)
Format Time, Format characters: yy yyyy m mm mmm mmmm d dd ddd dddd h hh n nn s ss z zzz

PDFCopyTo(Destination: **string**)
Copy PDF file to Destination
Invalid characters are replaced with _
Multiple backslashes are reduced singles
Force creation of complete Destination directory
Ensure .PDF extension
A _0000 counter is appended to prevent file overwriting
After an error, the file is moved to a FAILED sub-folder
After an un-recoverable error, Automation is paused

PDFMoveTo(Destination: **string**)
Move PDF file to Destination
Invalid characters are replaced with _
Multiple backslashes are reduced singles
Force creation of complete Destination directory
Ensure .PDF extension
A _0000 counter is appended to prevent file overwriting
After an error, the file is moved to a FAILED sub-folder
After an un-recoverable error, Automation is paused
Will update the PDFPath and PDFFile variables

LastPDFFile(Destination: **string**)
Find the last Destination PDF file eventual with an appended _0000 counter

PDFPrint
Print to the PaperPrinter

Print(FileName: **string**)
Print the specified file to the PaperPrinter

PDFDelete
Delete the PDF file

Contains(Text, SubStr: **string**)
Test Text for present SubString, case-insensitive

Left(Text: **string** N: integer)
Extract first count characters of the PDF Content

Right(Text: **string** N: integer)
Extract last count characters of the PDF Content

Between(Text, SubStr1, SubStr2: **string**)
Extract characters of Text between SubStr1 and SubStr2, case-insensitive

After(Text, SubStr: **string**)
Extract characters of Text after SubStr, case-insensitive

Before(Text, SubStr: **string**)
Extract characters of Text before SubStr, case-insensitive

AfterN(Text, SubStr: **string** Count: integer)
Extract characters of Text after Count occurrences of SubStr, case-insensitive

BeforeN(Text, SubStr: **string** Count: integer)
Extract characters of Text before Count occurrences of SubStr, case-insensitive

Mid(Text: **string** Start, Length: integer)
Extract Count characters of Text after Start character

PosI(SubStr, Text: string)
Find position of SubStr in Text, case-insensitive

SameText(Text1, Text2: string)
Compares Text1 and Tex2, case-insensitive

Token(Text, Delimiter: **string** Index: integer)
Split Text on each occurrence of the Delimiter string and return the SubString at Index, a negative number will start counting from the end, case-insensitive
Token('ABC DEF GHI JKL MNO',' ',2) *returns 'DEF'*
Token('ABC DEF GHI JKL MNO',' ',-2) *returns 'JKL'*

Tokens(Text, Delimiter: string; Count: integer);
Split Text on each occurrence of the Delimiter string and return SubStrings up to Count separated by the delimiter, a negative number will start counting from the end, case-insensitive
Tokens('ABC DEF GHI JKL MNO',' ',2) *returns 'ABC DEF'*
Tokens('ABC DEF GHI JKL MNO',' ',-2) *returns 'ABC DEF GHI JKL'*

REFind(Text, RegExp: **string** Index: integer)
Regular Expression, extract the match within the indexed brackets
REFind('abcd efgh 1234 5678', '**(\d+).(\d+)**',2) *returns '5678'*
See Regular Expression information on the Internet

REMatch(Text, RegExp: **string**)
Regular Expression, matches the PDF Content
REMatch('abcd efgh 1234 5678', '**efgh**') *returns true*
See Regular Expression information on the Internet

REReplace(Text, RegExp, Substitute: **string**)
Regular Expression, replaces all matching SubStrings with Substitute
REReplace('abcd efgh 1234 5678', '**efgh**','a') *returns 'abcd a 1234 5678'*
See Regular Expression information on the Internet

TrimAll(Text: **string**)
Remove all leading and trailing white space characters (space, tab, linefeed, etc)

TrimBackSlash(FileName: **string**)
Remove all trailing and double backslashes

ParentFolder(FilePath: string);
Return the Parent Folder

SubDir(FilePath: **string**; Index: integer)
Return the SubDirectory at Index, a negative number will start counting from the end
SubDirs('C:\ABC\DEF\GHI',2) *returns 'ABC'*
SubDirs('C:\ABC\DEF\GHI',-2) *returns 'DEF'*

SubDirs(FilePath: string; Count: integer)
Return the SubDirectories up to Count, a negative number will start counting from the end
SubDirs('C:\ABC\DEF\GHI',2) *returns 'C:|ABC'*
SubDirs('C:\ABC\DEF\GHI',-2) *returns 'C:|ABC|DEF'*

FixFileName(FileName: **string**)
Remove illegal characters: \/:*?"<>|

FixFilePath(FilePath: **string**)
Remove illegal characters: /:*?"<>|

CopyFile(Source, Destination: **string**)
Copy Source File to Destination

DeleteFile(Path: **string**)
Delete File

FileExists(Path: **string**)
Test File for Existence

AppendToTextFile (Path, Text: **string**)
Append Text to text file. Create file if not exists.

DirectoryExists(Path: **string**)
Test Directory for Existence

NewLine
NewLine character Useful for searching Line Breaks

TextReplace(Text, Old, New: **string**)
Replace all occurrences of the old sub string with new the substring, case-insensitive

QueueFolder
Queue Folder path

FailedFolder
Failed Folder path

ComputerName
Local Computer Name

UserName
Windows User Login Name

Standard Pascal Script Commands

Abs(X: real)
AnsiCompareStr(S1, S2: **string**)
AnsiCompareText(S1, S2: **string**)
AnsiLowerCase(S: **string**)
AnsiUpperCase(S: **string**)
Append(F: text)
ArcTan(X: real)
Assigned(P: pointer)
AssignFile(F: file FileName: **string**)
Beep
Chdir(S: **string**)
Chr(X: byte)
CloseFile(F: file)
CompareStr(S1, S2: **string**)
CompareText(S1, S2: **string**)
Copy(S: **string** Index, Count: integer):
Cos(X: real)
CreateOleObject(**const** ClassName: **string**)
Date
DateTimeToStr(DateTime: TDateTime)
DateToStr(DateTime: TDateTime)
DayOfWeek(DateTime: TDateTime)
Dec(X: integer [N: integer])
DecodeDate(DateTime: TDateTime Year, Month, Date: integer)
DecodeTime(DateTime: TDateTime Hour, Min, Sec, MSec: integer)
Delete(S: **string** Index, Count: integer)
EncodeDate(Year, Month, Date: integer)
EncodeTime(Hour, Min, Sec, MSec: integer):
EOF(F: file)
Exp(X: real)
FilePos(F: file)
FileSize(F: file)
FloatToStr(Value: real)
Format(Format: **string** Arg: array of **const**)
FormatDateTime(format: **string** DateTime: TDateTime)
FormatFloat(Format: **string** X: real)
Frac(X: real)
GetActiveOleObject(**const** ClassName: **string**)
High(X: array)
Inc(X: integer [N: integer])
IncMonth(DateTime: TDateTime [N: integer])
InputQuery(Caption, Prompt, Value: **string**)
Insert(SubStr, Text: **string** Index: integer)
Int(X: real)
IntToHex(X: integer)
IntToStr(X: integer)
IsLeapYear(Year: integer)
IsValidIdent(Ident: **string**)
Length(S: **string**)
Ln(X: real)
Low(X: array)
LowerCase(s: **string**)
Now
Odd(X: integer)
Ord(X: ordinal)
Pos(SubStr, Text: **string**)
Raise
Random([Range: integer])
ReadLn(F: file [**...**VN])
Reset(F: file [RecSize: integer])

Rewrite(F: file [RecSize: integer])
Round(X: real)
ShowMessage(Text: **string**)
Sin(X: real)
Sqr(X: real)
Sqrt(X: real)
StrToDate(S: **string**)
StrToDateTime(S: **string**)
StrToFloat(S: **string**)
StrToInt(S: **string**)
StrToIntDef(S: **string** Def: integer)
StrToTime(S: **string**)
Time
TimeToStr(DateTime: TDateTime)
Trim(S: **string**)
TrimLeft(S: **string**)
TrimRight(S: **string**)
Trunc(X: real)
UpperCase(S: **string**)
VarArrayCreate(**const** Bounds: array of Integer VarType: TVarType)
VarArrayHighBound(**const** A: Variant Dim: Integer)
VarArrayLowBound(**const** A: Variant Dim: Integer)
VarIsNull(**const** V: Variant)
VarToStr(**const** V: Variant)
Write(F: file P1 [..PN])
WriteLn(F: file P1 [..PN])

Standard Pascal Script Structure

Script structure is made of two major blocks: a) procedure and function declarations and b) main block. Both are optional, but at least one should be present in script. There is no need for main block to be inside begin..end. It could be a single statement. Some examples:

SCRIPT 1:
```
procedure DoSomething;
begin
   CallSomething;
end;
begin
   CallSomethingElse;
end;
```

SCRIPT 2:
```
begin
   CallSomethingElse;
end;
```

SCRIPT 3:
```
function MyFunction;
begin
   result:='Ok!';
end;
```

SCRIPT 4:
```
CallSomethingElse;
```

Unlike in pascal, statements terminated by ";" character is not required. begin..end blocks are allowed to group statements.

Identifiers

Identifier names in script (variable names, function and procedure names, etc.) follow the most common rules in pascal: should begin with a character (a..z or A..Z), or '_', and can be followed by alphanumeric chars or '_' char. Cannot contain any other character os spaces. Valid identifiers:

```
VarName
_Some
V1A2
_____Some____
```

Invalid identifiers:

```
2Var
My Name
Some-more
This,is,not,valid
```

Assign statements

Just like in Pascal, assign statements (assign a value or expression result to a variable or object property) are built using ":=". Examples:

```
MyVar:=2;
Button.Caption:='This ' + 'is ok.';
```

Character strings

strings (sequence of characters) are declared in pascal using single quote (') character. Double quotes (") are not used. You can also use #nn to declare a character inside a string. There is no need to use '+' operator to add a character to a string. Some examples:

```
A:='This is a text';
Str:='Text '+'concat';
B:='String with CR and LF char at the end'#13#10;
C:='String with '#33#34' characters in the middle';
```

Comments

Comments can be inserted inside script. You can use // chars or (* *) or { } blocks. Using // char the comment will finish at the end of line.

```
//This is a comment before ShowMessage
ShowMessage('Ok');
(* This is another comment *)
ShowMessage('More ok!');
{
  And this is a comment
  with two lines
}
ShowMessage('End of okays');
```

Variables

There is no need to declare variable types in script. Thus, you declare variable just using var directive and its name. There is no need to declare variables if scripter property OptionExplicit is set to false. In this case, variables are implicit declared. If you want to have more control over the script, set OptionExplicit property to true. This will raise a compile error if variable is used but not declared in script. Examples:

SCRIPT 1:

```
procedure Msg;
var S;
begin
  S:='Hello world!';
  ShowMessage(S);
end;
```

SCRIPT 2:

```
var A;
begin
  A:=0;
  A:=A+1;
end;
```

SCRIPT 3:

```
var S;
S:='Hello World!';
ShowMessage(S);
```

Note that if script property OptionExplicit is set to false, then var declarations are not necessary in any of scripts above.

Indexes

Strings, arrays and array properties can be indexed using "[" and "]" chars. For example, if Str is a string variable, the expression Str[3] returns the third character in the string denoted by Str, while Str[I + 1] returns the character immediately after the one indexed by I. More examples:

```
MyChar:=MyStr[2];
MyStr[1]:='A';
MyArray[1,2]:=1530;
Lines.Strings[2]:='Some text';
```

Arrays

Script support array constructors and support to variant arrays. To construct an array, use "[" and "]" chars. You can construct multi-index array nesting array constructors. You can then access arrays using indexes. If array is multi-index, separate indexes using ",". If variable is a variant array, script automatically support indexing in that variable. A variable is a variant array is it was assigned using an array constructor, if it is a direct reference to a Delphi variable which is a variant array (see Delphi integration later) or if it was created using VarArrayCreate procedure. Arrays in script are 0-based index.
Some examples:

```
NewArray := [ 2,4,6,8 ];
Num:=NewArray[1]; //Num receives "4"
MultiArray := [ ['green','red','blue'] , ['apple','orange','lemon'] ];
Str:=MultiArray[0,2]; //Str receives 'blue'
MultiArray[1,1]:='new orange';
```

if statements

There are two forms of if statement: if...then and the if...then...else. Like normal pascal, if the if expression is true, the statement (or block) is executed. If there is else part and expression is false, statement (or block) after else is execute.
Examples:

```
if J <> 0 then
  Result := I/J;

if J = 0 then
  Exit
else
  Result := I/J;

if J <> 0 then begin
  Result := I/J;
  Count := Count + 1;
end else
  Done := True;
```

while statements

A while statement is used to repeat a statement or a block, while a control condition (expression) is evaluated as true. The control condition is evaluated before the statement. Hence, if the constrol condition is false at first iteration, the statement sequence is never executed. The while statement executes its constituent statement (or block) repeatedly, testing expression before each iteration. As long as expression returns True, execution continues.
Examples:

```
while Data[I] <> X do
  I := I + 1;

while I > 0 do begin
  if Odd(I) then
           Z := Z * X;
  I := I div 2;
  X := Sqr(X);
end;

while not Eof(InputFile) do begin
  Readln(InputFile, Line);
  Process(Line);
end;
```

repeat statements

The syntax of a repeat statement is repeat statement1; ...; statementn; until expression where expression returns a Boolean value. The repeat statement executes its sequence of constituent statements continually, testing expression after each iteration. When expression returns True, the repeat statement terminates. The sequence is always executed at least once because expression is not evaluated until after the first iteration. Examples:

```
repeat
  K := I mod J;
  I := J;
  J := K;
until J = 0;

repeat
  Write('Enter a value (0..9): ');
  Readln(I);
until (I >= 0) and (I <= 9);
```

for statements

Scripter support for statements with the following syntax: for counter := initialValue to finalValue do statement For statement set counter to initialValue, repeats execution of statement (or block) and increment value of counter until counter reachs finalValue.
Examples:

SCRIPT 1:

```
for c:=1 to 10 do
  a:=a+c;
```

SCRIPT 2:

```
for i:=a to b do begin
  j:=i^2;
  sum:=sum+j;
end;
```

case statements

Scripter support case statements with following syntax:

```
case selectorExpression of
  caseexpr1: statement1;
  ...
  caseexprn: statementn;
else
  elsestatement;
end
```

if selectorExpression matches the result of one of caseexprn expressions, the respective statement (or block) will be execute. Otherwise, elsestatement will be execute. Else part of case statement is optional. Different from Delphi, case statement in script doesn't need to use only ordinal values. You can use expressions of any type in both selector expression and case expression.
Example:

```
case uppercase(Fruit) of
  'lime': ShowMessage('green');
  'orange': ShowMessage('orange');
  'apple': ShowMessage('red');
else
  ShowMessage('black');
end;
```

function and procedure declaration

Declaration of functions and procedures are similar to Object Pascal in Delphi, with the difference you don't specify variable types. Just like OP, to return function values, use implicited declared result variable. Parameters by reference can also be used, with the restriction mentioned: no need to specify variable types.
Some examples:

```
procedure HelloWord;
begin
  ShowMessage('Hello world!');
end;

procedure UpcaseMessage(Msg);
begin
  ShowMessage(Uppercase(Msg));
end;

function TodayAsString;
begin
  result:=DateToStr(Date);
end;

function Max(A,B);
begin
  if A>B then
  result:=A
else
  result:=B;
end;

procedure SwapValues(var A, B);
Var Temp;
begin
  Temp:=A;
  A:=B;
  B:=Temp;
end;
```