Technologies of Ohio, Inc.

Home of filePro ® Development Software

432 W. Gypsy Lane Road
Bowling Green, OH 43402
Tel. (800) 847-4740
Fax (877) 606-6853
sales@fptech.com

Version 6.1.02 USP New Features

Enhanced DIM

IMPORT and EXPORT commands can now be mapped to an array.

Example:
```
      Then: IMPORT WORD ifile=(fname)
      Then: DIM data(10):ifile(1)     ' data can now be used in place of ifile
      Then: ct(4,.0)="1"
loop  If: ct le "10"
      Then: msgbox data(ct); ct=ct+"1"; goto loop
      Then: close ifile
```

Enhanced COPY, COPY TO, and COPYIN

Each command now allows for any combination of lookups and arrays to copy
data, including mapped/aliased arrays.

Syntax:
```
      COPY lookup          ' Copy the current record to a lookup file
      COPY array           ' Copy the current record to an array
      COPYIN lookup        ' Copy a lookup file record to the current record
      COPYIN array         ' Copy an array to the current record
      COPY lookup TO lookup ' Copy a lookup record to a lookup record
      COPY array TO lookup  ' Copy an array to a lookup record
      COPY lookup TO array  ' Copy a lookup record to an array
      COPY array TO array   ' Copy an array to an array
```

Examples:
```
      (Copy the current record to an array)
      Then: DIM array(10)
      Then: COPY array

      (Copy an IMPORT to the current record)
      Then: IMPORT WORD ifile=(fname)
      Then: DIM data(10):ifile(1)     ' data can now be used in place of ifile
      Then: COPYIN data               ' Copy the import to the current record
      Then: close ifile

      (Copy a lookup record to an EXPORT)
      Then: EXPORT WORD ofile=(fname)
      Then: DIM data(10):ofile(1)     ' data can now be used in place of ofile
      Then: lookup inv=invoices r=(rec) -nx
      Then: COPY inv TO data
      Then: close ofile
      Then: close inv
```

COPY() Command

```
n = COPY(array1, array2 [,src [,dest [,len]]])
```
Function to copy data between arrays. Returns the number of elements copied from array1 to array2.

Parameters:

array1: Array to copy from.
array2: Array to copy to.
src: The array index to start copying from array1.
dest: The array index to start copying to in array2.
len: The number of elements to copy from array1 to array2.

If no optional parameters are provided COPY() will copy as many items from array1 that will fit into array 2. Parameters src and dest default to the first index of each array. Parameter len defaults to the entire array length.

Example:

```
Then: DIM fruit(3)
Then: DIM food(3)
Then: fruit["1"]="Apple"; fruit["2"]="Orange"; fruit["3"]="Pear"
Then: x=COPY(fruit,food,"1","1","2")
```
(The food array will contain "Apple", "Orange", and "")

XML import and export

filePro now has the ability to import and export XML files.

Export:

XML [id] :CR fname	- Creates an XML file. The id is optional and defaults to "0" if only one file is open at a time. If two or more are open, the id must be supplied ("0"-"99")
XML [id] :CR-|:CL	- Closes an open XML file.
XML [id] :EL name	- Starts an element in an XML file.
XML [id] :EL-	- Closes an element.
XML [id] :AT name value	- Adds an attribute to an XML element.
XML [id] :TX text	- Adds a text element to an XML document.

Example:

```
Then: XML :CR "/tmp/myfile.xml"
Then: XML :EL "EmployeeData"
Then: XML :EL "employee"
Then: XML :AT "id" "21"
Then: XML :EL "firstName"
Then: XML :TX "Tom"
Then: XML :EL-
Then: XML :EL "lastName"
Then: XML :TX "Anderson"
Then: XML :EL-
Then: XML :EL-
Then: XML :EL-
Then: ML :CL
```

Output:
```
<?xml version="1.0"?>
<EmployeeData>
  <employee id="21">
    <firstName>Tom</firstName>
    <lastName>Anderson</lastName>
  </employee>
</EmployeeData>
```

Import:

XML [id] :RO fname	- Opens an XML file for reading. The id is optional and defaults to "0" if only one file is open at a time. If two or more are open, the id must be supplied ("0"-"99")
v = XML [id] :GV key [attr]	- Get a value from an XML file using a path to a key. An attribute name can optionally be provided to return an attribute value rather than the text element value.

Keys are a way to reference part of an XML document using dot syntax. An example of dot syntax would be a key, such as "name.first" or "age". There are reserved symbols used in key syntax that can be used to retrieve certain values from the XML:

'#' is used to get the number of child elements inside of an element.
'@' is used to specify a literal, or if at the end of the path, get the
    name of the current object.

Index positions can also be used to reference specific elements by numeric position inside of an XML document. Indexes in Key Syntax start at position 1.

x = XML :GV "food.10" will attempt to find the tenth (10) item inside
    a food element.

x = XML :GV "food.@10" will attempt to find a key named "10" inside a
    food element and return its value.

x = XML :GV "food.fruit[10]" will attempt to find the tenth (10) fruit
    element inside of the food element and return its value.

x = XML :GV "food.fruit[#]" will return the number of fruit elements inside
    of the food element.

Example:
Given the following XML, here are example commands and what they return.
```
<?xml version="1.0"?>
<EmployeeData>
  <employee id="21">
    <firstName>Tom</firstName>
    <lastName>Anderson</lastName>
  </employee>
  <employee id="99">
    <firstName>Tiffany</firstName>
    <lastName>Anderson</lastName>
  </employee>
</EmployeeData>
```

```
Then: XML :RO "/tmp/myfile.xml"                 ' open the XML file for reading
Then: x=XML :GV "EmployeeData.employee.firstName"  ' x contains "Tom"
Then: x=XML :GV "EmployeeData.employee[1]" "id"    ' x contains "21"
Then: x=XML :GV "EmployeeData.employee.1.@"        ' x contains "firstName"
Then: x=XML :GV "EmployeeData.#"                   ' x contains "2"
Then: x=XML :GV "EmployeeData.2.firstName"         ' x contains "Tiffany"
Then: x=XML :GV "EmployeeData.2" "id"              ' x contains "99"
Then: XML :CL                                       ' close the XML file
```

filePro now has support for basic loops.

FOR loop
A loop that runs from a value to a value. Built in edits are supported.
If a STEP value is not supplied, filePro will determine a STEP value
based on the FROM and TO expression values. A FROM value that is less
than a TO value will result in a positive STEP ("1"). If FROM is greater
than TO the STEP value will be negative ("-1").

Each iteration of the loop will update the value of "f", incrementing by
STEP, and goto the label specified by DO.

Syntax:
   FOR f[(len,edit)] FROM exp TO exp [STEP exp] DO label

Example:
```
     Then: FOR f(10,.0) FROM "1" TO "10" STEP "1" DO lp1; goto en1
lp1   If:
     Then: msgbox f     ' print the value of "f" from 1 to 10
     Then: end
en1   If:
     Then: FOR d(10,mdyy/) FROM "12/01/2024" TO "12/31/2024" DO lp2; goto en2
lp2   If:
     Then: msgbox d     ' print the value of "d" from 12/01/2024 to 12/31/2024
     Then: end
en2   If:
     Then: end
```

Note: The FROM, TO, and STEP expressions are evaluated once when the
      loop is first executed. Changing these values once the loop starts
      executing will not change how the loop runs.

WHILE loop
A loop that runs while the condition is true. Each iteration checks the
condition (cnd) and while the value is true goes to the label specified by
DO. A condition can be an IF expression or label.

Syntax:
   WHILE cnd DO label

Example:
```
     Then: declare total(10,.0)
     Then: total="0"
     Then: lookup inv=invoice r=(rec) -nx
     Then: WHILE inv DO lp1; goto en1
lp1   If:
     Then: total=total+inv(1)
     Then: getnext inv
     Then: end
en1   If:
     Then: close inv; end
```

LOOP ... WHILE|UNTIL
A loop that runs while the condition is true (WHILE) or until the condition
is true (UNTIL). Each iteration starts by going to the label specified by
DO, then the condition is checked and the loop either continues or
terminates based on the value of the condition. A condition can be an IF
expression or label.

Syntax:
```
LOOP label WHILE cnd
LOOP label UNTIL cnd
```

Example:
```
    Then: i(10,.0)="10"
    Then: LOOP lp1 WHILE i gt "0"; goto en1
lp1   If:
    Then: i=i-"1";
    Then: end
en1   If:
    Then: end
```


BREAK command
BREAK can be used inside of a loop to terminate its execution early.

Example:
```
    Then: i(10,.0)="10"
    Then: LOOP lp1 WHILE i gt "0"; goto en1
lp1   If: i eq "5"
    Then: BREAK            ' Terminate the loop early when i equals 5
    Then: i=i-"1";
    Then: end
en1   If:
    Then: end
```


New XLSX Functions
```
e = XL_FREEZEPANE([row [, col [, sheet]]])
```

Parameters:
    row: Row to split the cell (0 indexed)
    col: Column to split the cell (0 indexed)
    sheet: Handle to sheet to freeze the cell on. Leave blank, "0", or "-1" to
           use the default sheet.

Notes:
Returns "1" on success and "-1" on error. XL_ERROR() can be called to
return the last error.

The split is specified at the top or left of a cell and uses zero based
indexing. Therefore to freeze the first row of a worksheet it is
necessary to specify the split at row 2.

You can set one of the row and col parameters as zero if you do not want
either a vertical or horizontal split.

e = XL_FREEZEPANE2([cell [, sheet]])

Parameters:
   cell: The Excel style cell to freeze the cell. e.g. "A1" "D6" "F6".
   sheet: Handle to sheet to freeze the cell on. Leave blank, "0", or "-1"
       to use the default sheet.

Notes:
   Returns "1" on success and "-1" on error. XL_ERROR() can be called to
   return the last error.

   Split is specified at the top or left of a cell and uses zero based
   indexing. Therefore to freeze the first row of a worksheet it is
   necessary to specify the split at row 2.

   You can set one of the row and col parameters as zero if you do not want
   either a vertical or horizontal split.

e = XL_SPLITPANE([vertical [, horizontal [, sheet]]])

Parameters:
   vertical: The position for the vertical split. e.g. "1", "12.5", "15"
   horizontal: The position for the horizontal split. e.g. "1", "12.5",
                "15"
   sheet: Handle to sheet to freeze the cell on. Leave blank, "0", or
                "-1" to use the default sheet.

Notes:
   Returns "1" on success and "-1" on error. XL_ERROR() can be called to
   return the last error.

   This function divides a worksheet into horizontal or vertical regions
   known as panes. This function is different from the XL_FREEZEPANE
   function in that the splits between the panes will be visible to the
   user and each pane will have its own scroll bars.

   The parameters vertical and horizontal are used to specify the vertical
   and horizontal position of the split. The units for vertical and
   horizontal are the same as those used by Excel to specify row height and
   column width. However, the vertical and horizontal units are different
   from each other. Therefore you must specify the vertical and horizontal
   parameters in terms of the row heights and column widths that you have
   set or the default values which are 15 for a row and 8.43 for a column.

Environmental variable PFXLASCII
   Default OFF. If enabled, any non-printable ASCII characters will be
   automatically stripped from data wheninserted into an XLSX document.

LOOKUP Enhancement
   **Added preliminary support for variable index selection in lookups. You can
   now use an expression to select which index to use for a lookup at runtime.**

   Example:
       Then: declare index(1,*); index="A"
       Then: lookup myfile = test k=aa i=(index) -nx

   Note: The lookup wizard has not been updated at this time. Support will be
        added in a future version.

Fill-In-The-Blank PDFs

Added support for read-only PDF fields when generating a fill-in-the-blank PDF document. Each field type now contains an option to flag the field as read-only.

New CLI Option for Clerk, Report, and Index Maintenance

Added -MN command line option to hide [NONE] qualifier from the qualifier list in dclerk, rclerk, dreport, rreport, and dxmaint. Same as PFNOQUAL=OFF.

READSCREEN() Enhancement

Added an option "7" to READSCREEN() to get cursor path.

Dynamically sized, returns a list of fields separated by colons,
e.g. " 1: 2:TAB:aa :".

ENCODE() and DECODE() Enhancement

Added new option to ENCODE() and DECODE(), "URL", to handle URL percent encoding.

Failure to decode will return an empty string.

Example:
```
Then: ' x contains "Hello%2C%20World%21"
Then: x=ENCODE("URL","Hello, World!")
Then: ' x contains "Hello, World!"
Then: x=DECODE("URL","Hello%2C%20World%21")
```

READMAP() Command
s = READMAP(file)

Takes the name of a filePro file and returns information from the first line of the map file. On error or if the file is an invalid filePro file, the function will return blank.

Parameters:
file: The name of a filePro file.

Example return value:
Each section is 5 characters long by default.
"type:kreclen:dreclen:keyflds:"
Where:
type is the filePro map type; map, map2, odbc, alien.
kreclen is the key record length for a record in the file.
dreclen is the data record length for a record in the file.
keyflds is the number of key fields for a record in the file.
e.g. "map : 100:   0:   10:"

PRINTCODE() Command
    x = PRINTCODE(code [,flag])

    Returns either the expanded print code for the current printer or its
    description.

    Parameters:
        code: The print code number to evaluate.
        flag: 0 - Return the "raw" expanded print code.
              1 - Return the comment for the print code.

    Example:
    Given a print code table containing the following entries:

    Number   Sequence   Description
       1   %2 %3   Initialize printer
       2   <page>   New Page
       3   <font name="Courier">   Set Font

      If: ' x will contain '<page> <font name="Courier">'
    Then: x = PRINTCODE("1")
       If: ' x will contain '<page> <font name="Courier">'
    Then: x = PRINTCODE("1","0")
       If: ' x will contain 'New Page'
    Then: x = PRINTCODE("2","1")

GETLOCKS() Command
    n = GETLOCKS(array,lookup)

    Returns the number of elements populated in the array. Fills the array with
    locked record information for a given lookup. Use '-' for current file. If
    passing a multi-dimensional, the array must point to the final sub array OR
    the second to last. This allows us to return the PID and Username/UID for
    the given lock. Returns "0" on Windows.

    Restrictions:
        Linux|Unix|FreeBSD Only.

    Parameters:
        array: An array to place the locked record information in.
        lookup: The lookup to use to check a filePro file for locked records.

    Examples:
        Then: ' Fill array with the record number of locked records in the file
        Then: dim array(10)(10,.0)
        Then: ' x will contain the number of locks on the
        Then: x = GETLOCKS(array,-) ' file that will fit into array

        Then: ' Fill array with locked records including PID and Username/UID
        Then: dim array(10,3)
        Then: ' x will contain the number of locks on the
        Then: x = GETLOCKS(array,-) ' file that will fit into array

        In the second example each "row" of the array will contain the locked
        record number, the PID of the locking process, and the user holding the
        lock. e.g.

        Then: x = array["1","1"] ' x holds the record number
        Then: x = array["1","2"] ' x holds the PID
        Then: x = array["1","3"] ' x holds the username OR UID

ISDIR Command

n = ISDIR(fname)
Test if a given path is a directory. Returns "1" if the file exists and is a directory. Returns "0" if it is not. Returns a negated system error on failure.

Parameters:
    fname: A path to an on-disk resource.

Note: Shares the same @FSTAT array used by EXISTS() in filePro.

ISFILE Command

n = ISFILE(fname)
Test if a given path is a file. Returns "1" if the file exists and is a file. Returns "0" if it is not. Returns a negated system error on failure.

Parameters:
    fname: A path to an on-disk resource.

Note: Shares the same @FSTAT array used by EXISTS() in filePro.

ISLINK Command

n = ISLINK(fname)
Test if a given path is a link. Returns "1" if the file exists and is a link. Returns "0" if it is not. Returns a negated system error on failure.

Parameters:
    fname: A path to an on-disk resource.

Note: Shares the same @FSTAT array used by EXISTS() in filePro.
      ISLINK() always returns "0" on Windows.

Define Processing

Enhanced find and replace with an optional match whole word function. This makes it much easier to find places where variables like "aa" and "zz" are used.

Enhanced F9 search in dcabe/rcabe to allow for whole word searching by using a single quote before the search term, e.g. 'WORD. This makes it much easier to find places where variables like "aa" and "zz" are used.

FPSTAT Command

x = FPSTAT(lookup)
Function to return map information and basic access attributes for a given
filePro lookup.

Parameters:
lookup: A lookup to a filePro file to retrive basic attributes from.
Can be "-" for the current file.

Returns:
kfilesize;dfilesize;mdate;mtime;
Blank on error.

Where:
kfilesize is the total sum of the size of all key segments in bytes.
dfilesize is the total sum of the size of all data segments in bytes.
mdate is the last date a key/data file was modified, e.g. 03/24/2025
mtime is the last time a key/data file was modified, e.g. 02:19:59

Note: The returned values are ONLY for the active qualifier on the lookup.

GETQUAL Command

s = GETQUAL(fname)
Returns a colon delimited list of all qualifiers for the file given by
"fname"

Parameters:
fname: A filePro file name.

Example:
(File invoices has 3 qualifiers 2022, 2023, and 2024)
Then: s=GETQUAL("invoices") ' s will contain "2022   :2023   :2024   :"

n = GETQUAL(array, fname)
Returns the number of qualifiers for the file given by "fname" while filling
"array" with the list of qualifier names.

Parameters:
array: An array to fill with a list of qualifier names.
fname: A filePro file name.

Example:
(File invoices has 3 qualifiers 2022, 2023, and 2024)
Then: DIM quals(10)
Then: n = GETQUAL(quals, "invoices") ' n will contain "3"
Then: q = quals["1"]   ' q will contain 2022
Then: q = quals["2"]   ' q will contain 2023
Then: q = quals["3"]   ' q will contain 2024

Debugging
Updated how 'C' continue works in the debugger.

The debugger should now correctly maintain the "step" mode when switching
between processing and entering and leaving calls. Previously, using
continue while inside a call would take you out of single-step mode when
returning from said call. Now, if you were in single-step mode before a
call, continuing inside of the call will place you back into single step
mode upon returning or entering a new processing table.

Define Menus

Added new F8 options to dmakemenu. You can now move, copy, delete, save, and load menu items inside of dmakemenu.

Expanded menu version from 8 characters to 16 in dmakemenu and runmenu. Using a longer title and running the menu in an older version of filePro will only display the first 8 characters.

Added new environmental variable PFMENUVER=0, Default 0. This globally changes how filePro menus display their version strings.
- **0** - Show menu version as-is.
- **1** - Show filePro version if menu version is blank.
- **2** - Show menu file name if menu version is blank.
- **3** - Always show filePro version.
- **4** - Always show menu file name.

Added pseudo environmental variable @MN that can be used in the version string or menu title to show the menu file name in its place. To use, place $@MN in the menu title or menu version section when designing a menu.

JSON Import and Export

filePro now has the ability to import and export JSON files.

Export:

JSON [id] :CR fname	- Creates a JSON file. The id is optional and defaults to "0" if only one file is open at a time. If two or more are open, the id must be supplied ("0"-"99")
JSON [id] :CR-|:CL	- Closes an open JSON file.
JSON [id] :OB [name]	- Starts an object in a JSON file.
JSON [id] :OB-	- Closes an object.
JSON [id] :AR [name]	- Starts an array in a JSON file.
JSON [id] :AR-	- Closes an array in a JSON file.
JSON [id] :IT name [value]	- Adds an item to a JSON file, if a value is not supplied, the resulting value will be null.
JSON [id] :NO name [value]	- Adds a number to a JSON file, if a value is not supplied, the resulting value will be null.
JSON [id] :BL name [value]	- Adds a boolean value to a JSON file, if a value is not supplied, the resulting value will be null.

Note: Names will be ignored when adding an item, number, or boolean directly to an array.

Example:
```
JSON :CR "/tmp/myfile.json"
JSON :OB
JSON :OB "name"
JSON :IT "first" "Tom"
JSON :IT "last" "Anderson"
JSON :OB-
JSON :NO "age" "37"
JSON :AR "children"
JSON :IT "" "Sara"
JSON :IT "" "Alex"
JSON :IT "" "Jack"
JSON :AR-
JSON :IT "fav.movie" "Deer Hunter"
JSON :OB-
JSON :CL
```

Output:
```
{
  "name":     {
    "first": "Tom",
    "last":    "Anderson"
  },
  "age": 37,
  "children": ["Sara", "Alex", "Jack"],
  "fav.movie": "Deer Hunter"
}
```

Import:

JSON [id] :RO fname	- Opens a JSON file for reading. The id is optional and defaults to "0" if only one file is open at a time. If two or more are open, the id must be supplied ("0"-"99")
value = JSON [id] :GV key	- Get a value from a JSON file using a path to a key.

Keys are a way to reference part of a JSON document using dot syntax. An example of dot syntax would be a key, such as "name.first" or "age". There are reserved symbols used in key syntax that can be used to retrieve certain values from the JSON:

'#' is used to get the number of elements inside of an object or array.
'@' is used to specify a literal, or if at the end of the path, get the name of the current object.

Index positions can also be used to reference specific elements by numeric position inside of an object or an array. Indexes in Key Syntax start at position 1.

x = JSON :GV "fruits.10" will attempt to find the tenth (10) item inside a fruits object or array.

x = JSON :GV "fruits.@10" will attempt to find a key named "10" inside a fruits object and return its value.

Example:
Given the following JSON, here are example commands and what they return.
```
{
  "name":     {
    "first": "Tom",
    "last":    "Anderson"
  },
  "age": 37,
  "children": ["Sara", "Alex", "Jack"],
  "fav.movie": "Deer Hunter"
}
```

```
Then: JSON :RO "/tmp/myfile.json" ' open the JSON file for reading
Then: x=JSON :GV "name.first"     ' x contains "Tom"
Then: x=JSON :GV "name.1.@"       ' x contains "first"
Then: x=JSON :GV "age"            ' x contains "37"
Then: x=JSON :GV "children.#"     ' x contains "3"
Then: x=JSON :GV "children.1"     ' x contains "Sara"
Then: x=JSON :GV "fav\.movie"     ' x contains "Deer Hunter"
Then: JSON :CL                    ' close the JSON file
```

Fill-In-The-Blank PDFs

filePro now has the ability to place fill-in-the-blank PDF objects on output formats and also retrieve values from PDF documents that have fill-in-the-blank fields to be used in Processing.

There are four types of PDF Form Objects that can be used:
- **Textbox**
- **Dropdown**
- **Checkbox**
- **Radio**

When a PDF output is generated, placed objects will be interactive in any supporting PDF viewer/editor. These PDF files can be saved after filling in fields, and processing can be written to retrieve values from these fields.

Note: Using the new generation features in a report can lead to unintended results. Fields are shared across records and pages. Updating one field updates all matching instances of that field throughout the document. It is recommended to use output forms over output report

Please See Fill In PDFs in the manual for more information on document creation.

Manual Link

If the PDF was created with filePro, field names will be either the real-field or dummy field used to create the PDF object.
    e.g. "1", "42", "aa", "ab".

Use these commands to read filled-in PDF documents:

handle = PDF_OPEN(pdf_path)
    Returns a handle value (10,.0) that points to a PDF document with pdf_path as the filename. Returns a negative value on error.

error_value = PDF_CLOSE(handle)
    Frees all values and memory associated with a PDF handle and closes the document. Returns a non-zero number on error.

num_fields = PDF_GETNUMFIELDS(handle)
    Returns the number of fields in the PDF document.

name = PDF_GETFIELDNAME(handle, index)
    Returns the full name of a field in a PDF document, given its index. The index is a number between "1" and the num_fields value returned by **PDF_GETNUMFIELDS.**

type = PDF_FIELDTYPE(handle, fieldname)
    Returns the field type name of the specified field fieldname, which is one of:
- **NONE**
- **BUTTON**
- **RADIO**
- **CHECKBOX**
- **TEXT**
- **RICHTEXT**
- **CHOICE**
- **UNKNOWN**

type = PDF_FIELDTYPE2(handle, index)
    Returns the field type name of the specified field index, which is one
    of:
- **NONE**
- **BUTTON**
- **RADIO**
- **CHECKBOX**
- **TEXT**
- **RICHTEXT**
- **CHOICE**
- **UNKNOWN**

    The index is a number between "1" and the num_fields value returned by
    PDF_GETNUMFIELDS.

value = PDF_GETVALUE(handle, fieldname [, richtext])
    Returns the field value, e.g. the text in the field, checkbox status,
    combo box index, etc. for the given field name fieldname. Optionally,
    richtext can be set to "1" to return rich text data if it exists.

value = PDF_GETVALUE2(handle, index [, richtext])
    Returns the field value, e.g. the text in the field, checkbox status,
    combo box index, etc. for the given field index index. Optionally,
    richtext can be set to "1" to return rich text data if it exists. The
    index is a number between "1" and the num_fields value returned by
    PDF_GETNUMFIELDS.

QRCODE Command
ret = QRCODE(str, dest [, size [, logo [, fg [, bg]]]])
Create a QR Code from a text string.

str is the text to store in the QR code.

dest is the full name and path to the QR code to be generated.

size is the size of the QR code to be generated in pixels. Must be large
enough to store the full QR code.

logo is an optional logo to place in the center of the QR code.

fg is the foreground color of the QR code in hexadecimal.

bg is the background color of the QR code in hexadecimal.

Returns the size of the generated QR code, or -1 on error.

Example:
    Then: ret=QRCODE("fptech.com","/tmp/website.png")

```
<QRCODE TEXT="qr text" [SIZE="size"] [COLOR="color"] [FILL="bg color"]
    [X="x-pos"] [Y="y-pos"]>
```

Adds a QR code with the specified text to the PDF document.

All attributes, except for **"TEXT"**, are optional.

TEXT is the text to add to the QR code when generating the image.

SIZE is the width and height of the QR code, must be large enough to fit the entire generated image.

COLOR is the foreground color of the QR code (in hexadecimal).

FILL is the background color of the QR code (in hexadecimal).

X X position. (Default: current X position.)

Y Y position. (Default: current Y position.)

FPML Print Code Enhancements
FPML print codes can now use field names for any attribute.

Any attribute inside of an FPML print code can now reference a real field or variable inside of processing. Use "@" to reference a field.

e.g.
```
<IMAGE FILE="@1">            ' reference a real field
<IMAGE FILE="@im">           ' reference a dummy field
<IMAGE FILE="@image_path">   ' reference a long name variable
```

Note: Print codes can also be stored in a print code table and do not need to be placed directly on the output to work.

Array Commands and Enhancements
Added initial support for multi-dimensional arrays.

DIM array[n1,n2,...,n8](l,e)
Multi-Dimensional array of fields with length "l" & edit "e". Array edit is optional.

Example:
```
    dim array(2,2)
    array["1","1"]="John"
    array["1","2"]="Smith"
    array["2","1"]="Sarah"
    array["2","2"]="Jane"
```

Existing array functions can also use multi-dimensional arrays by referencing one of an array's sub arrays.

Example:
```
    CLEAR array["1"]
```

```
subscript = INDEXOF(array, value)
    Find the subscript of some value in an array.

    Example:
        array["1"]="cat"
        array["2"]="dog"
        array["3"]="bird"

        subscript = INDEXOF(array, "dog") ' subscript will contain "2"

value = A_MAX(array [, array2 [, array3 [, ... [, arrayN]]]])
    Find the maximum value between the passed in arrays.

    Example:
        array1["1"]="5"
        array1["2"]="7"
        array2["1"]="30"
        value = A_MAX(array1, array2) ' value will contain "30"

    Note: This method supports multi-dimensional arrays.


value = A_MIN(array [, array2 [, array3 [, ... [, arrayN]]]])
    Find the minimum value between the passed in arrays.

    Example:
        array1["1"]="5"
        array1["2"]="7"
        array2["1"]="30"
        value = A_MIN(array1, array2) ' value will contain "5"

    Note: This method supports multi-dimensional arrays.


value = A_TOT(array [, array2 [, array3 [, ... [, arrayN]]]])>
    Total all of the values in the passed in arrays.

    Example:
        array1["1"]="5"
        array1["2"]="7"
        array2["1"]="30"
        value = A_TOT(array1, array2) ' value will contain "42"

    Note: This method supports multi-dimensional arrays.


value = A_AVG(array [, array2 [, array3 [, ... [, arrayN]]]])
    Find the avereage of all of the values in the passed in arrays.

    Example:
        array1["1"]="5"
        array1["2"]="7"
        array2["1"]="30"
        value = A_AVG(array1, array2) ' value will contain "14"

    Note: This method supports multi-dimensional arrays.
```

DECLARE Enhancement
    Added the ability to assign directly to a longvar when creating it.

    Example:
```
        DECLARE my_var(32,*)="Hello, World!"
```

Runtime Engine

Reworked tokenization engine to no longer require setting PFTOKSIZE or related variables. Variable will now be silently ignored.

Define Processing

Added a new F5 shortcut in Define Processing for calls. F5 will now open a call for editing, or, will prompt you to create the call if it does not exist.

Debugging

New stacktrace option.

Added a new option **T** to the debugger to display a stacktrace.