

Welcome

Welcome to the filePro Full Reference Manual. This resource has been developed to satisfy a wide range of audiences ranging from new filePro users to our loyal filePro programming experts. **The How Do I ?** and **filePro Menu Options** topics serve as guides to get you started if you are a beginner and contains links to **Advanced Concepts** and **Expert** topics. By focusing on practical usage of the language, these guides are intended to familiarize you with filePro at each level. It is up to you to learn more of the syntax later as you find the need for it. By merely being creative and trying out your ideas with these guides, you will undoubtedly propel yourself into areas that will force you to learn more. It is at these points that you should stop and more thoroughly investigate the reference topics and other sections of the fPmanual.

Disclaimer

This documentation is distributed by fP Technologies Inc of Ohio for your use with licensed copies of filePro and may not be distributed except as covered by the filePro license agreements. In using this documentation, you assume all risks arising from the use of this documentation. fP Technologies Inc or its suppliers are not liable for any damages (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or other loss) arising out of the use of or the inability to use the documentation.

The How Do I ? topic guides you through the design of a typical relational database application. This section is highly recommended if you are new to filePro or if you have a need to refresh your memory on some advanced concepts. Depending on your level of filePro programming experience, you may want to skip some of the sub-topics, but at least give each topic a cursory review to get an idea of what information is presented. The topic order is meant to indicate a structured path of increasing familiarity with the program and not meant to categorize your ability or that of the package at any level of understanding. The most expert filePro programmers may not use or even know about some simple technique that a beginner uses quite productively.

Getting from point A to point B - There are as many different ways to solving a business problem as there are programmers. The wonderful thing about filePro is that it can always start as a blank screen, a blank output format, and a blank processing table. There are some standards, conventions, and ways of thinking that will help you program more quickly and more universally.

The **filePro Menu Items** topic contains a section for each of the filePro menu items. If you are new to filePro, review the beginning of each section within this topic to become acquainted with the filePro environment and each of the filePro menu options. At the beginning of each section, you will see links to **Advanced Concepts**, and **Expert** sections within the section. You can click on these links to quickly move to the more advanced areas within each section.

The **fileProODBC** topic provides information relative to using ODBC related commands and functions. The feature was separately licensed up until version 5.7 when it was added to the base product as a standard feature.

The **Glossary** topic provides common computer and filePro terminology.

You will probably use the **References** topic more than any other once you start designing in filePro. It provides references to all filePro commands, system maintained fields, environment variables, etc. with code examples.

The **Revision History** topic has been added to provide a complete historical record of things that have been changed in filePro. If you are not running a current release, make sure to check this topic before reporting bugs since you may find that your bug has already been addressed in a later release.

The **Technical Notes** topic provides information that is important to filePro users depending on platform or the specific version you are using.

This **Terminal Guide** contains key usage for common terminal types, key label codes and other information related to filePro's termcap for the Unix based operating systems.

The **Utilities** section describes commonly used filePro Utilities.

Acknowledgements

For the most part, these writings and guides are new. However, some pieces in this work have been gathered from issues of Smalltalk, written by Ken Brody. Other pieces were taken from issues of **The Guru**, or **The filePro CookBook**, written by John Esak. Most of the documentation associated with the HTML topics tools were written by Lee Machan.

The article in the reference **filePro and Laserjet Printing** was written by Jim Asman.

A special thank you to John Esak and Ray Hoover for the many hours of contributions they have made through the years.

fP Technologies Inc. of Ohio

Revision Date: October 2020

BUG Reporting

Writing clearly is essential in a bug report. If the technical support person can't tell what you meant, you might as well not have said anything.


- Be specific. If you can do the same thing two different ways, state which one you used. "I selected Load" might mean "I clicked on Load" or "I pressed Alt-L". Say which you did. Sometimes it matters.
- Be verbose. Give more information rather than less. If you say too much, the programmer can ignore some of it. If you say too little, they have to come back and ask more questions. One bug report received was a single sentence; every time the customer had to be asked for more information, the reporter would reply with another single sentence. It took several weeks to get a useful amount of information, because it turned up one short sentence at a time.
- Be careful of pronouns. Don't use words like "it", or references like "the window", when it's unclear what they mean. Consider this: "I started FooApp. It put up a warning window. I tried to close it and it crashed." It isn't clear what the user tried to close. Did they try to close the warning window, or the whole of FooApp? It makes a difference. Instead, you could say "I started FooApp, which put up a warning window. I tried to close the warning window, and FooApp crashed." This is longer and more repetitive, but also clearer and less easy to misunderstand.
- Read what you wrote. Read the report back to yourself, and see if you think it's clear. If you have listed a sequence of actions that should produce the failure, try following them yourself, to see if you missed a step.

Summary

- The first aim of a bug report is to let the support person see the failure with their own eyes. If you can't be with them to make it fail in front of them, give them detailed instructions so that they can make it fail for themselves.
- In case the first aim doesn't succeed, and the programmer can't see it failing themselves, the second aim of a bug report is to describe what went wrong. Describe everything in detail. State what you saw, and also state what you expected to see. Write down the error messages, especially if they have numbers in those messages.
- When your computer does something unexpected, freeze. Do nothing until you're calm, and don't do anything that you think might be dangerous.
- By all means try to diagnose the fault yourself if you think you can, but if you do, you should still report the symptoms as well.
- Be ready to provide extra information if the programmer needs it. If they didn't need it, they wouldn't be asking for it. They aren't being deliberately awkward. Have version numbers at your fingertips, because they will probably be needed.
- Write clearly. Say what you mean, and make sure it can't be misinterpreted.
- Above all, be precise. Programmers like precision.

Using Help

There are many hyperlinks in the filePro Full Reference Manual. They will stay on your screen until you press ESCAPE, press ENTER, or click the left mouse button. There are also many hyperlink jumps . These are secondary HELP screens that will appear on top of your primary window (depends on the Help version you are using). They have scroll bars and look very much like your primary HELP window. Read them and then close the jump window as you normally would close any secondary window. Do not lose your place on the primary topic by maximizing one of these secondary windows. It is also important to understand that these jumps are usually extra reference material and may be much more advanced in content than what you are studying. If this is true, just close the window and keep reading where you are. You can always find these resources later.

To go to the next sub-topic within a topic, use the >> button in the Help Menu bar. To go to previous topics, use the << button. If one of the buttons is not lit up, you are at one end or the other within that topic and should probably press Help Topics to choose the next (or previous)  topic.

Viewing Problems:

The text quality and size is dependent on your system video configuration and various other factors such as viewer/browser preferences. For the Windows version, you can change the font size by selecting "Options" and then "Font" to change to a larger or smaller font. For the HTML versions, check your viewer/browser settings and adjust settings to see what gives you the best results.

Plain **HTML versions of fPManual** - This version uses a style sheet "styles.css" and is the primary manual release.

filePro Plus Capacities

Number of Files	limited only by disk space
Number of Records per file	1 Billion
Number of Fields per Record	999
Number of Characters per Record	65,400
Number of Reports Forms, Labels	limited only by disk space
Length of Output Formats	255 lines per record
Width of Output Formats	255 characters
Number of Report Levels	9 levels: 8 subtotal and 1 grand total
Number of Labels Across Page	1 to 9
Number of Screens	limited only by disk space
Number of Fields per Screen	200
Number of Print Codes for Each Printer	9999
Number of Processing Elements per Table	9999
Number of Automatic Indexes	26
Number of Demand Indexes	10
Dummy Fields	32767
Number of Fields to Sort On Per Report /Index	8
Number of Criteria for Record Selection	unlimited
Associated Field Instances	32
Processing Line Length	120 characters
Selection Sets Called in Processing	unlimited
Token Table Size	Available Memory
Number of Global Edits	100
Number of File-specific edits	100
Number of Built-in Field Edits	38
Mathematical Precision	16 Places to left. 8 places to right of decimal point.
Number of User-defined Menus	Limited only by disk space
Number of Choices on Each Menu	24
Maximum line length within a help file.	Version 5.0 - Increased from 132 to 512 characters.

Disclaimer

This documentation is distributed by fP Technologies Inc for your use with licensed copies of filePro and may not be distributed except as covered by the filePro license agreements. In using this documentation, you assume all risks arising from the use of this documentation. fP Technologies Inc or its suppliers are not liable for any damages (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or other loss) arising out of the use of or inability to use the documentation.

Disclaimer Encryption

IMPORTANT: If the key is lost the data is lost - there is no way fP or anyone else can recover the data. fP therefore disclaims any responsibility for loss of data that results from use of the encryption routines.

Caution: Be careful when integrating encryption for stored data fields with edits and for fields that you have a need to scan for values or using indexes.

Carefully think through the process of encrypting your data before applying encryption to your stored data fields since you can destroy your data or at least make it difficult to properly access your data.

- Don't apply edits other than "*" to stored data fields you encrypt.
- Avoid encrypting fields you need to build indexes on.
- Always make a backup before applying encryption.

System Requirements

filePro

Windows Network

Windows XP or later.

RAM: 48 MB

Disk Space: 70 MB

If you are running Windows XP or later, the minimum RAM requirements for Windows will usually allow you to run filePro applications satisfactorily. Additional RAM may be required for Peer-to-Peer Network Servers to achieve acceptable performance for filePro if the machine designated as the filePro file Server is also used as a workstation and/or provides other services such as printing, FAX services, etc.

LINUX/UNIX/UNIXWARE

RAM: 64 MB

Disk Space: 50 MB

Note: RAM requirements will depend on how many simultaneous connections are expected, other server tasks and the demands placed on the server for your application. If you experience performance problems, additional memory may help.

GI Server

RAM: 96 MB

Disk Space: 10 MB

Note: RAM requirements will depend on how many simultaneous connections are expected, other server tasks and the demands placed on the server for your application. If you experience performance problems, additional memory may help.

fileProGI Client

Windows XP or later

RAM: 64MB

Video: SVGA capable of at least 800x600 pixels.

Disk Space: 50 MB

fPSQL

This is a separate purchase which allows the user to use standard SQL query language to search and report filePro data. fPSQL comes with its own PDF manual.

This will open your browser to download or display the PDF manual

[Click Here](#)

Standard filePro Terms

Databases, Fields, Records, Files

An example of a simple database is an address book. In this book, it is possible to keep track of contacts, usually arranged by last name. The information typically collected in an address book includes first and last names, home and work phone numbers, street address, city, state, zip code, email address, and possibly personal notes. filePro is a program that helps users to organize and use this type of information. The usage can range from a simple mailing list (which can be organized and sorted at incredible speeds) all the way to a complex registration program, which keeps track of students, teachers, fees, courses, and materials.

Within filepro, each type of information stored, such as city or a phone number, is called a field. Each full collection of fields is called a record. In the address book, a completed entry may include the persons name, contact information and personal notes. The specific information in a field is called data and is saved when each record is saved. The records are collected to create a database file.

filePro uses what is called a map to define the fields. This map is created by the Define Files option of the filePro Plus Main Menu.

Within the map, fields are referred to as "real fields", and are defined by:

Number (and name, although name is optional)
Length : number of characters (up to 999)
Type of data: alphanumeric, numeric, date, time

There can be up to 999 fields per record.

You can build many different named filePro databases (limited only by disk space), and they can all share and use data from each other.

The filePro programs are reached through the filePro Plus Main Menu.

§ To open the filePro Plus Main Menu in UNIX/LINUX/AIX, simply press p at any shell prompt (since the filePro Plus program 'p' is found in the /usr/bin directory and this directory is listed in the PATH variable for all users under these systems).

Note: The path environment variables must be set in a file fppath, fpplus.bat or other batch file in order to properly run filePro. Refer to the [path](#) variables for more information.

§ To open the filePro Plus Main Menu in Windows, click on the ICON created by the install shield. This will call a batch file e.g. fpplus.bat or fpws.bat which sets the filePro environment and executes "p". If the filePro Plus Main Menu is not properly displayed, check the last line of the batch file to see if it has been modified to call a user menu name. If so, create a copy of the batch file and modify the last line to drop the user menu after \p.exe.

You can create and run filePro files from this filePro Plus Main Menu. It is divided into two sides: Creation Programs (left side) and Runtime Programs (right side).

In most instances, end users will be provided with custom "user" menus. For our purposes, we will use the filePro Plus Main Menu. This menu is used to develop the applications. It is divided into two sections. You will use this menu to do virtually all design work within filePro. It is structured in this way so that you can also use it to test your work before packaging it up for the end - users. (Some filePro users do use this menu to access their programs; however, there are preferred ways of accessing user programs and pre-built applications. You will learn about them later.)

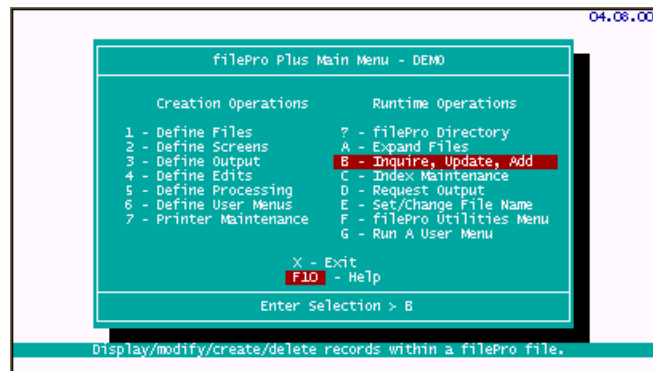
This primer will give you an overview of how to use the **Inquire, Update & Add (IUA)** menu option. This is provided to familiarize you with the "look and feel" of filePro. After we have something to work with, the "Beginner" guide will go through things in more detail.

By the way, case (upper or lower) is not important to Windows-based versions of filePro, but VERY important to Linux/Unix based systems. Throughout these guides, lower case will be used in most instances, and occasionally filePro will turn them into UPPER case as needed. You may use whichever suits you best, but you must be consistent. If you name files in lowercase letters, you must refer to them in lowercase as well.

All menu choices throughout filePro are "hot-keyed". This means you can just press the letter or number of the desired choice. Alternatively, you can move the highlighted bar to the desired choice and then press ENTER. Move the highlighted bar with the Arrow keys. The SPACE bar acts just like the Down Arrow.

The filePro Plus Main Menu

Inquire, Update, Add (IUA)



NOTE: Depending on your version of filePro, the menu examples may not look exactly the same as those pictured.

From the filePro Plus Main Menu, Select **B**Inquire, Update, Add.

filePro will then prompt you for a filename. In our example, we will be using the "rolodex" file.

NOTE: You may see a different set of filenames on your screen, or only the filename "rolodex". It is unimportant how many files exist when you are choosing a particular file; you need only to choose the one you will be working with.



You can choose a file in a variety of ways. You may use the arrow keys to move the highlighted bar to the desired selection and press **ENTER**. Or, you can type in a part of the desired choice until the cursor "automatically" moves to it, then press **ENTER**. Choose rolodex.

At this point, filePro will ask you which screen number you wish to access.



For "Enter Screen Number", Select **1**.

The following is the primary menu for locating and updating filePro files. It is called the INQUIRE, UPDATE, ADD menu.



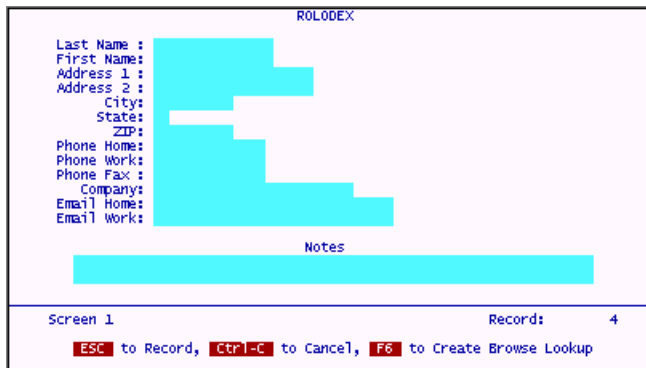
Select **3** to add records.

We will enter several records. When you are done with each full screen, **press ESC** to save your work. filePro will automatically present you with a new blank record. This is because you chose 3 (Add Records Mode). filePro thinks you want to keep adding records until you tell it that you want to stop. You do this by pressing **BREAK** instead of ESC.

§ The BREAK key on Windows systems is Control-C. (Hold down Ctrl and while holding it down, press C).

§ On a Unix system, the BREAK key is usually listed as DELETE, and the keyboard key so designated performs the BREAK function filePro needs. The BREAK key on Linux usually defaults to Ctrl-C the same as Windows.

It is important that you do not prematurely press BREAK since your records will not be saved until pressing ESC. Do NOT confuse the DELETE key with "deleting" records from your file. These are two completely different functions.



When you add the following sample records, your entries may go onto different record numbers than those shown in the pictures. This might happen if you press ESC too many times, or make a mistake and start over. Not to worry, as this will not affect your work. (Record number is an internally assigned number that filePro uses in certain instances. Users do not often need to use this number.)

Enter the following data, using the enter key to change between fields.

Robin Hakan
Penthouse
777 Lucky Way
Penthouse, NJ 07436-0777
Home: (201) 422-3333
Work: (201) 422-4444
Fax: (201) 422-5555
Company: The Valor Group
Email home: rhakan@home.com
Email work: rhakan@work.com
Notes: Good looking, filePro programmer likes brownies

If you need to return to a field, use the arrow keys. When done, it should resemble this:

ROLODEX

Last Name : rakan
 First Name : Robin
 Address 1 : 777 Lucky Way
 Address 2 : Penthouse
 City: Penthouse
 State: NJ
 ZIP: 07436-0777
 Phone Home: (201) 422-3333
 Phone Work: (201) 422-4444
 Phone Fax: (201) 422-5555
 Company: THE VALOR GROUP
 Email Home: rhakan@home.com
 Email Work: rhakan@work.com

Notes

Brilliant, Good Looking, Filepro Programmer
 Likes Brownies

Screen 1 Record: 1

ESC to Record, **Ctrl-C** to Cancel, **F6** to Create Browse Lookup

Press **ESC** to record, and continue to add the next 2 records.

Remember, when you press **ESC** after entering your data, filePro will automatically bring you to the next blank record. This is normal. You did not lose anything. filePro merely stored the previous record and is presenting you with a chance to add another record.

Fill in the next few records, pressing **ESC** when you complete each.

Mark Farmer
200 Willow Drive
Paterson NJ 07555
Home: 427-2928
Notes: carpenter, has truck, fathers name is Al

Sally Smith
44 Jones Blvd
Washington DC 22722-4444
Home: (800) 444 4444
Work: (800) 444 1234
Company: Rand Corporation
Email Home: sally@aol.com
Email Work: sallysmith@work.com
Notes: WIN95 wizard, filePro expert, likes brownies

After you fill in the final record, press **ESC**.

You should now be on a blank record.

ROLODEX

Last Name :
 First Name :
 Address 1 :
 Address 2 :
 City:
 State:
 ZIP:
 Phone Home:
 Phone Work:
 Phone Fax :
 Company:
 Email Home:
 Email Work:

Notes

Screen 1 Record: 4

ESC to Record, **Ctrl-C** to Cancel, **F6** to Create Browse Lookup

Press **CTRL-C (BREAK)** to exit from Add Records Mode.

The prompts at the bottom of the screen will change to look like the following screen.

ROLODEX

Last Name :
 First Name :
 Address 1 :
 Address 2 :
 City:
 State:
 ZIP:
 Phone Home:
 Phone Work:
 Phone Fax :
 Company:
 Email Home:
 Email Work:

Notes

Screen 1 Record: 3

Enter Selection > █

D-Delete, **H**-Hardcopy, **U**-Update, **X**-Exit, **F**-Print Form, **B**-Browse

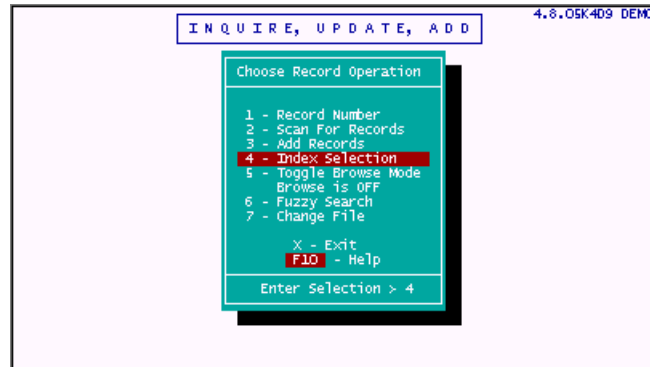
You can **press X** to return to the IUA, or "clerk", menu. From there we can look for the records we just added.

Index Selection

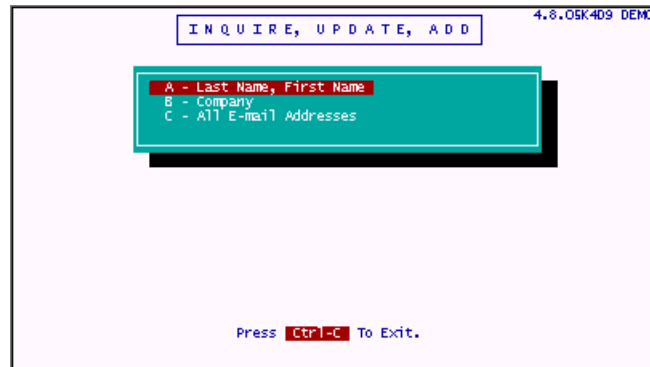
Often, when working with a database, you need to find a record quickly. Just as we keep our address books in alphabetical order, so that we can quickly look up an individual, if we have filePro sort the records within a file, it can help us to find the needed file almost instantaneously. Moreover, filePro can sort all the records based on any field not just names.

In many cases, you will not have to enter the complete data you are searching for with an index. Often, just filling in a few starting characters will get you very close to the desired record. Keep in mind that this will not work with certain types of data, like dates. Dates must be entered exactly and there must be an exact match in the file for the search to be successful and land you on a record.

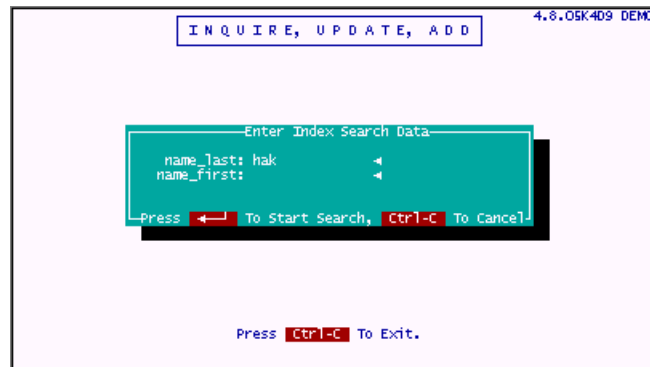
From the IUA menu, **Select 4 - Index Selection**



Select "A - Last Name, First Name".



Enter the letters **Hak** in the name_last field, and press **ENTER** twice (Do not put anything in name_first). Pressing the **F7** key in any blank field will take you to the last record of the index.



filePro finds the record immediately, and you are now also inside the file in Index Mode. This means you are viewing the first file that filePro found that your search. At this point, you can use the arrow keys to move up and down through the records. You will notice that they are in alphabetical order.

To see the alphabetical nature of an index graphically, while standing on the Hakan record you just found, press the **B** key to "browse" the other records in the file. You will see the "Hakan" record at the top of the screen and many records following your selection of "Hak".

Last Name	First Name	Company
Hakan	Robin	THE VALOR GROUP
Hall	William	Halls' Flowerworld
Hoben	Gerald	Ceco Associates Inc.
Icker	Kevin	The Ice Cream Cottage
Loftus	Joseph	Continental Transmissions
Lowry	Kate	Comics On The Green
Malcolm	Jason	Forum Plaza Nautilus
Mart	Jason	Uniserve Insurance Agency
O'Dea	Thomas	O'Dea Construction Co.
O'Malley	Ann	O'Malley & Langan, PC
Ploskona	Richard	NEPA's Auto Sales
Rogo	John	John Rogo CPA
Rogo	Kate	
Romar	June	Romar Dance Studio
Rose	Mary L.	Roof Pro Inc.
Scholes	Deb	Merry Maids
Slick	Bernadette	Cozmic Production Company
Smith	Sally	Rand Corporation

Index: **ARROWS** -Change Record, **←** -Select Record, **U** -Update Record
S -Set Selection, **Z** -fuzzy search, **R** -Reset **C** -Continuous On
F -Format, **H** -Hardcopy, **X** -Exit

Press either the **Page Up** key or the up arrow at this point to see records that come before "Hak" (alphabetically).

Last Name	First Name	Company
***** BEGINNING OF FILE *****		
The Hillier Group		
Claire's Boutique		
AMC Credit Corp		
Anytime Lock & Key Inc		
Eddie's Tire Service		
Farmer	Mark	
Hakan	Robin	THE VALOR GROUP
Hall	William	Halls' Flowerworld
Hoben	Gerald	Ceco Associates Inc.
Icker	Kevin	The Ice Cream Cottage
Loftus	Joseph	Continental Transmissions
Lowry	Kate	Comics On The Green
Malcolm	Jason	Forum Plaza Nautilus
Mart	Jason	Uniserve Insurance Agency
O'Dea	Thomas	O'Dea Construction Co.

Index: **ARROWS** -Change Record, **←** -Select Record, **U** -Update Record
S -Set Selection, **Z** -fuzzy search, **R** -Reset **C** -Continuous On
F -Format, **H** -Hardcopy, **X** -Exit

This functionality is maintained throughout all indexes. Those indexes built on character type data, like names and companies, will always organize the file alphabetically. Indexes built on numerical data will organize the file in ascending order from lowest to highest numbers. Indexes built on date data will organize the file chronologically from lowest to highest dates.

Try the Company index, by **pressing X** to get back to the "clerk" menu. Then select 4 (Index Selection) and then:

Select **B - Company**.

4.8.05K409 DEMO

INQUIRE, UPDATE, ADD

A - Last Name, First Name
B - Company
C - All E-mail Addresses

Press **Ctrl-C** To Exit.

Try just one letter. **How about r?**

4.8.05K409 DEMO

INQUIRE, UPDATE, ADD

Enter Index Search Data

Company: r

Press **←** To Start Search, **Ctrl-C** To Cancel

Press **Ctrl-C** To Exit.

You can see that you have come to the first "r" record in the file. This time the file is being searched by the Company index, not the Last Name index.

```

ROLODEX

Last Name : Smith
First Name : Sally
Address 1 : 44 Jones Blvd.
Address 2 :
City: Washington
State: DC
ZIP: 22722-4444
Phone Home: (800) 444-4444
Phone Work: (800) 444-1234
Phone Fax :
Company: Rand Corporation
Email Home: sally@aol.com
Email Work: sallysmith@work.com

Notes
WDN95 wizard, filePro expert
Likes brownies

Screen 1          Enter Selection >          Record: 2
Index Mode
D -Delete, H -Hardcopy, U -Update, X -Exit, F -Print Form, B -Browse

```

In the screen above, note the label Index Mode. If this label is not desired when in Index Selection mode, use the -DM to suppress this label. **Version 6.0.02**

If you **press B** for Browse at this point, you will see that you are in the file alphabetically by Company. Note that the record with NO company comes before the "r" in Rand Corporation. Blanks come before any characters in an index. In fact, to get to the first record in any index, enter a blank as the search criteria. This guarantees that you will see the first record in the index. If there are no blank records, you will see the first non-blank record, and progress from there .

Last Name	First Name	Company
Smith	Sally	Rand Corporation
Romar	June	Romar Dance Studio
Rose	Mary L.	Roof Pro Inc.
		The Hillier Group
Ticker	Kevin	The Ice Cream Cottage
Hakan	Robin	THE VALOR GROUP
Mart	Jason	Uniserve Insurance Agency
***** END OF FILE *****		

Index:

ARROWS -Change Record, ← -Select Record, U -Update Record

S -Set Selection, Z -Fuzzy search, R -Reset C -Continuous On

F -Format, H -Hardcopy, X -Exit

Defining Browse Formats

Often times, when looking in the browse format, the information you need is not shown. For example, currently, our rolodex file does not show us the individuals phone numbers. While on a browse view of records, you can change the "format" of what you are seeing in order to view any fields contained in each record. To change the browse fields, use the **(F)** ommat key.

While in browse mode, **Press F.**

Last Name	First Name	Company
1- name_last	6- state	11- company_name
2- name_first	7- zip	12- n)Notes
3- a0)address_1	8- p)phone_home	13- n)Notes
4- a0)address_2	9- p)phone_business	14- e)email-personal
5- city	10- p)phone_fax	15- e)email-business

Browse Header:
Last Name First Name Company

Browse Format:
*1 *2 *11

Enter Selection >
U-Update, **C**-Clear, **S**-Save, **L**-Load, **X**-Exit

The prompts change, and you can now adjust the format by **(U)** pdating it.

Press U.

You are now in update mode, and can move throughout the screen using the arrow keys or the Tab key. filePro update screens all function in ovrtype mode, meaning that if you try to add a space in front of a word by pressing the space bar, you will not insert a space, but instead wipe out what information was in the space in front of the cursor. You can use the insert key to add spaces, and the delete key to remove them.

Last Name	First Name	Company
1- name_last	6- state	11- company_name
2- name_first	7- zip	12- n)Notes
3- a0)address_1	8- p)phone_home	13- n)Notes
4- a0)address_2	9- p)phone_business	14- e)email-personal
5- city	10- p)phone_fax	15- e)email-business

Browse Header:
Last Name First Name Company

Browse Format:
*1 *2 *11

Press **ESC** To Record Format
Press **F5** To Toggle Between Field Headings And Lengths/Edits

Add the field for phone number in the same manner the other fields were done. Use * and the field number you want to see on the browse screen. In this case, we will use field 9 (phone_business)

Last Name	First Name	Company	Business Phone
1- name_last	6- state	11- company_name	
2- name_first	7- zip	12- n)Notes	
3- a0)address_1	8- p)phone_home	13- n)Notes	
4- a0)address_2	9- p)phone_business	14- e)email-personal	
5- city	10- p)phone_fax	15- e)email-business	

Browse Header:
Last Name First Name Company

Browse Format:
*1 *2 *11 *9

Press **ESC** To Record Format
Press **F5** To Toggle Between Field Headings And Lengths/Edits

To see the fields the new format will show, **press ESC** to save your work, and then **press X** to redisplay the records. The phone number is now showing for each record.

Last Name	First Name	Company	Business Phone
Hall	William	Halls' Flowerworld	(123) 133-1334
Hoben	Gerald	Ceco Associates Inc.	
Icker	Kevin	The Ice Cream Cottage	
Loftus	Joseph	Continental Transmission	
Lowry	Kate	Comics On The Green	
Malcolm	Jason	Forum Plaza Nautilus	
Mart	Jason	Uniserve Insurance Agenc	(225) 698-7125
O'Dea	Thomas	O'Dea Construction Co.	
O'Malley	Ann	O'Malley & Langan, PC	(485) 696-7777
Ploskona	Richard	NEPA's Auto Sales	
Rogo	John	John Rogo CPA	
Rogo	Kate		
Romar	June	Romar Dance Studio	
Rose	Mary L.	Roof Pro, Inc.	
Scholes	Deb	Merry Maids	
Slick	Bernadette	Cozmic Production Compan	
Smith	Sally	Rand Corporation	(800) 444-1234

***** END OF FILE *****

Index:
ARROWS -Change Record, **←** -Select Record, **U** -Update Record
S -Set Selection, **Z** -FuZzy search, **R** -Reset **C** -Continuous on
F -Format, **H** -Hardcopy, **X** -Exit

Go back to the format screen by **pressing F**.

You can also add a "heading" for the fields displayed.

Do this by **pressing U** and adding the following. While you are at it, you can make browses easier to read by pushing certain fields together (like first and last names). To do this, use the < operator instead of the * operator on the desired fields. This pushes the specified field one space away from the field to its left. (The < is called the "push left" operator.)

Enter the following and **press ESC** to save the screen.

Last Name	First Name	Company	Business Phone
1- name_last	6- state	11- company_name	
2- name_first	7- zip	12- n)Notes	
3- a0)address_1	8- p)phone_home	13- n)Notes	
4- a0)address_2	9- p)phone_business	14- e)email-personal	
5- city	10- p)phone_fax	15- e)email-business	
Browse Header:			
Contact Name		Company	Business Phone
Browse Format:			
*2	<1	*11	*9

Press **ESC** To Record Format
Press **F5** To Toggle Between Field Headings And Lengths/Edits

Press X to see the records again. Your work will look like this:

Contact Name	Company	Business Phone
William Hall	Halls' Flowerworld	(123) 123-1234
Gerald Hoben	Ceco Associates Inc.	
Kevin Icker	The Ice Cream Cottage	
Joseph Loftus	Continental Transmission	
Kate Lowry	Comics On The Green	
Jason Malcolm	Forum Plaza Nautilus	
Jason Mart	Uniserve Insurance Agenc	(225) 698-7125
Thomas O'Dea	O'Dea Construction Co.	
Ann O'Malley	O'Malley & Langan, PC	(485) 696-7777
Richard Ploskona	NEPA's Auto Sales	
John Rogo	John Rogo CPA	
Kate Rogo		
June Romar	Romar Dance Studio	
Mary L. Rose	Roof Pro Inc.	
Deb Scholes	Merry Maids	
Bernadette Slick	Cozmic Production Compan	
Sally Smith	Rand Corporation	(800) 444-1234
**** END OF FILE ****		

Index:
ARROWS -Change Record, **←** -Select Record, **U** -Update Record
S -Set Selection, **Z** -Fuzzy search, **R** -Reset **C** -Continuous on
F -Format, **H** -Hardcopy, **X** -Exit

If this is a browse format you will be using often, you can **SAVE** it for future use. You will not have to design it all over again. Do this by **pressing F** to return to the format screen.

Press S

Contact Name	Company	Business Phone
1- name_last	6- state	11- company_name
2- name_first	7- zip	12- n)Notes
3- a0)address_1	8- p)phone_home	13- n)Notes
4- a0)address_2	9- p)phone_business	14- e)email-personal
5- city	10- p)phone_fax	15- e)email-business
Browse Header:		
Contact Name		Company
Browse Format:		
*2	<1	*11
		*9

Enter Browse Format Name:
 Or Press **Ctrl-C** To Exit.

[NEW]
 default

Press **ENTER** at the [NEW] prompt, and give your browse format a name.

To retrieve a previously saved browse format, **press L** (for Load) and select the one you want. You may have an unlimited number of browse formats. (Limited by your disk space that is.)

After you save this format, if you **press L** at this point, you will only see one name displayed. This is because there have been no other browse formats for this file yet saved. If you want a particular browse format to come up as soon as you enter a file, give it the name "default". The format named "default" is the format first chosen by filePro to display when you enter IUA (Inquire, Update & Add). It will remain the browse format until you change it.

Contact Name	Company	Business Phone
1- name_last	6- state	11- company_name
2- name_first	7- zip	12- n)Notes
3- a0)address_1	8- p)phone_home	13- n)Notes
4- a0)address_2	9- p)phone_business	14- e)email-personal
5- city	10- p)phone_fax	15- e)email-business

Browse Header:

Contact Name	Company	Business Phone
*2	<1	*11
		*9

Browse Format:

default

Enter Browse Format Name:

Or Press **Ctrl-C** To Exit.

Leave the browse format section and return to the clerk menu.

NOTE: ver. 5.8.03

Password protection of .sel and .brw formats

By setting the environment variable PFBRWFORMPWD to ON, one can then select certain .brw formats and assign a password to protect against unauthorized changing and saving of the .brw format. Without knowing the password assigned to the .brw format and PFBRWFORMPWD set to OFF (default if not set) you will not be able to modify and save.

Scanning For Records

Scanning, an alternative method of finding records, matches each record to entered criteria. This method for finding records is called "scanning", because each record is tested (or scanned) to see if the values in its fields match those criteria designated by the user. The selection criteria is entered as a "query" in either short or extended format.

If a record matches your criteria, it is brought to the screen. Successive matches can be seen from that point by pressing **ENTER** to go to the next matching record, or by **pressing B** to browse the next 18 records which match your criteria (if there are that many).

Note: In version 6.0.00 additional comparisons were added for associated fields. The full list of comparisons are:

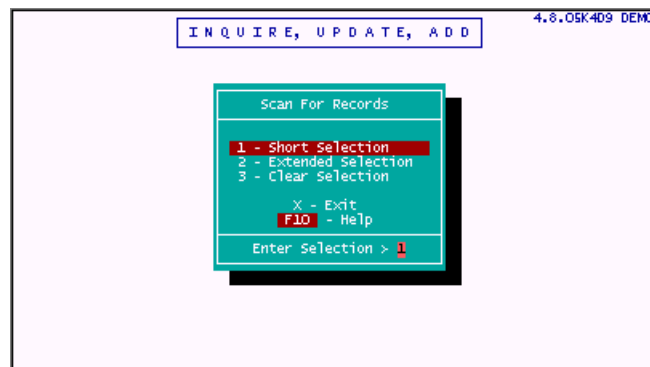
- LT - Less than
- LE - Less than or equal
- EQ - Equal to
- NE - Not equal to
- GE - Greater than or equal
- GT - Greater than
- CO - Contains
- LTF - Less than field
- LEF - Less than or equal field
- EQF - Equal to field
- NEF - Not equal to field
- GEF - Greater than or equal field
- GTF - Greater than field
- COF - Contains field
- AEQ - Associated field, all equal
- ANE - Associated field, all not equal
- ACO - Associated field, all contain

Short Selection

Select 2 - Scan For Records.



Select 1 - Short Selection.



filePro tests matching criteria using the de-facto standards, "equal to", "not equal to", "greater than or equal to", "greater than", "less than or equal to", "less than". It also has a valuable addition to these in "contains". Besides these criteria matching operators, it also adds a unique tool to speed building queries "range". All of these operators are spelled as closely as possible to what they mean, i.e., EQ means "equal to", GE means "greater than or equal to", CO means "contains" and RG means "range". The following screens show how to use these operators.

Enter the following query and press **ENTER** at the "Enter Connective (and/or)" prompt.

```

INQUIRE, UPDATE, ADD 4.8.05K409 DEMO
1- name_last      7- zip      13- n)Notes
2- name_first    8- p)phone_home 14- e)email-personal
3- a0)address_1  9- p)phone_business 15- e)email-business
4- a0)address_2 10- p)phone_fax
5- city          11- company_name
6- state         12- n)Notes

Enter Field To Select By: 5
Enter Relationship (EQ,NE,GE,GT,LE,LT,RG,CO): eq
Enter city: Pat
Enter Connective (and/or):

Press PgDn For Next, PgUp For Previous Fields
Press F5 To Toggle Between Field Headings And Lengths/Edits

```

This query will search the city field of every record for the word pat.
The first record that matches the scan criteria is displayed.

```

ROLODEX
Last Name : Farmer
First Name: Mark
Address 1 : 300 Willow Drive
Address 2 :
City: Paterson
State: NJ
ZIP: 07555
Phone Home: 427-2928
Phone Work:
Phone Fax :
Company:
Email Home:
Email Work:

Notes
carpenter, has truck
father's name is Al

Screen 1 Enter Selection > Record: 4
Delete, H-Hardcopy, U-Update, X-Exit, F-Print Form, B-Browse

```

To see all records that match your criteria, press **B** for browse.

```

Contact Name      Company      Business Phone
Mark Farmer
William Hall      Halls' Flowerworld (123) 123-1234
**** END OF FILE ****

Select:
ARROWS -Change Record, Select Record, U-Update Record
S-Set Selection, Z-fuzzy search, R-Reset C-Continuous On
F-Format, H-Hardcopy, X-Exit

```

Without moving the highlighted bar, press **ENTER** to take you back to the full screen view of the record on which you are standing (Mark Farmer). You will notice a new sign in the bottom right corner of the screen. It says, "ENTER for Next Match" (The little left-pointing arrow means ENTER). This is very important! It means that you must press ENTER, not the Down Arrow to go to the next matching record in your scan. Once you press ENTER to move to the next matching record, you can NOT press the Up Arrow to go back to the record on which you were standing. Scanning, unlike indexing, does not move the records around to organize them. It would be as if you had an address book in which you randomly wrote names and numbers. If at some point, you went through the book and highlighted all of the local phone numbers, you could flip through and find them quickly, but the records themselves would not be ordered. That is what filePro does when it performs a scan - it shows you the highlighted records in a list, but if you use the arrows to navigate, it moves you through the other records. This will be shown more clearly later in this section.

```

ROLODEX
Last Name : Farmer
First Name: Mark
Address 1 : 300 Willow Drive
Address 2 :
City: Paterson
State: NJ
ZIP: 07555
Phone Home: 427-2928
Phone Work:
Phone Fax :
Company:
Email Home:
Email Work:

Notes
carpenter, has truck
father's name is Al

Screen 1 Enter Selection > Record: 4
Delete, H-Hardcopy, U-Update, X-Exit, F-Print Form, B-Browse

```

Note the sign on the bottom right of the screen.

Press **ENTER** now, and you will be brought to the next matching record in full screen view.

```

                                ROLDEX
Last Name : Hall
First Name : William
Address 1 : 123 Lake Drive
Address 2 :
City: Paterson
State: NY
ZIP: 12345
Phone Home:
Phone Work: (123) 123-1234
Phone Fax : (321) 321-4321
Company: Halls' Flowerworld
Email Home:
Email Work:

                                Notes
best flowers

Screen 1          Enter Selection > █          Record: 5
D -Delete, H -Hardcopy, U -Update, X -Exit, F -Print Form, B -Browse

```

Press ENTER again to see the next match. Since there are no more matching records, you will see the following screen:

```

                                4.8.05K409 DEMO
                                INQUIRE, UPDATE, ADD

                                Scan For Records

                                1 - Short Selection
                                2 - Extended Selection
                                3 - Clear Selection

                                X - Exit
                                F10 - Help

                                Enter Selection > 1

```

Try another scan by pressing 1 - Short Selection.

Enter this query.

```

                                4.8.05K409 DEMO
                                INQUIRE, UPDATE, ADD

01- name_last          07- zip          13- n)Notes
02- name_first        08- p)phone_home 14- e)email-personal
03- a0)address_1     09- p)phone_business 15- e)email-business
04- a0)address_2     10- p)phone_fax
05- city             11- company_name
06- state            12- n)Notes

Enter Field To Select By: 7
Enter Relationship (EQ,NE,GE,GT,LE,LT,RG,CO): ge
Enter zip: 079

Enter Connective (and/or):

Press PgDn For Next, PgUp For Previous Fields
Press F5 To Toggle Between Field Headings And Lengths/Edits

```

The criteria GE means that records must contain data that is "greater than OR equal to" the data designated in the query. If the data in the field being used is all numbers, then this means a match on numbers that are higher than or equal to the supplied number. Even though field 7 is a ZIP code made up of 5 characters, they will all be numbers and filePro will compare the first three characters of each record's field 7 against our test criteria... in ascending order through all 5 characters of the field. In other words, 07900 and 08000 and 09999 and 99999 will all be "greater than or equal to" the "079" we have entered. Zip codes of "07899" and lower will be excluded.

Contact Name	Company	Business Phone
Sally Smith	Rand Corporation	(800) 444-1234
William Hall	Halls' Flowerworld	(123) 123-1234
Gerald Hoben	Ceco Associates Inc.	
Kevin Icker	The Ice Cream Cottage	
Joseph Loftus	Continental Transmission	
Jason Malcolm	Forum Plaza Nautilus	
John Rogo	John Rogo CPA	
June Romar	Romar Dance Studio	
Mary L. Rose	Roof Pro Inc.	
Amy O'Malley	O'Malley & Langan, PC	(485) 696-7777
Thomas O'Dea	O'Dea Construction Co.	

***** END OF FILE *****

```

Select
ARROWS -Change Record, ← -Select Record, U -Update Record
S -Set Selection, Z -FUZZY search, R -Reset C -Continuous on
F -Format, H -Hardcopy, X -Exit

```

If you perform the query above by pressing ENTER, you will be brought to the first record that matches.

Press B to browse the results, and you will see the following screen. However, no matter which browse format you are using, there is no ZIP code on it! But, you do know how to add a field to the browse format. Do that before looking at any of these records.

Press F and update the screen to add field 7 to the browse format line. Put the Zip code field to the right of the other fields on your screen and enter the header "Zip Code".

When you are done adjusting the browse to include the Zip code field, **press X** again to view the records found by your scan. The screen should look something like this.

Contact Name	Company	Zip Code
Sally Smith	Rand Corporation	22722-4444
William Hall	Halls' Flowerworld	12345
Gerald Hoben	Ceco Associates Inc.	47223
Kevin Icker	The Ice Cream Cottage	46125
Joseph Loftus	Continental Transmission	56987
Jason Malcolm	Forum Plaza Nautilus	58620
John Rogo	John Rogo CPA	47265
June Romar	Romar Dance Studio	47265
Mary L. Rose	Roof Pro Inc.	47223-4569
Ann O'Malley	O'Malley & Langan, PC	47273
Thomas O'Dea	O'Dea Construction Co.	58965

***** END OF FILE *****

-Select-
 ARROWS -Change Record, ← -Select Record, U -Update Record
 S -Set Selection, Z -Fuzzy search, R -Reset C -Continuous on
 F -Format, H -Hardcopy, X -Exit

IMPORTANT : Notice that the correct records have been retrieved. They are all greater than or equal to 079, but they are not in sorted order! The records may not be in any special order. As stated earlier, scan normally does not use indexes to do its work. Scan merely stops at every record (in record number order) and tests to see if the criteria for the query is met. If so, it selects the record, if not it goes to the next record and tests it. You will see only records that match your query, but they are not necessarily in any particular order. Since scan stops on every record in the file to do its testing, it is VERY slow compared to using an index to find a particular match. Using an index to find the first 079 in a large file of records causes filePro to jump over all the records lower than 079 (as this indexed field data is in ascending numerical order) and stop immediately at the first one that is equal to or greater than the criteria (079). All you need to remember now is how scanning and indexing are different. It will help you in future use of filePro.

One of the additions to de-facto queries implemented by filePro is the RG(range) function. This operator will automatically prompt you for the lowest match that you are looking for, and then prompt you for the highest match you want. It speeds up the query entry considerably.

Enter the following query. When you choose RG, it will automatically ask you for the lowest and highest limits for the zip code.

```

4.8.05K409 DEMO
INQUIRE, UPDATE, ADD
1- name_last      7- zip           13- n)Notes
2- name_first    8- p)phone_home 14- e)email-personal
3- aO)address_1  9- p)phone_business 15- e)email-business
4- aO)address_2 10- p)phone_fax
5- city          11- company_name
6- state         12- n)Notes

Enter Field To Select By: 7
Enter Relationship (EQ,NE,GE,GT,LE,LT,RG,CO): rg
Enter Lowest zip: 074
Enter Highest zip: 075
Enter Connective (and/or):

Press PgDn For Next, PgUp For Previous Fields
Press F5 To Toggle Between Field Headings And Lengths/Edits
  
```

After filePro brings you to the first matching record, **press B** to see the full browsed list of matches shown below.

Contact Name	Company	Zip Code
Robin Hakan	THE VALOR GROUP	07436-0777
Mark Farmer		07555
Jamey Gibson	Anytime Lock & Key Inc	07444

***** END OF FILE *****

-Select-
 ARROWS -Change Record, ← -Select Record, U -Update Record
 S -Set Selection, Z -Fuzzy search, R -Reset C -Continuous on
 F -Format, H -Hardcopy, X -Exit

Sometimes a query will bring up more than one screen full of records. In these cases, you can continue to use browse and the **Page Up**, **Page Down** keys to view the records that match the query. While on the browse screen you can also use the Up and Down arrows to view the retrieved records, but remember, if you are on a full screen, you can only go "forward" in the scan by pressing **ENTER**. If you DO press an UP or DOWN arrow during a scan selection from a full screen, you may see a record that DOES NOT match your criteria. To see this VERY IMPORTANT concept about scanning graphically, do the following:

Enter a query selecting by Cities that are not equal to Pat. It should look like this:

4.8.05K409 DEMO

I N Q U I R E , U P D A T E , A D D

1- name_last	7- zip	13- n)Notes
2- name_first	8- p)phone_home	14- e)email-personal
3- a0)address_1	9- p)phone_business	15- e)email-business
4- a0)address_2	10- p)phone_fax	
5- city	11- company_name	
6- state	12- n)Notes	

Enter Relationship (EQ,NE,GE,GT,LE,LT,RG,CO): ne

Enter Field To Select By: 5

Enter city: Pat

Enter Connective (and/or):

Press PgDn For Next, PgUp For Previous Fields
Press F5 To Toggle Between Field Headings And Lengths/Edits

When the first matching record is retrieved, **press B** to browse the records. You can move the highlighted bar with the arrow keys to any record in the selection and you will never come to a record that does not match the criteria. However, put your highlighted bar on the following record (Hakan), then press **ENTER** to go to the full screen view of this record.

Contact Name	Company	Zip Code
Robin Hakan	THE VALOR GROUP	07436-0777
Sally Smith	Rand Corporation	22722-4444
Gerald Hoben	Ceco Associates Inc.	47223
Kevin Icker	The Ice Cream Cottage	46125
Joseph Loftus	Continental Transmission	56987
Jason Malcolm	Forum Plaza Nautilus	58620
Jason Mart	Uniserve Insurance Agenc	
Kate Rogo		
John Rogo	John Rogo CPA	47265
June Romar	Romar Dance Studio	47265
Mary L. Rose	Roof Pro Inc.	47223-4569
Deb Schöles	Merry Maids	
Bernadette Slick	Cozmic Production Compan	
Kate Lowry	Comics On The Green	
Ann O'Malley	O'Malley & Langan, PC	47273
Thomas O'Dea	O'Dea Construction Co.	58965
Richard Ploskona	NEPA's Auto Sales	
Smith Ray	The Hillier Group	

-Select
ARROWS -Change Record, ← -Select Record, U -Update Record
S -Set Selection, Z -Fuzzy search, R -Reset C -Continuous On
F -Format, H -Hardcopy, X -Exit

Full Screen View:

ROLODEX

Last Name : Hakan
First Name : Robin
Address 1 : 777 Lucky Way
Address 2 : Penthouse
City : Penthouse
State : NJ
ZIP : 07436-0777
Phone Home : (201) 422-3333
Phone Work : (201) 422-4444
Phone Fax : (201) 422-5555
Company : THE VALOR GROUP
Email Home : rhakan@home.com
Email Work : rhakan@work.com

Notes
Brilliant, Good Looking, Filepro Programmer
Likes Brownies

Screen 1 Enter Selection > Record: 1

Index Mode
D -Delete, H -Hardcopy, U -Update, X -Exit, F -Print Form, B -Browse

Press **ENTER** to see the next record of Smith.

While on the full screen view of Smith(record #2), press **ENTER** and you will be brought to the next match, Hoben (record #6).

ROLODEX

Last Name : Hoben
First Name : Gerald
Address 1 : 123 Anywhere Street
Address 2 :
City : Our Town
State : IN
ZIP : 47223
Phone Home : (123) 456-7895
Phone Work :
Phone Fax :
Company : Ceco Associates Inc.
Email Home : me@hotdog.com
Email Work :

Notes
this is the only guy in the world that has this e-mail address

Screen 1 Enter Selection > Record: 6

D -Delete, H -Hardcopy, U -Update, X -Exit, F -Print Form, B -Browse
← for Next Match

Everything is still working as you would expect, you are only seeing records within the desired selection set.

However, now press the **Up arrow** and you will see that you are brought to Hall (record #5) and this record does NOT match the criteria. The city is EQUAL TO Paterson and we asked for city NOT EQUAL TO Paterson.

```

ROLDEX
Last Name : Hall
First Name : William
Address 1 : 123 Lake Drive
Address 2 :
City: Paterson
State: NY
ZIP: 12345
Phone Home:
Phone Work: (123) 123-1234
Phone Fax : (321) 321-4321
Company: Hall's Flowerworld
Email Home:
Email Work:

Notes
best flowers

Screen 1      Enter Selection > |      Record:      5
D-Delete, H-Hardcopy, U-Update, X-Exit, F-Print Form, B-Browse

```

So remember to use the ENTER key to find the next match when on a full screen view.

CO Operator

The CO operator asks whether the specified field contains the supplied criteria anywhere within it. For example, does an address field contain the word "Avenue"? An address of "23 First Avenue" would match this criteria. Imagine that you have forgotten someone's phone number. All you can remember is that it had "4444" in it.

Enter this query, using the CO "contains" operator.

```

Contact Name      Company      Home Phone
-----
Sally Smith      Rand Corporation      (800) 444-4444
Steve Christian  AMC Credit Corp      (321) 101-4444
***** END OF FILE *****

-Select-
ARROWS -Change Record, ← -Select Record, U -Update Record
S -Set Selection, Z -Fuzzy search, R -Reset C -Continuous on
F -Format, H -Hardcopy, X -Exit

```

You should see two records for this selection as follows.

```

Contact Name      Company      Home Phone
-----
Sally Smith      Rand Corporation      (800) 444-4444
Steve Christian  AMC Credit Corp      (321) 101-4444
***** END OF FILE *****

-Select-
ARROWS -Change Record, ← -Select Record, U -Update Record
S -Set Selection, Z -Fuzzy search, R -Reset C -Continuous on
F -Format, H -Hardcopy, X -Exit

```

You may have noticed that certain fields on the rolodex screen have a p) in front of them. This marks them as associated files. Associated fields are unique to filePro and they are a very powerful feature of the program. They allow you to perform a query on a group of fields at one time without having to specify each field individually. If the desired criteria is found in any of the fields within the associated field group, the record will be selected for inclusion in the selection set.

To put an associated field group in a query, use its associated field "name". Since the "rolodex" file has an associated field group defined as p), this is what you use on the selection set screen.

Enter the following criteria:

```

INQUIRE, UPDATE, ADD      4.8.05K409 DEMO
1- name_last      7- zip      13- n)Notes
2- name_first     8- p)phone_home      14- e)email-personal
3- a0)address_1   9- p)phone_business  15- e)email-business
4- a0)address_2  10- p)phone_fax
5- city          11- company_name
6- state         12- n)Notes

Enter Relationship (EQ,NE,GE,GT,LE,LT,RG,CO): CO <
Enter Field To Select By: p) <
Enter p)phone_home: 4444 <
Enter Connective (and/or): <

Press PgDn For Next, PgUp For Previous Fields
Press F5 To Toggle Between Field Headings And Lengths/Edits

```

The following records match the criteria.

Contact Name	Company	Home Phone	Business Phone
Robin Halgan	THE VALOR GRO	(201) 422-3333	(201) 422-4444
Sally Smith	Rand Corporat	(800) 444-4444	(800) 444-1234
Steve Christian	AMC Credit Co	(321) 101-4444	
***** END OF FILE *****			

-Select
 ARROWS -Change Record, F5 -Select Record, U -Update Record
 S -Set Selection, Z -fuzzy search, R -Reset C -Continuous On
 F -Format, H -Hardcopy, X -Exit

NOTE: We changed the browse format to reflect two phone numbers.

Testing for one piece of criteria is fine, but not very powerful. How can we limit or expand the criteria searching capability? This is where the "Connective..." prompt we have been skipping comes in. With this, you can test two different criteria at once. We can use this connective to specify that both criteria have to be true (the first AND the second), or that only one OR the other has to be true in order for a record to be retrieved.

Assume you have made brownies and want to deliver them by hand to only your local friends who like them. By performing the following search, only records for people in town will come up.

Enter a query that selects records containing the word **brownie** in either of the note fields. At the "Connective..." prompt, type the word "and". Now enter a query that selects records containing the word **wash** in the city field. It should look like this:

```

INQUIRE, UPDATE, ADD 4.8.05K409 DEMO
1- name_last          7- zip              13- n)Notes
2- name_first        8- p)phone_home    14- e)email-personal
3- a0)address_1     9- p)phone_business 15- e)email-business
4- a0)address_2    10- p)phone_fax
5- city             11- company_name
6- state           12- n)Notes

Enter Field To Select By: n)
Enter Relationship (EQ,NE,GE,GT,LE,LT,RG,CO): co
Enter n)Notes: brownies

Enter Connective (and/or): and

Enter Field To Select By: 5
Enter Relationship (EQ,NE,GE,GT,LE,LT,RG,CO): eq
Enter city: wash

Press PgDn For Next, PgUp For Previous Fields
Press F5 To Toggle Between Field Headings And Lengths/Edits
  
```

The only records to match will be the ones that "contain" the word brownies in either field 12 or 13 "AND" the city is "EQUAL" to Washington.

To see the connective feature on Short Selection work as an "OR", lets try the following criteria.

Enter a query that selects records in which the company name is not empty OR the email is not empty. In other words you want filePro to pick out the records that contain any company name or any email information.

While this may seem difficult at first, try to think of how you can represent blank fields. As you can see below, it is still just simple criteria.

```

INQUIRE, UPDATE, ADD 4.8.05K409 DEMO
1- name_last          7- zip              13- n)Notes
2- name_first        8- p)phone_home    14- e)email-personal
3- a0)address_1     9- p)phone_business 15- e)email-business
4- a0)address_2    10- p)phone_fax
5- city             11- company_name
6- state           12- n)Notes

Enter Field To Select By: 11
Enter Relationship (EQ,NE,GE,GT,LE,LT,RG,CO): ne
Enter company_name:

Enter Connective (and/or): or

Enter Field To Select By: e
Enter Relationship (EQ,NE,GE,GT,LE,LT,RG,CO): ne
Enter e)email-personal:

Press PgDn For Next, PgUp For Previous Fields
Press F5 To Toggle Between Field Headings And Lengths/Edits
  
```

The above selection will bring up all records where the company field OR the email field has something in it. (Not equal to "blank" means there is "something... anything besides space" in the field.)

Contact Name	Company	E-mail- Personal	Work
Robin Hakan	THE VALOR GROUP	rhakan@home.com	rhakan@work.com
Sally Smith	Rand Corporation	sallyea@1.com	sallysm@work.co
William Hall	Halls' Flowerworl		
Gerald Hoben	Ceco Associates I	me@hotdog.com	
Kevin Icker	The Ice Cream cot		you@doggiedo.com
Joseph Loftus	Continental Trans		
Jason Malcolm	Forum Plaza Nauti		
Jason Mart	Uniserve Insuranc		
John Rogo	John Rogo CPA		rogo@hotmail.com
June Romar	Romar Dance Studi		
Mary L. Rose	Roof Pro Inc.		
Deb Scholes	Herry Maids		
Bernadette Slick	Cozmic Production		
Kate Lowry	Comics On The Gre		
Ann O'Malley	O'Malley & Langan	lostefp.com	foundefp.com
Thomas O'Dea	O'Dea Constructio		
Richard Ploskona	NEPA's Auto Sales		
Smith Ray	The Hillier Group		

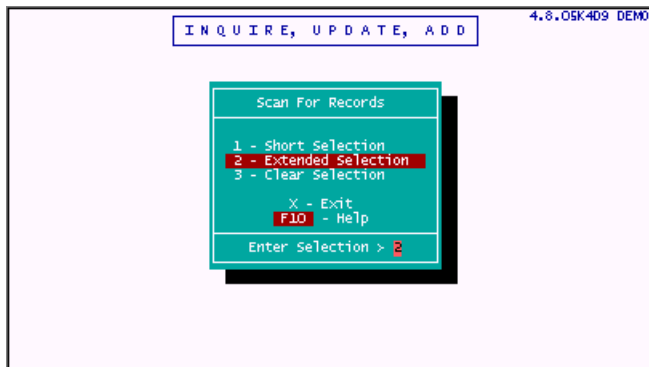
Select
 ARROWS -Change Record, ← -Select Record, U -Update Record
 S -Set Selection, Z -fuzzy search, R -Reset C -Continuous On
 F -Format, H -Hardcopy, X -Exit

Extended Selection

The "Connective..." prompt on Short Selection is sometimes not enough for complicated queries. Besides, you may want to save a query for later use (so you don't have to type it all in again). Extended Selection addresses both needs. Queries can be made up of many, many connectives (**and**, **or** and even **not**). In fact, if one screen full of connectives is not enough, you can attach up to five screens together!

Enter the Extended Selection screen.

From the Scan for Records menu, **Select 2 - Extended Selection.**



The same query we did on Short Selection for locals who like brownies would be done as follows using the extended format screen:

Group	Field	Heading	Rel	Value
	n)	n)Notes	co	brownies
	5	city	eq	Wash
Selector Sentence:				

Press ESC To Record Selection, Ctrl-C To Cancel, F10 For Help

The extended selection screen has a set of rules for how lines work with each other. The most important rule is that each line automatically defaults to the **and** connective. In other words, each line of the screen has to be true for a record to be considered a match. This is always true, until you learn how to override this default.

There are several ways to make the lines become OR conditions instead of the usual ANDs. The simplest of these is to "group" the lines you want to be OR conditions in the same group. A group can be any "name", i.e., "a", "abc", "fred", "group1", etc.

By naming the two lines from the previous selection set as "a", they will not both have to be true in order for a record to match this query. Now, either one OR the other being true means the record matches, and we want to select it.

Group	Field	Heading	Rel	Value
a	n)	n)Notes	co	brownies
a	5	city	eq	Wash

Selector Sentence:

Press **ESC** To Record Selection, **Ctrl-C** To Cancel, **F10** For Help

F8 To Display Fields

If you try the selection set above, it will select two records. Each record has at least one of the criteria met.

IUA Hints

Some Hints For Using IUA

The Duplicate Key

While you are entering data into the fields of a record in IUA there is a shortcut to save you lots of time. It is called the "duplicate" key. In Windows systems, this key is **F5**. It operates as follows. F5 will fill the field your cursor is standing in with the contents of the same field from the last "SAVED" record. Therefore, if you are entering many records with similar data (for example the city is the same on many records), **press [F5]** when you reach the city field on each record, and filePro will keep filling in this field for you automatically.

Insert Mode - Toggle

In version 5.0 and later, you can toggle between "Insert" and "Overwrite" modes by pressing [**Alt**] [**F9**] or the equivalent keys in *NIX which is typically [**Ctrl**] [**Z**]. Refer to the terminal guide for your specific terminal if the aforementioned keys do not work.

Last Record

If you want to quickly get to the last record, while using in the Index Selection in "Inquire/Update/Add", **press F7** when prompted for the index value. This will take you to the last record for the selected index.

Today's Date

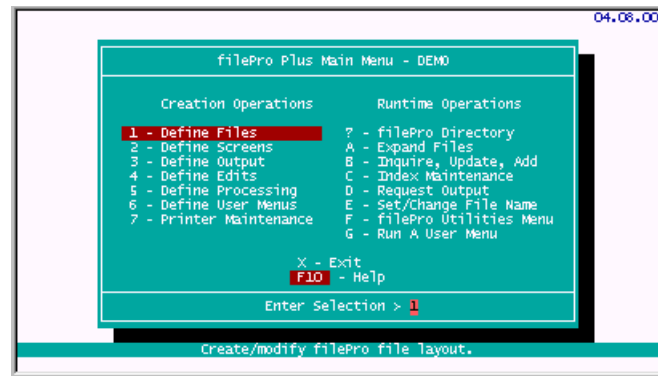
If your cursor is in a date field and you want to enter today's date, all you have to do is put a forward slash (/) as the first character in the field and press **ENTER**. Today's date will fill in automatically. The same is true for time fields, except that to insert the current time into a time field, you must put a colon (:) as the first character.

Key Table & User Count

On *NIX systems, try pressing [**ESC**] [**?**] to display the "Key Table" for your system. [**Alt**] [**F10**] will display the "Key Table" on Windows systems. The registered License # and session count for the product you are in will display at the bottom. The session count may be different depending on the program and how many sessions you license that program for..

Creating A File

Select 1 - Define Files.



NOTE: You may see a different set of filenames on your screen, or only one filename "rolodex" (used in the Primer Guide). It is unimportant how many files already exist when you are creating a new filePro file. You may ignore any other names you see and choose [NEW].



Enter **vidcust** as shown below.



Please keep the name "vidcust". (Do not change names in these steps, as they will be used throughout all of the examples.) Press **ENTER** to save this screen entry.

You will see the following screen:

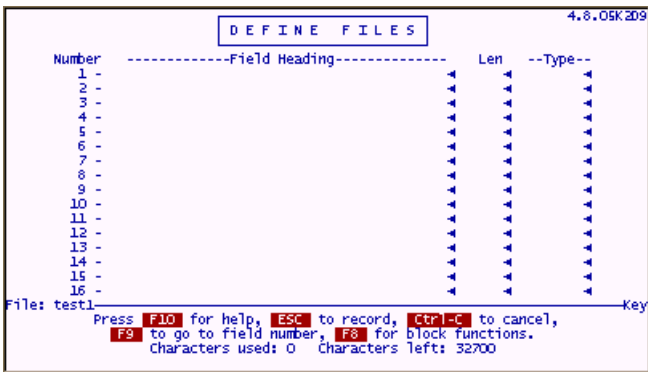


Select 1 - filePro.

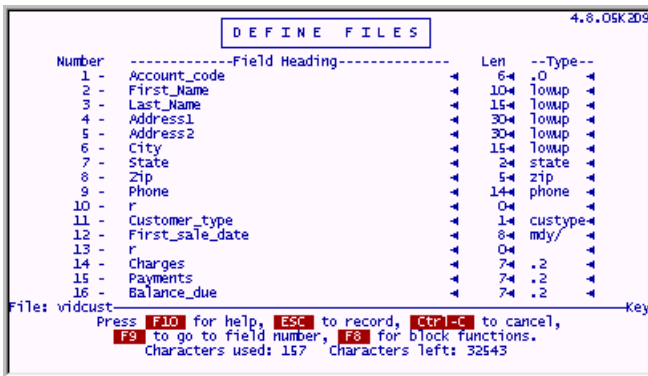
You will see a prompt for entering a creation password for this file. Do NOT do this. Press **ENTER** to continue without creating a password.



You will see the following screen:

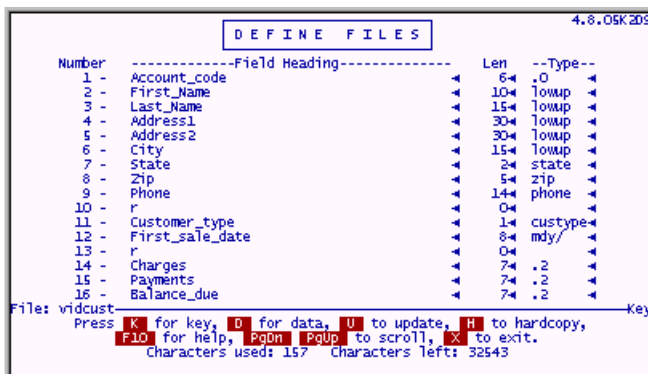


Enter the following information:

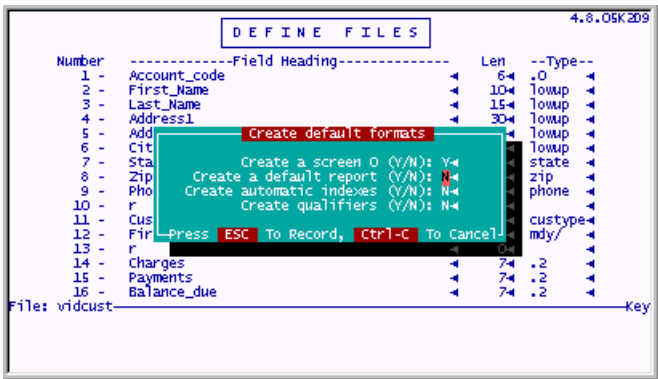


When you are done **press ESC** to save your work.

The following prompts will appear. **Press X** to finish the file design.



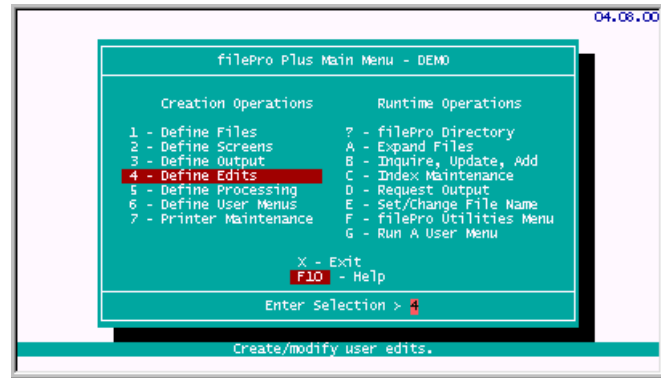
A popup "options" screen will appear. **Put a Y** in the designated field. (Create a screen 0)



The file "vidcust" is now created and would be ready for use, except that there is an edit type we designated which does not exist. This "user edit" was called "custype" and we must define it now.

Defining a User Edit

At the main menu, select 4 - Define Edits.

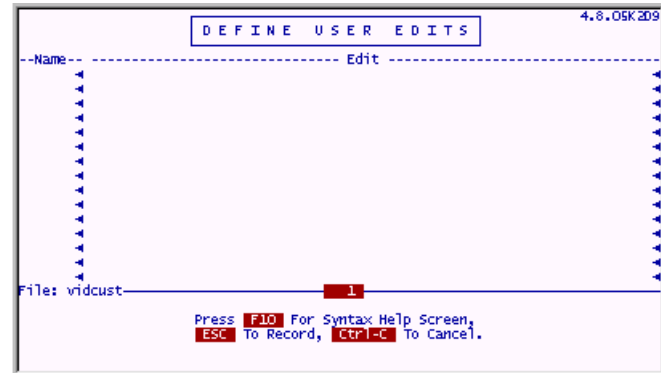


Choose the file "vidcust".

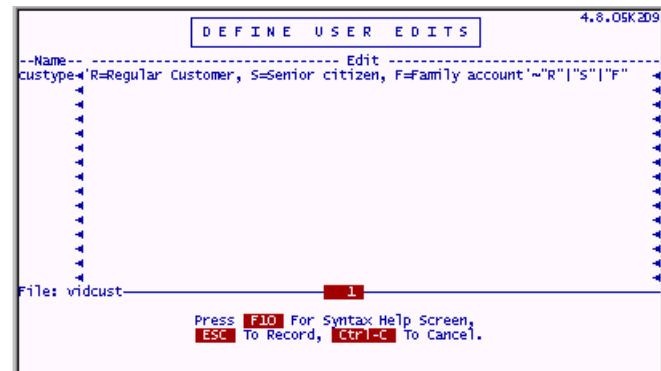


Answer Y to the prompt "Is this a new edit dictionary?" It is new, because we haven't designed any user edits for the file "vidcust" yet.

The following screen appears:



We are going to design a special edit or "filter" for data entry. Any field that uses this edit type will only allow the operator to enter an "R", "S" or "F" in the field. These are abbreviations for "Regular", "Senior Citizen" and "Family". We can keep control over the types of accounts we have in the video store using this edit to make sure we have no Z or Q accounts, or anything other than R, S or F accounts.



Edits work by following a series of rules. The rules we need for our customer type edit are simple. We only want to allow R, S or F in the field. While we are at it, the edit builder allows us to specify a "prompt" that will come up on the bottom of the screen each time the user's cursor goes into the customer account type field. This is also shown above. Prompts are put in apostrophes before the edit rules.

To see the rules for designing edits press the **HELP** key **F10** while entering this data.

```

Edits Syntax
In the following descriptions, "X" and "Y" are edit expressions,
and "L" is any literal, surrounded by quotes.

{ X } -- Parentheses may be used to separate expressions as in algebra.
[X ] -- The expression X is optional.
{ X } -- The expression X may occur any number of times, but must
occur at least once.
< L > -- The literal may appear, but if it doesn't, filePro will add it.
For example, "Y<es>" will accept either "Y" or "Yes" as input
and will turn a "Y" into a "Yes".

! L ! -- The literal must appear, and filePro will delete it.
X | Y -- Either expression is permitted.
For example: "N" | "N"lo!
will accept only "N" or "No", and will turn a "No" into an "N".
X & Y -- The data must conform to both expressions.

(cont'd)
← - Exit, PgUp, PgDn To Scroll, F9 - Search

```

```

Edits Syntax (cont'd)
^ -- Accept any single character.
^ -- At beginning of line. Right-justifies the resulting field.
~ -- Ignores case differences. Takes effect where it occurs on line.
~ -- Turns off case conversion. Takes effect where it occurs.
~ -- Converts data to uppercase. Takes effect where it occurs.
~ -- Converts data to lowercase. Takes effect where it occurs.
For example, the edit ~"N_<o>" will accept any of the
following as input and turn it into "No".

N n no NO No nO

Punctuation may be combined to form the following functions:
{ X } -- The expression may occur any number of times, or not at all.
{ ! L ! } -- If the literal appears, it will be deleted.

(cont'd)
← - Exit, PgUp, PgDn To Scroll, F9 - Search

```

You will see in the help screen that the "~" tilde causes the edit to make all entries UPPER CASE, and the pipe "|" character separates "allowable" entries. Users can only enter the three designated characters e.g. "R", "S", "F" or leave the field blank.

Note: A BLANK is always acceptable in any edit as a default. If you do not want to allow a blank, you must program this in a processing table.

Once you are done entering the edit name and its rules, press **ESC** to save your work.

You can test the "custype" edit by **pressing T** at the edit design screen. This will popup the following screen.

```

4.8.05K209
DEFINE USER EDITS
-----
--Name----- Edit
custype←R=Regular Customer, S=Senior citizen, F=Family account~"R"|"S"|"F"
Test user edits
Edit name: custype← (Press F6 for list of edits.)
Length: 1←
Test data: m←
Edit Failed At Cursor Position
Press F10 for help, Ctrl-C when done.
File: vidcust

```

You must enter the name of the edit to test and the length for which you will be testing. Once this is done, you will enter sample answers that a user might enter in the field. The edit checker will tell you which entries are valid and which entries will fail the edit. Experiment with some different letters and numbers to see how this works.

Finally, **type "S"** for "Senior Citizen" to see that it passes the edit test.

```

4.8.05K209
DEFINE USER EDITS
-----
--Name----- Edit
custype←R=Regular Customer, S=Senior citizen, F=Family account~"R"|"S"|"F"
Test user edits
Edit name: custype← (Press F6 for list of edits.)
Length: 1←
Test data: S←
Data passes edit.
Press F10 for help, Ctrl-C when done.
File: vidcust

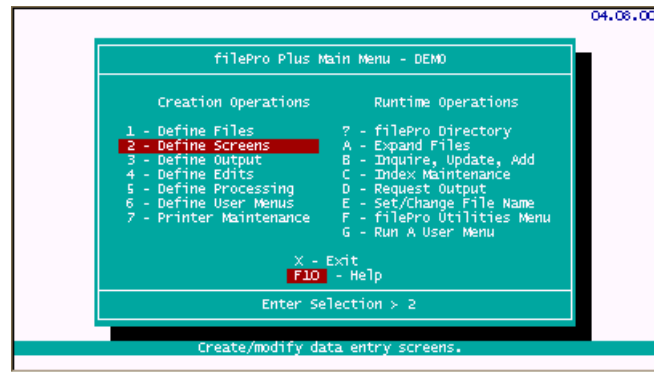
```

When you are done experimenting, **press Ctrl & C** then **press X** to return to the main menu.

Again, we could immediately go and enter data into this file, but let's not. Instead, let's look at the default screen that filePro built for us.

Defining a Screen

Select 2 - Define Screens.



Choose "vidcust".

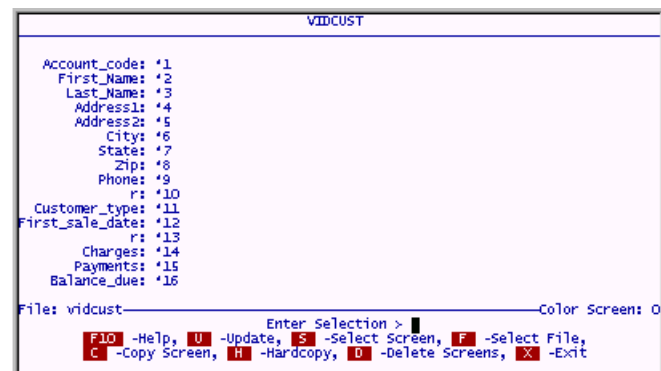


Earlier in this section when we defined our file, a "default" screen 0 was created. We can use this screen as it is, or modify it to suit our tastes. (We could also ignore it, or delete it and build any number of other screens.)

Choose screen 0.



You will see the following screen:



Press U to update this screen and modify it to look like the following screen:


```

VIDCUST
Account_code: *1
First_Name: *2
Last_Name: *3
Address1: *4
Address2: *5
City: *6
State: *7
Zip: *8
Phone: *9
Customer_type: *11
First_sale_date: *12
Charges: *14
Payments: *15
Balance_due: !16
File: vidcust                                     Color Screen: 0
F9 -Toggle Graphics, F6 -Display Fields, F5 -Resolve Fields
F10 -Help, F7 -Box Func's, F8 - Extended Func's
20,19 Ctrl F10 -Color Help, ESC -Record, Ctrl-C -Cancel

```

Notice that we have omitted fields 10 and 13. The fields with asterisks (*) will be available to the user. In other words, they can put their cursor "into" these fields. The fields with exclamation points (!) are called "protected" fields and they will not allow the user to put his cursor in them. These are fields that will be calculated or displayed by your program. The (!) protects the field from being changed by the user. Another symbol you may want to use is the percent sign (%). This symbol designates that the corresponding field MUST be filled. If the field does not contain data and the user tries to SAVE the record, a warning will be given and the cursor will be returned to this field automatically.

filePro builds our default screen:0 by filling in as many of the fields as it can fit onto the screen. It does this sequentially starting from field 1. It will continue putting fields on the screen in multiple columns until it runs out of space on the screen. It will stop at the first field that causes it to reach the right side of the screen or wrap around it. When it built the "vidcust" default screen, it put the two fields we named "r" on this screen. These fields wont be used at this time. The "r" just stands for "reserved". This is not a filePro term, just a convention. Some programmers label these fields as "spare". Reserved fields are sometimes used just to separate different sections of a file's layout as we did here. They have no place on any screen so we removed them.

When your screen looks like the one shown above, **press F5** to see how it will look when it is used in Inquire, Update and Add. This is called "resolving the fields". The markers are called the "end of field markers" and they show graphically where the data for each field will end.

```

VIDCUST
Account_code:
First_Name:
Last_Name:
Address1:
Address2:
City:
State:
Zip:
Phone:
Customer_type:
First_sale_date:
Charges:
Payments:
Balance_due:
File: vidcust                                     Color Screen: 0
4,24 Press [Left Arrow] To Continue

```

Press ENTER to stop the "resolve fields" display and then **press ESC** to save your work. Return to the main menu by **pressing X**.

New in Version 5.8.02 and higher

To implement a scrolling field, place a field as you normally do, but then place a backslash ("\") at the location you want to truncate the visible part of the field. Then, in *clerk, the field will only display as wide as the place you specified with the "\". However, when you are in the field, you can scroll horizontally. The EOF (end of field) designator will now also show a > indicating additional characters for a field when viewing a record. DO NOT attempt to scroll any field less than 3 characters.

New in Version 5.8.03 and higher

In dscreen, pressing the F8 – Extended Functions and the C – Change cursor path you can now press F5 to view the screen, then press ENTER to return to the Cursor path setup.

New in Version 6.0.00 and higher

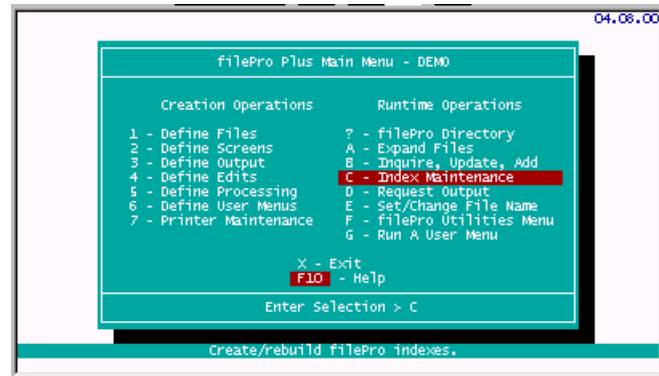
In dscreen on NEW screens, pressing the F8, and C - Change cursor path, you now have the option to select an automatic cursor path.

- Vertical selection
- Horizontal selection
- Natural Order

Note: Both horizontal and vertical will pick up dummy vars. "Natural order" will not.

Defining an Index

Select C - Index Maintenance.



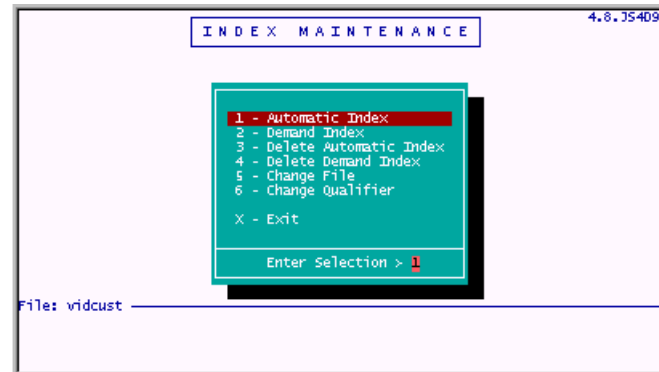
We need to build some indexes to use with the "vidcust" file. Indexes will allow us to rapidly find any record in our "vidcust" file based on sort definition.

Choose "vidcust".

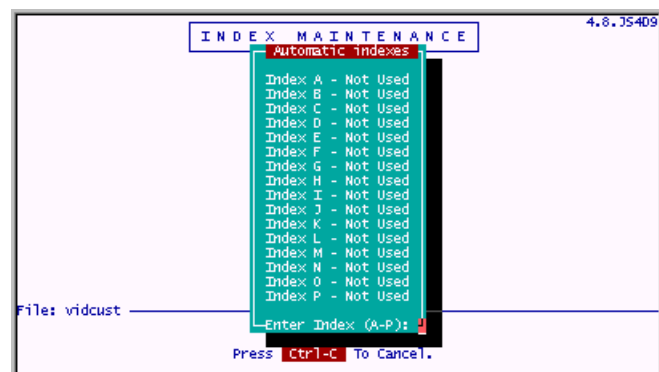


We will be using Automatic Indexes in the Video Store application. This means that they "automatically" update themselves every time you add, delete or modify a record. If an automatic index is changed by one of these actions, it will rebuild itself accordingly. (The opposite of this, Demand Indexes, will be discussed later.)

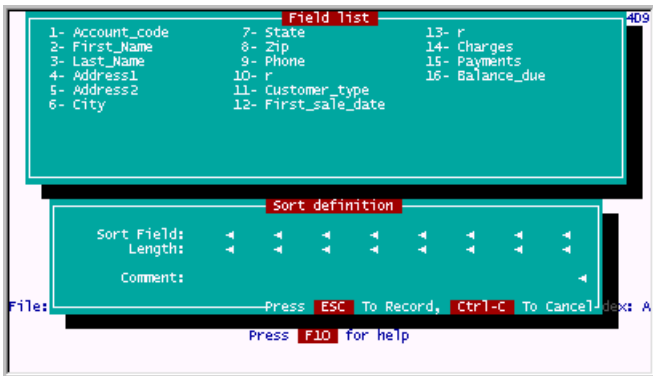
Select 1 - Automatic Index.



Choose A.



The following screen will appear:



Enter the following fields and comment for this index:

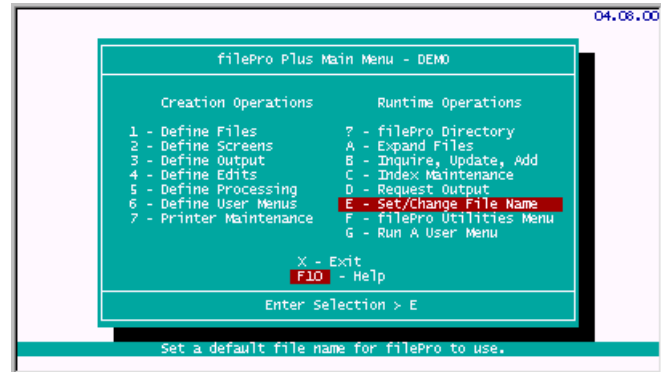


Press ESC to record. Return to the main menu.

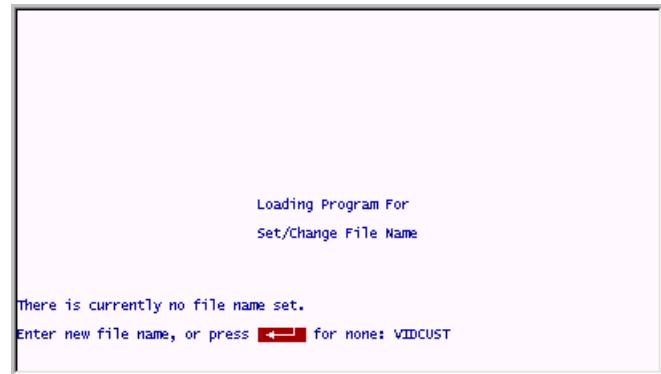
Setting the File Name

We have been doing a bit of design work with the "vidcust" file and we will be doing a lot more. Since it is time consuming and cumbersome to keep typing in the filename first for each operation, we can specify the "vidcust" name with the Set/Change File Name feature. This tells filePro that we want to skip the file name prompt for every operation on the main menu and use our designated choice as the default file name. It will save a good deal of time so lets do that now.

Select E- Set/Change File Name.



Enter the name "vidcust" and press ENTER.



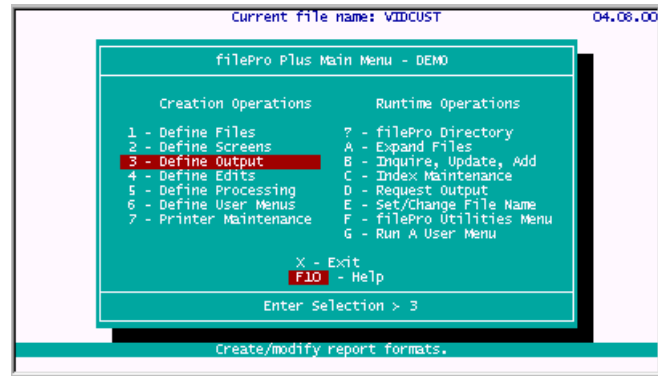
Notice that the file name "vidcust" now appears at the top of the main menu as the Current file name. We will not be bothered by this prompt again, until we set/change the filename to something else or clear it.



Defining an Output Format

We will now design an output format for this file. This will let us send the contents of this file to the printer.

Select 3 - Define Output.

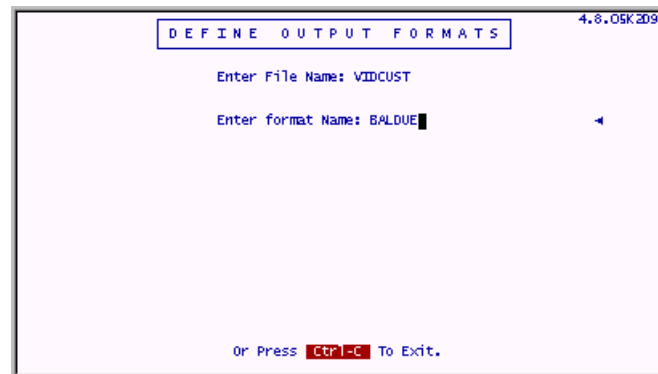


(See, the pesky "file name" prompt did NOT show up! It knows we are working in the "vidcust" file.)

Choose [NEW].

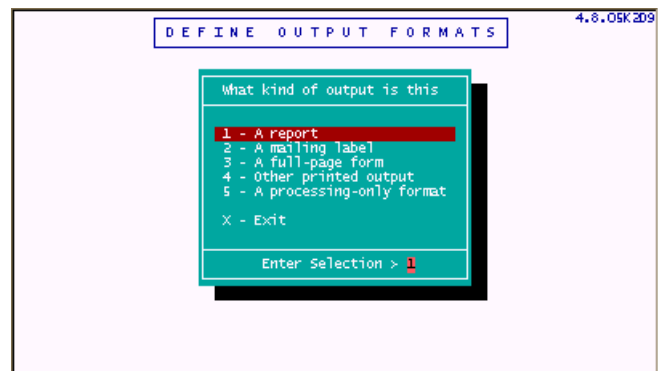


Enter the following.

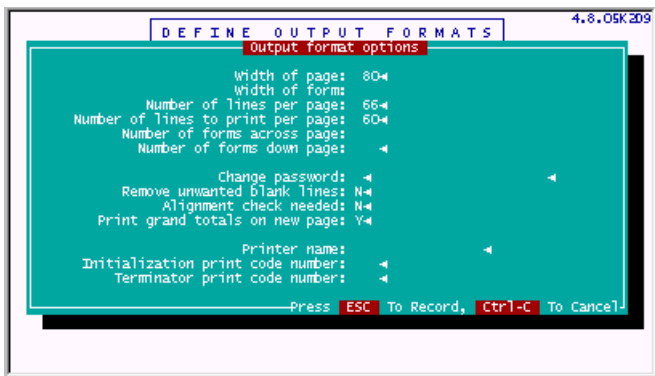


filePro allows you to define several different kinds of output. We will be building a simple "report" type output. The records will be shown several per page. Reports can carry totals and subtotals of records on the report. (Other kinds of output are "forms" and "labels". The "form" type usually shows one record per page and does not provide for subtotalling and totaling of the records. The "label" type prints records in label format, in multiple columns if desired.)

Select 1 - A report.



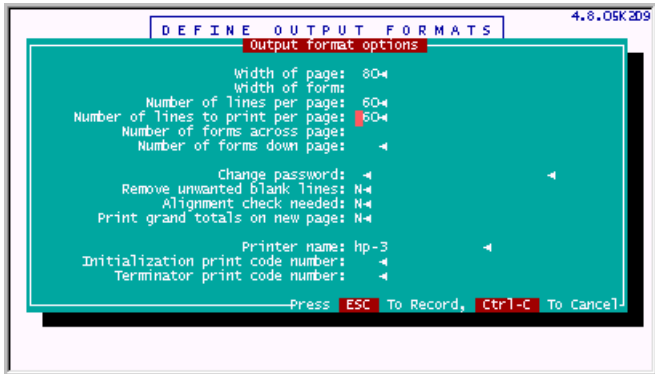
The following screen appears:



Change the default settings as follows: (You can press F6 to see a list of printers and printer types.)

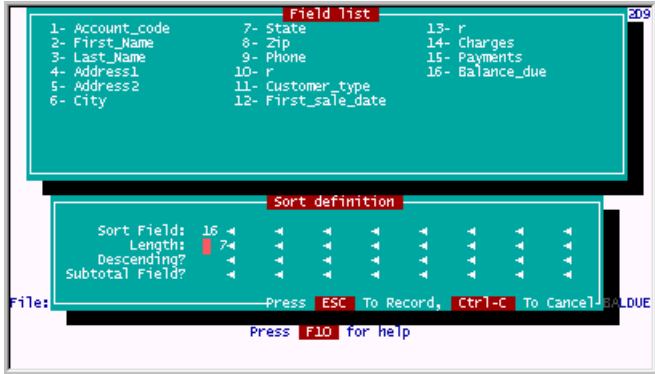
NOTE: version 5.8.03 When browsing printers in Options for output formats, filePro now shows only valid printers. Pressing F6 again will display the complete printer and printer type list as it did in previous version. F6 will toggle between the two lists.

Form filtering version 5.8.02



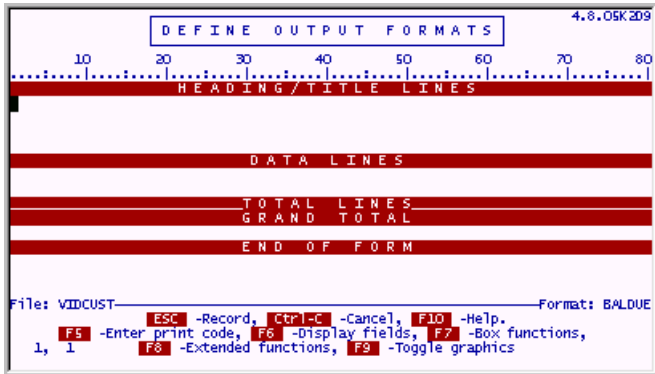
Press ESC to save your work.

The following screen appears blank. Fill it in as shown:

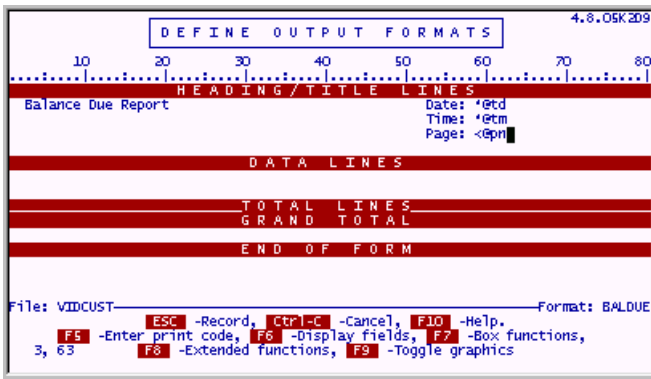


Press ESC to save your work on this screen.

The following screen appears:

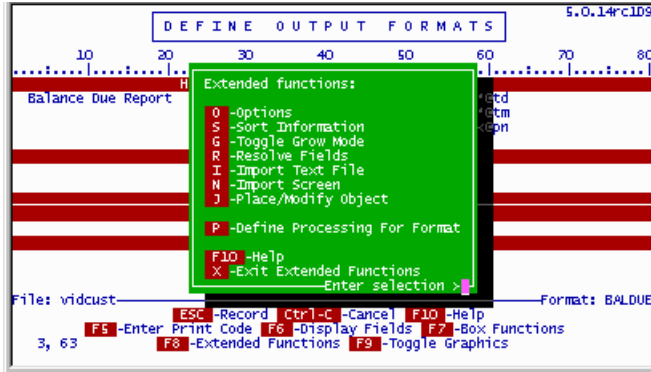


Enter the following text and fields:



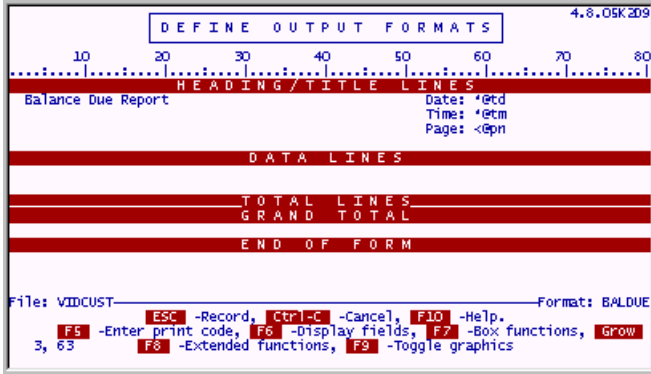
We do not have enough room on this default report format to put the things we want, so it is necessary to change the shape of the format somewhat. We will "grow" it a little.

Press the F8 key.



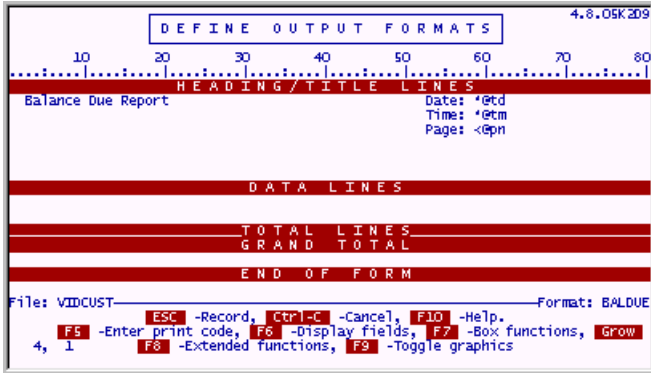
From this options popup screen, choose G.

This will turn ON the "grow mode". Note this fact is displayed by the word GROW in reverse video at the bottom right corner of the screen.



While in grow mode, the F3 and F4 keys work to add lines to, or remove lines from the form. We can literally gain lines of space in any section.

Make sure your cursor is in the Heading/Title Lines section and press F3 key two times. This will add two lines. The screen should look as follows:



Turn Off grow mode by pressing F8 and choosing G again. This option is a toggle. It is either ON or OFF.

Be careful, when not in grow mode, the F3 and F4 keys will add and remove BLANK lines. Any text or fields on the lines will move down or up accordingly. You can easily push whole lines of text and fields off the bottom of the format.

While you are in update mode on an output format, you can view the file layout by pressing F6.

Press F6.

1- Account_code	7- State	13- r
2- First_Name	8- Zip	14- Charges
3- Last_Name	9- Phone	15- Payments
4- Address1	10- r	16- Balance_due
5- Address2	11- Customer_type	
6- City	12- First_sale_date	

File: VMDUCST

PgDn And PgUp - Scroll Fields, H - Hardcopy, ← - Return
 Press F5 For Field Lengths and Edits, F6 For Different File

Use this map layout listing frequently to refresh your memory as to field names, numbers and lengths. (Press F5 while on this view to see lengths.)

Press Return and enter the following text and fields as shown:

DEFINE OUTPUT FORMATS				4.8.05K209			
10	20	30	40	50	60	70	80
HEADING / TITLE LINES							
Balance Due Report				Date: *@td			
				Time: *@cm			
				Page: <@pn			
Name		Phone		Balance Due			
DATA LINES							
*2	<5	*9	*16				
TOTAL LINES							
GRAND TOTAL							
Grand Total =16							
END OF FORM							
File: VMDUCST				Format: BALDUE			
ESC -Record, Ctrl-C -Cancel, F10 -Help,							
F5 -Enter print code, F6 -Display fields, F7 -Box functions,							
1, 61 F8 -Extended functions, F9 -Toggle graphics							

Draw a line on this format by using the "box" functions. Put your cursor at position 6,2 (note the "box" position is shown on the bottom left corner of the screen.) This will initiate the box functions and show you a cross-bar symbol where your cursor was placed.

The screen should look like this.

DEFINE OUTPUT FORMATS				4.8.05K209			
10	20	30	40	50	60	70	80
HEADING / TITLE LINES							
Balance Due Report				Date: *@td			
				Time: *@cm			
				Page: <@pn			
Name		Phone		Balance Due			
DATA LINES							
*2	<5	*9	*16				
TOTAL LINES							
GRAND TOTAL							
Grand Total =16							
END OF FORM							
File: VMDUCST				Format: BALDUE			
F10 -Help, X -Cancel box mode,							
D -Draw, E -Erase, B -Blank,							
6, 2 C -Copy, M -Move, O -Overlay, S -Save.							

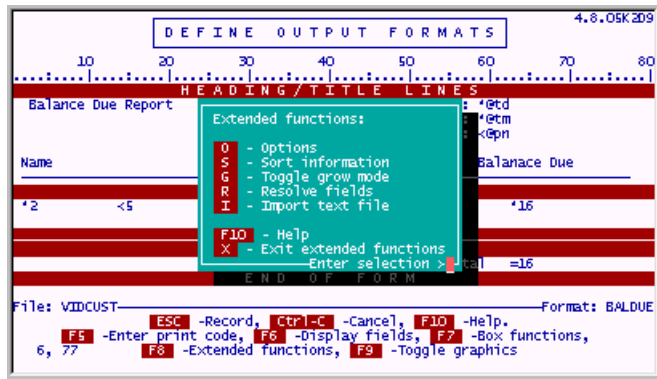
To draw the line, position your cursor at 6,77 (the cross-bar will not move, but the cursor will) and press D to draw the line.

The screen will look like this:

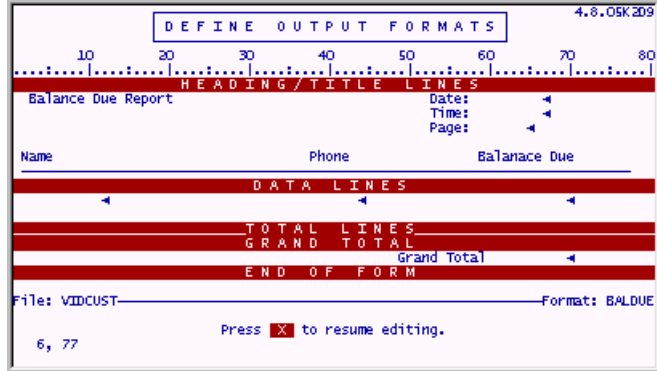
DEFINE OUTPUT FORMATS				4.8.05K209			
10	20	30	40	50	60	70	80
HEADING / TITLE LINES							
Balance Due Report				Date: *@td			
				Time: *@cm			
				Page: <@pn			
Name		Phone		Balance Due			
DATA LINES							
*2	<5	*9	*16				
TOTAL LINES							
GRAND TOTAL							
Grand Total =16							
END OF FORM							
File: VMDUCST				Format: BALDUE			
ESC -Record, Ctrl-C -Cancel, F10 -Help,							
F5 -Enter print code, F6 -Display fields, F7 -Box functions,							
6, 77 F8 -Extended functions, F9 -Toggle graphics							

Just as in Define Screens, filePro lets you "resolve the fields" to see how they will actually fit on the format. Use the F8 options popup to select this function.

Press F8 for the options screen, and then press R to resolve the fields on this output format.



The screen should look like this:



Remember that you can use the F6 key to see the fields and from there the F5 key to see the lengths and edits of the fields. Use a combination of F6/F5 and F8/R to line up any output format exactly the way you want it.

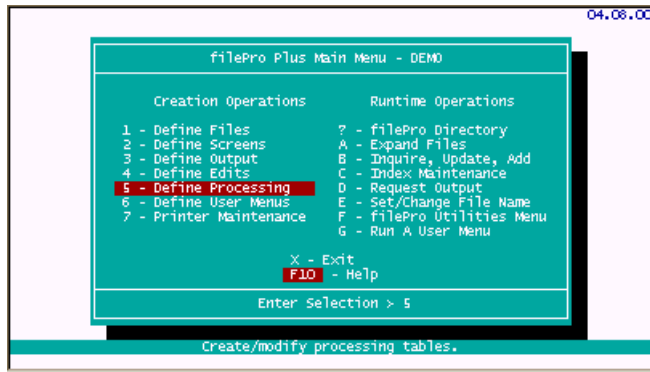
Press X to exit to resume editing, and when done editing, **press ESC** to save your report.

Press X to exit back to the filePro Main Menu.

Defining A Processing Table

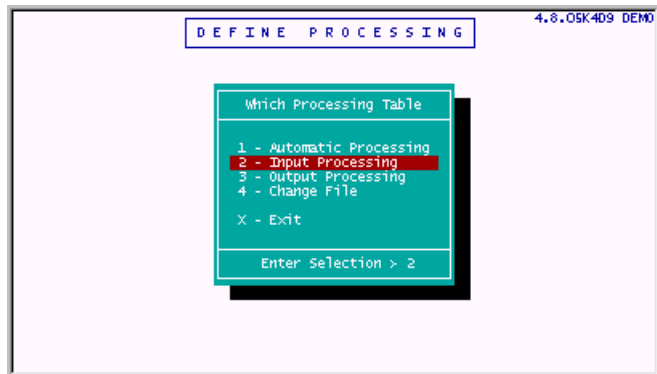
Let's put some simple processing into this application. Processing is where filePro allows you to manipulate the fields, records and files of your databases. This work is done on "processing tables".

Select 5 - Define Processing.

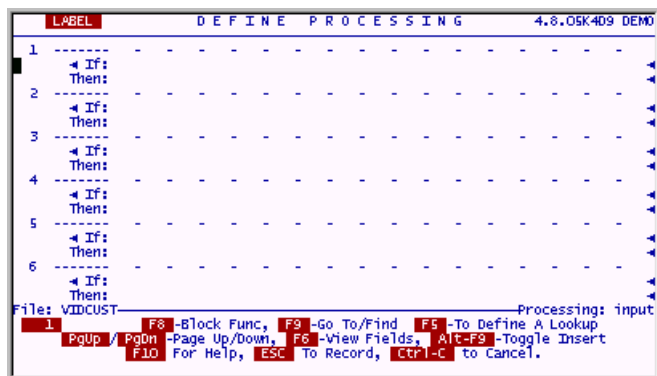


We will add some processing that will happen when the user is "inputting" data, or modifying it. The processing table usually associated with the Inquire, Update, & Add section of filePro is called the "input" table.

Select 2 - Input Processing.



The following blank processing table appears.



filePro bases its entire processing table strategy on the simplest computing structure. If something is true, then do this. That's it, that is all of it. Believe it or not, just about anything you can imagine doing in a computer can be done using this logic. Each tiny instruction is either true or not, and if it is, the action gets done. If it isn't true, it doesn't get done. By stringing these small "if-then" operations together, highly complex programs can be built.

There are several types of processing in filePro, two are shown below and used on this table. The first is INPUT processing. This is the code from line 1 to line 3 in the table below. This INPUT processing gets executed when the user stores or SAVES the data on records that are being added or modified. The user will be asked if the screen being displayed looks correct, if it does, the process is over. If the user indicates that he sees something wrong, the process puts him back on the screen and lets him make adjustments. When next he SAVES the screen, the question is asked again. Presumably, sooner or later, he will answer that it looks fine and the process will end.

The second kind of processing on this table (and much more useful) is "trigger" processing. This type of processing only happens when the indicated trigger is activated. On the following table the triggers are whenever the user's cursor leaves field 14 and field 15. At this point, filePro will make the Balance_Due field equal to the Charges field minus the Payments field. It does this by executing a subroutine called "totals" and then returning to display the fact that it did this and is ending. These little trigger processes can become much more elaborate, but the essentials of what can be done are shown here. When you actually try the program in a few minutes, you will see more graphically how this processing table actually works.

Later, the actual syntax on this table will be discussed and explained. For now, just type it in.

Enter the following:

```

LABEL          DEFINE PROCESSING          4.8.05K409 DEMO
1  -----
  < If:
  Then: input qq(1,yesno) "Does everything look alright? (y/n) "
2  -----
  < If: qq ne "y"
  Then: restart
3  -----
  < If:
  Then: end
4  -----
@wif14 < If:
  Then: gosub totals ; display ; end
5  -----
@wif15 < If:
  Then: gosub totals ; display ; end
6  -----
totals < If:
  Then: Balance_due = Charges - Payments ; return
File: VDCUST ----- Processing: input
1  F8 -Block Func, F9 -Go To/Find
  PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
  F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

When you are done, **press ESC** to save your work.

The following screen will appear. **Enter Y** to check the accuracy of your work. If it passes the syntax check, it is likely that you entered everything correctly.

```

          DEFINE PROCESSING          4.8.05K409 DEMO
          -----
          Check Syntax? (Y/N) █

```

A hardcopy prompt appears. It would be a good idea to hardcopy this table, so you can review it later when you try the application. The hardcopy prompt shows (Y/S/N). The S stands for short. A table will be produced that does not have all the dashed lines that separate processing elements. It is your choice which to print.

A prompt appears asking if you want a Cross Reference Hardcopy. Do not print this now, we will discuss it later.

Version 6.0.00 enhancements

Saving a section of code under F8 options will now prompt for confirmation if prc already exists.

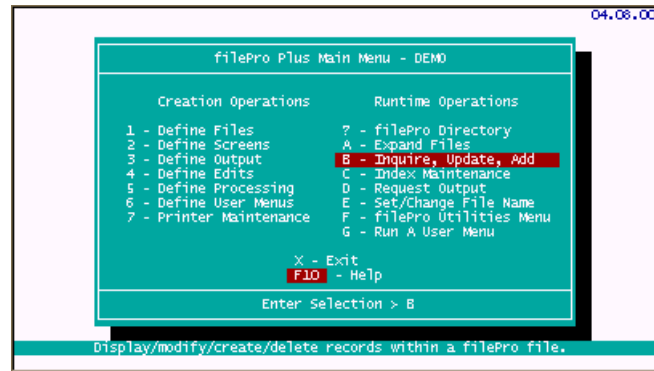
An alternate auto process can now be attached to a process. This process is used when creating the .tok files and syntax checking.

*clerk and *report will use this alternate process if there is no -y flag on command line.

Using IUA To Test File Design

It's time to try all this hard work out!

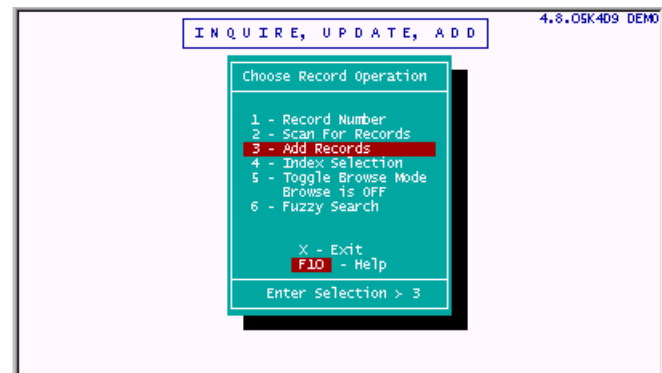
Select B - Inquire, Update, Add.



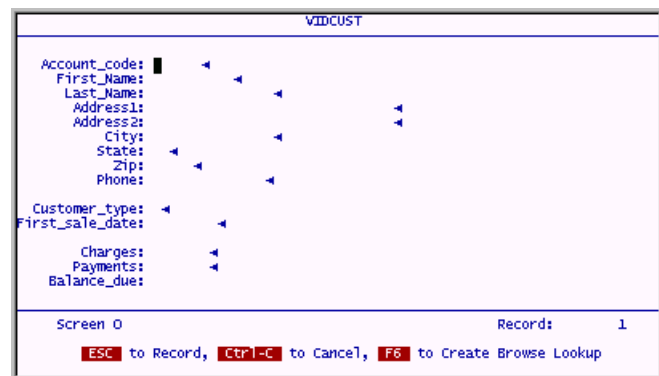
Select 0.



Select 3 - Add Records.



You will see a blank data entry screen as follows:



Enter the following data:

```

VIDCUST
Account_code: 100
First_Name: Karen
Last_Name: Smith
Address1: 111 Happy Lane
Address2: Apt. 2
City: Oakland
State: NJ
Zip: 07436
Phone: (201) 427-9888
Customer_type: R
First_sale_date: 10/04/99
Charges: 27.50
Payments: 13.00
Balance_due: 14.50
Screen 0                                Record: 1
ESC to Record, Ctrl-C to Cancel, F6 to Create Browse Lookup

```

When you are done, press **ESC** to save the screen.

You will be prompted with the following question. If you answer **(N)**, your cursor will be put back on the screen so you can fix whatever is wrong. Then press **ESC** again to save your work. Eventually, you should be able to answer **Y**.

```

VIDCUST
Account_code: 100
First_Name: Karen
Last_Name: Smith
Address1: 111 Happy Lane
Address2: Apt. 2
City: Oakland
State: NJ
Zip: 07436
Phone: (201) 427-9888
Customer_type: R
First_sale_date: 10/04/99
Charges: 27.50
Payments: 13.00
Balance_due: 14.50
Screen 0                                Record: 1
Does everything look alright? (y/n)

```

At each new blank screen, enter the next three records in the same manner as the first.

Enter the following data:

```

VIDCUST
Account_code: 101
First_Name: John
Last_Name: Jones
Address1: 1521 Cypress Street
Address2:
City: Hawthorne
State: NJ
Zip: 07806
Phone: (201) 443-2222
Customer_type: S
First_sale_date: 10/04/99
Charges: 104.00
Payments: 92.00
Balance_due: 12.00
Screen 0                                Record: 2
ESC to Record, Ctrl-C to Cancel, F6 to Create Browse Lookup

```

Press **ESC** to save the screen.

Enter the following data:

```

VIDCUST
Account_code: 103
First_Name: Ken
Last_Name: Blackburn
Address1: 20194 Applegate Drive
Address2: Apt 2E
City: Oakland
State: NJ
Zip: 07436
Phone: (201) 347-2828
Customer_type: R
First_sale_date: 10/04/99
Charges: 85.75
Payments: 85.75
Balance_due: .00
Screen 0                                Record: 3
ESC to Record, Ctrl-C to Cancel, F6 to Create Browse Lookup

```

Press **ESC** to save the screen.

Enter the following data:

Account_code: 104
First_name: Mary
Last_name: Rodrigues
Address1: 97 First Ave.
Address2:
City: Paterson
State: NJ
Zip: 07555
Phone: (201) 928-3838
Customer_type: F
First_sale_date: 10/04/99
Charges: 211.53
Payments: 185.00
Balance_due: 26.53

Screen 0

Record: 4

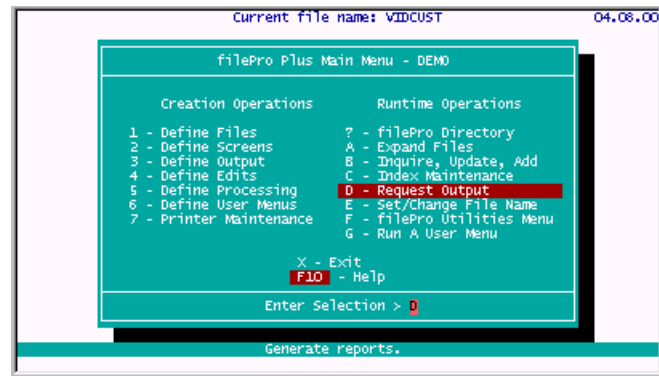
ESC to Record, Ctrl-C to Cancel, F6 to Create Browse Lookup

Press ESC to save the screen.

At the next blank record, press Control-C to stop entering records. Press X twice to back out of IUA to the main menu.

Requesting an Output Format

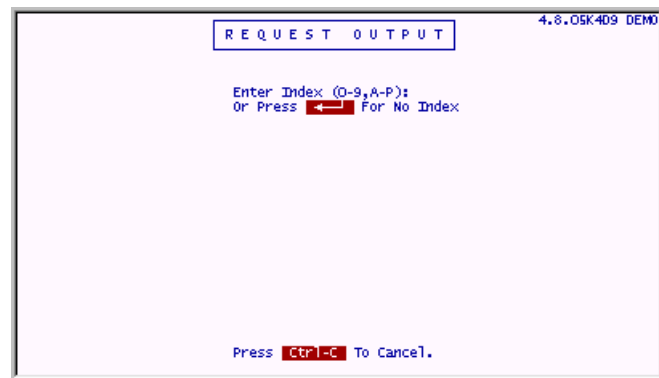
Select D - Request Output.



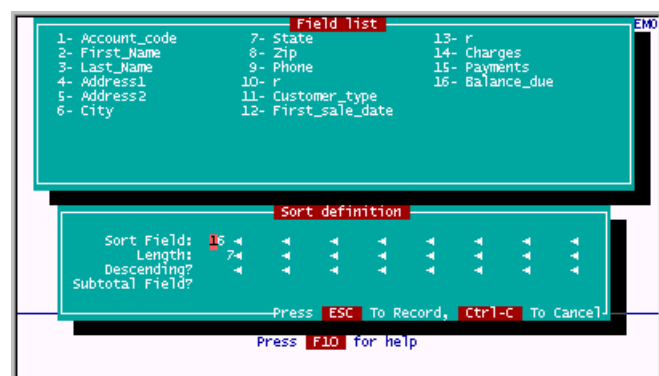
Choose BALDUE.



Press ENTER at the index prompt (for No Index).



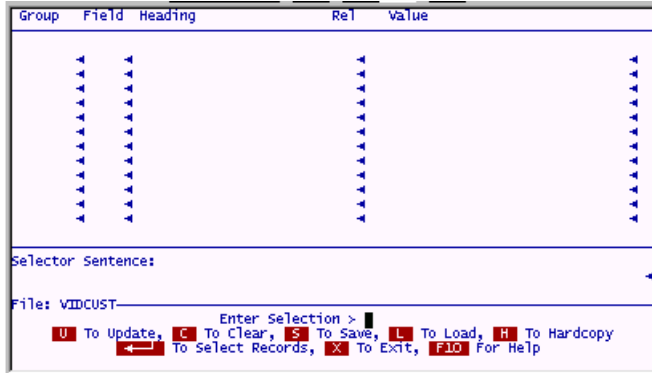
Press ESC at the Sort screen.



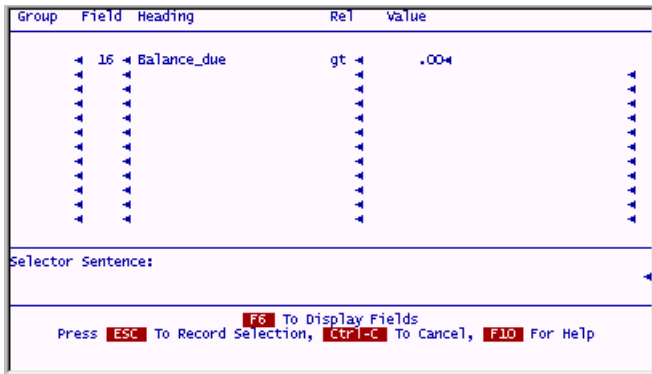
Enter N at the Select All Records prompt.



The following blank Extended Selection screen comes up. Press U to update it.



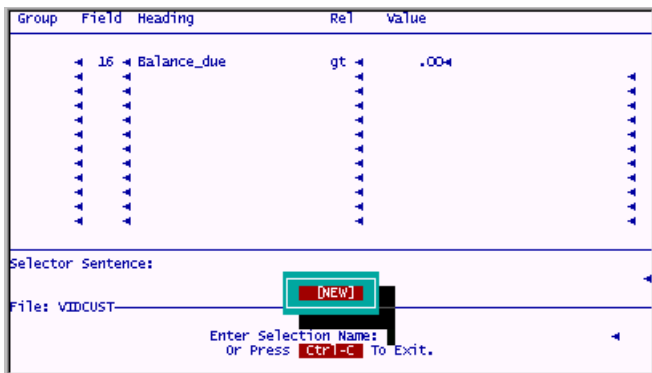
Enter the following criteria:



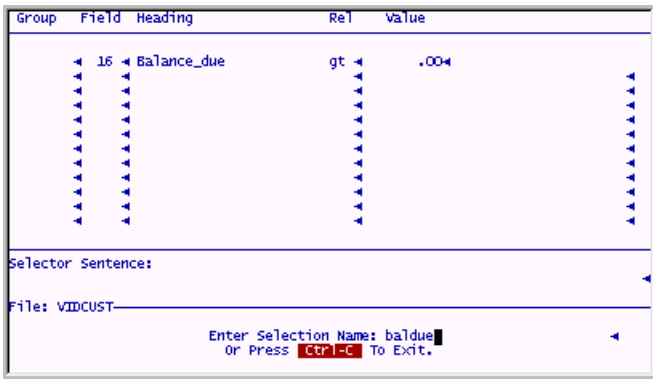
Press ESC to save this screen.

Since we will be using this selection set again, let's save it now and give it a name we can call upon later.

Press S and then choose [NEW]:



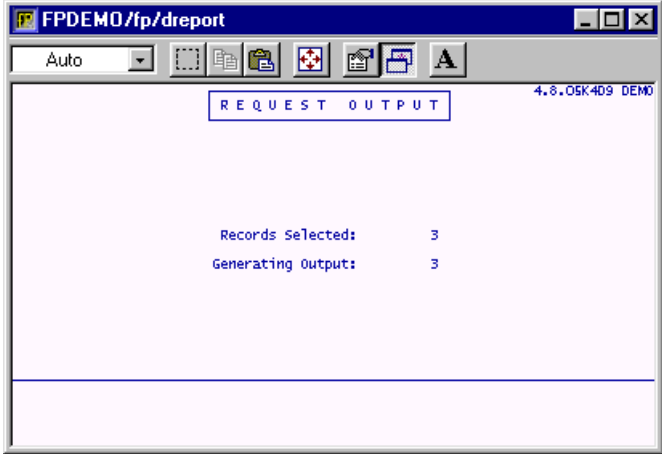
Enter the name "baldue".



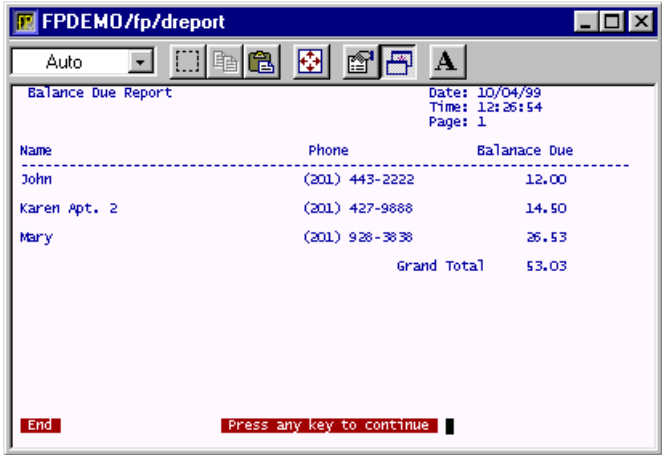
Once this selection set is saved, **press ENTER** to actually select the records for this output.

NOTE: As of Version 6.0 and higher, if an extended selection is set, all functions of dclerk will honor the selection criteria until the extended selection is manually cleared. (Option 3)

The following screen appears: (Your machine may be too fast to allow you to see this screen.)

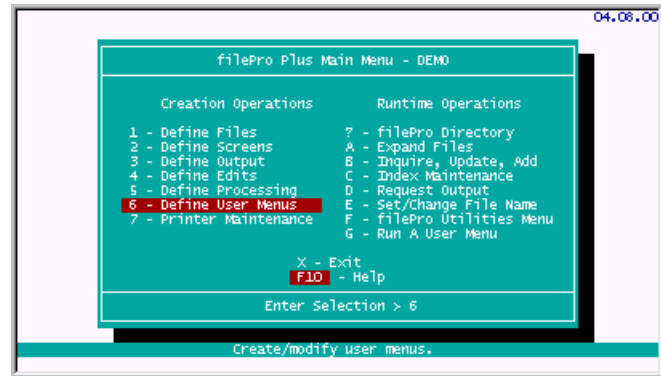


The following report should print on your printer. (The default printer for this guide has been specified as PRN. This is most likely the printer attached to LPT1. If this conventional setup is not so on your system, you will have to arrange things so that this tutorial will work. Later on, redirecting output and printer redirection is discussed.)



Defining a User Menu

Select 6 - Define User Menus.



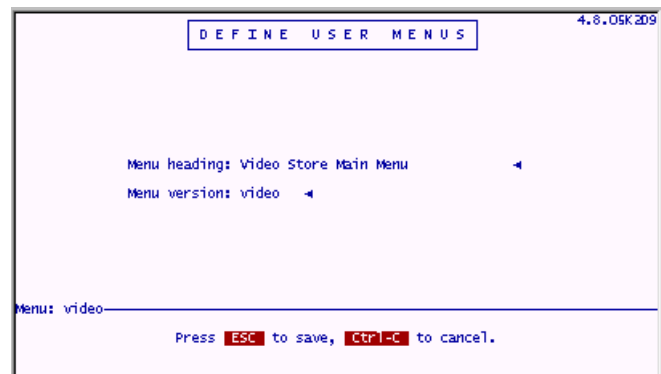
Choose [NEW]:



Enter the name "video".

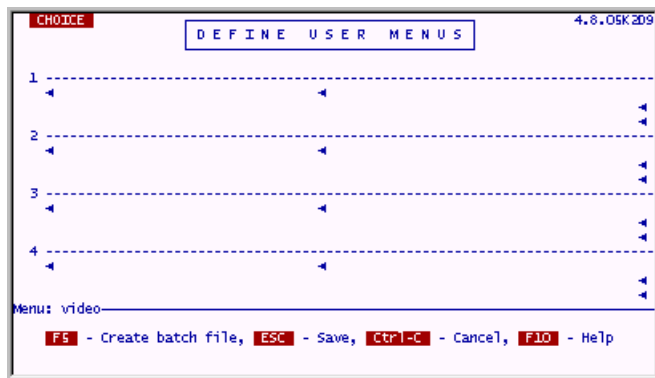


Enter the following data:



Press **ESC** to save this screen.

The following blank menu will appear:



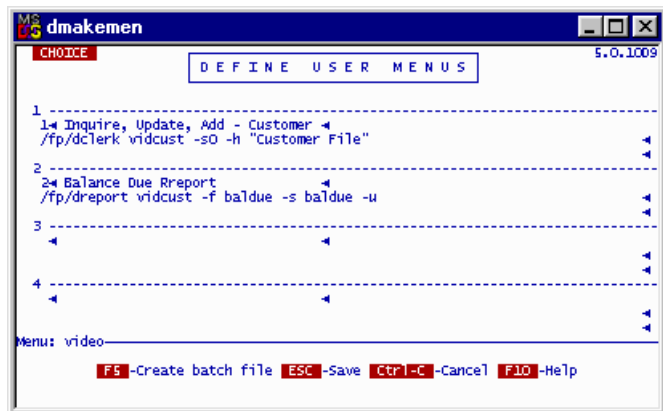
This menu will allow you to run various items in the tutorial. We will update it periodically.

On filePro menus, filePro can call its own programs with a shortened PATH. Do this by placing /fp/ in front of the filePro program you want to run, i.e., /fp/dclerk or /fp/dreport. After this put the rest of the action line that you want to execute should this choice on the menu be selected.

For choice 1, we will run IUA (dclerk) on the file "vidcust" using screen 0 (-s0) and when the action runs we want to display a heading of "Customer File" (-h "Customer File"). When the user chooses 1, this is exactly what will be run just as if they had typed it at the command line.

For choice 2, we will run Request Output (dreport) on the file "vidcust" using the output format "baldue" (-f baldue), against the records in selection set "baldue" (-s baldue), using the Unlock option (-u). Unlock is applicable on multi-user systems only (Unix & Network versions). It tells filePro to run this report even if there are other people using this filePro file. In other words, people can be using IUA on this file and the report will still run. If on the other hand, the -u is neglected, the report can only run when no one else is accessing this file. **IMPORTANT** : The -u option does not bypass "record locking". If someone is in Update Mode on any of the records selected for this report, or has one of these records locked with another process of any kind, our report will stop and wait for the record to be unlocked when it hits that record.

Enter the following data:



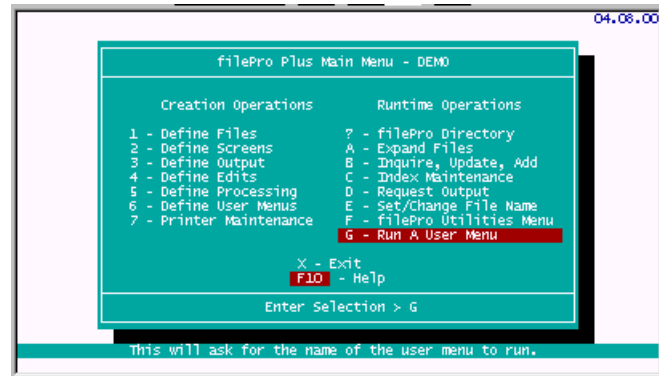
Press **ESC** to save this screen.

You will see a "Change Menu Password" prompt. Enter **N**. Do not do this at this time.

You will see a "Hardcopy Menu" prompt. This is your choice. Enter a **Y** or **N**, as you like.

Running a User Menu

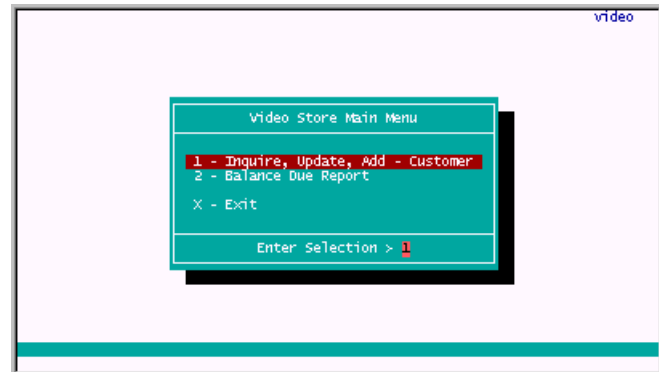
Select G - Run A User Menu.



Choose "video".



Try the choices to make sure they work.



Congratulations! You have successfully completed this section.

Obtaining a Unique Number

System Generated Numbers

Up until now, we have been adding the account number for each new record ourselves. And, just as our customers are unique, so must their account numbers be. As you might imagine, having two or more customers with the same account number would make things pretty complicated. When we manually enter the account number, we greatly increase the chances of this happening. So then, what we need is a way to automatically assign a unique number to each record. (This is often a basic database requirement for many types of records.) "filePro" can provide this function in a number of ways. The following procedures show a most reliable and flexible method.

First, the field on the screen has to be "protected". This way, the user can not modify an account number by accident. The computer will generate the next highest unique number and once generated, it will stay with this customer record forever.

From the main menu, go into Define Screens. (Incidentally, from this point forward, it will be assumed that you will execute all filePro design operations from the main menu. It will not be mentioned again in the instructions.)

Update and modify Screen 0 to look like this:

```
VIDCUST
Account_code: !1
First_Name: *2
Last_Name: *3
Address1: *4
Address2: *5
City: *6
State: *7
Zip: *8
Phone: *9

Customer_type: !11
First_sale_date: *12

Charges: *14
Payments: *15
Balance_due: !16

File: vidcust Color Screen: 0
F9 -Toggle Graphics, F6 -Display Fields, F5 -Resolve Fields
F10 -Help, F7 -Box Func's, F8 -Extended Func's
4,18 Ctrl F10 -Color Help, ESC -Record, Ctrl-C -Cancel
```

Once the account code has been protected with an "!", you can return to the main menu.

The next step in obtaining a unique number is defining a file that will hold this number. Such a control file can hold many unique numbers and other important information about the application we are designing. This will be a file that currently has only one field in it. Also, this file will only have one record, specifically record #1. The last assigned account number will be stored on record #1 of this file in field #1. Then, each time we add a new account, the program will do a lookup to this particular file and record and retrieve the number found there. It will increment this number by 1, thereby giving our new account its own unique number. This system will work until there are more than 999,999 account code's required. Not likely for our first Video Store. (Maybe once we're a nationwide chain, we can think about raising this field to a 7 digit number!)

Go into **Define Files** and create the following regular filePro file. It is called "vidctrl" because it will hold our control numbers and information.

Enter the following data:

```
DEFINE FILES 4.8.05K209
Number  Field Heading  Len  --Type--
1 - Next_cust_code  64  .0
2 -
3 -
4 -
5 -
6 -
7 -
8 -
9 -
10 -
11 -
12 -
13 -
14 -
15 -
16 -
File: vidctrl Key
Press K for key, D for data, U to update, H to hardcopy,
F10 for help, PgDn PgUp to scroll, X to exit.
Characters used: 6 Characters Left: 32694
```

When you are done, **press ESC** to save your work.

Press X to finish the file design. The options screen will appear.

Enter the following:

```
DEFINE FILES 4.8.05K209
Number  Field Heading  Len  --Type--
1 - Next_cust_code  64  .0
2 -
3 -
4 -
5 -
6 -
7 -
8 -
9 -
10 -
11 -
12 -
13 -
14 -
15 -
16 -
File: vidctrl Key

Create default formats
Create a screen 0 (Y/N): Y
Create a default report (Y/N): N
Create automatic indexes (Y/N): N
Create qualifiers (Y/N): N
Press ESC To Record, Ctrl-C To Cancel
```

Next, go into **Define Processing** for the file "vidcust".

Select **INPUT** processing.

Make sure your cursor is somewhere in the first element and **press F3** to push down all the current processing. This leaves you with a new blank element number 1.

We are only going to be getting a new account code for records that do not have one yet. If the account code field is already filled, the subroutine which gets the next unique number will not be run. The "if" condition on line 1 of this code assures this.

Enter the following:

```
LABEL          DEFINE  PROCESSING          4,8,05K409 DEMO
1  -----
  If: Account_code eq ""
  Then: gosub getnum ; display
2  -----
  If:
  Then: input qq(1,yesno) "Does everything look alright? (y/n) "
3  -----
  If: qq ne "y"
  Then: restart
4  -----
  If:
  Then: end
5  -----
@wif14 If:
  Then: gosub totals ; display ; end
6  -----
@wif15 If:
  Then: gosub totals ; display ; end
File: vidcust                                     Processing: input
1  F8 -Block Func, F9 -Go To/Find
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.
```

We will now add the subroutine that does the actual work. It is based on doing a lookup to the control file and retrieving the next number.

Enter the following and leave your cursor in element 9:

```
ACTION        DEFINE  PROCESSING          4,8,05K409 DEMO
7  -----
totals If:
  Then: Balance_due = Charges - Payments ; return
8  -----
  If:
  Then: aa(1,,0)="1"
9  -----
  If:
  Then:
10 -----
  If:
  Then:
11 -----
  If:
  Then:
12 -----
  If:
  Then:
File: vidcust                                     Processing: input
1  F8 -Block Func, F9 -Go To/Find, F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.
```

To define the lookup, **press F5** while your cursor is on line 9. The lookup editor will popup. Enter the name of the file we will be looking up, "vidctrl".

```
DEFINE LOOKUP DEMO
Lookup From File: vidctrl
Name Of Lookup Is:
Press Ctrl-C To Cancel Changes
```

Enter an "r" for a record number lookup. (We are going to be looking up to record number 1.)

```
DEFINE LOOKUP DEMO
Lookup From File: vidctrl
Name Of Lookup Is:
How Is The Record To Be Found? r K - Key Field R - Record Number
F - Free Record Z - Fuzzy Search
What Field In 'vidcust' Contains The Record Number?
1- Account_code 7- State 13- r
2- First_Name 8- Zip 14- Charges
3- Last_Name 9- Phone 15- Payments
4- Address1 10- r 16- Balance_due
5- Address2 11- Customer_type
6- City 12- First_sale_date
Press F5 For Field Lengths and Edits
Press Ctrl-C To Cancel Changes
```

Enter "aa" as the field in "vidcust" that contains the record number. (We will fill the dummy field "aa" with a 1 so that we can lookup to this particular record.

It is also **VERY IMPORTANT** to **put a "Y"** in the "Protect Record in Lookup File?" prompt. This is only necessary on multi-user systems, but it is **VERY** necessary. It tells filePro not to let anyone else retrieve a unique number out of this record at the same time we are doing so. It will prevent duplicate account codes from being issued. Protecting the record means "locking the record" so no other process can read it while we have it locked.

Enter the data as shown:

```

DEFINE LOOKUP DEMO
Lookup From File: vidctrl
Name Of Lookup Is:
How Is The Record To Be Found? r K - Key Field R - Record Number
F - Free Record Z - Fuzzy Search
What Field In 'vidctrl' aa
Contains The Record Number?
Protect Record in Lookup File? y
If Lookup Fails: n B - Blank The Field
N - Do Nothing
E - Report An Error
Create Browse Lookup? n
Press Ctrl-C To Cancel Changes
    
```

If Lookup Fails: In other words if record number 1 in the "vidctrl" file can not be opened, what should filePro do? We will choose to have filePro do nothing in this case. If the lookup fails, we will take our own action to explain the situation to the user. (If we had chosen "E" - Report an Error, filePro would put up a standard "lookup failed" message. If we had chosen "B" - Blank the Field, filePro would have blanked our lookup retrieval field. Neither are very valuable options, and quite generally, you should always handle lookup errors yourself.)

To complete the lookup, answer N to the "Create Browse Lookup" prompt.

When you are automatically returned to the processing table, you will see that filePro has created a lookup line for you that looks like this.

```

ACTION DEFINE PROCESSING 4.8.05K409 DEMO
7 -----
totals < If:
Then: Balance_due = Charges - Payments ; return
8 -----
< If:
Then: aa(1,.0)="1"
9 -----
< If:
Then: lookup vidctrl r=aa -np
10 -----
< If:
Then:
11 -----
< If:
Then:
12 -----
< If:
Then:
File: vidcust Processing: input
1 F8 -Block Func, F9 -Go To/Find, F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.
    
```

Finish the code for this subroutine by adding the final three lines.

Line 10 handles the error if the record can not be found. Line 11 retrieves the value of field 1 in the "vidctrl" file and assigns it to the Account_code field in our current file ("vidcust"). Then, line 11 increments the value of field 1 in the "vidctrl" file. The WRITE command on this line ensures that the file will be updated on the disk as soon as the O/S can do it. Line 12 returns the processing back to the place it was called from on Line 1 (gosub getnum)..

```

LABEL DEFINE PROCESSING 4.8.05K409 DEMO
7 -----
totals < If:
Then: Balance_due = Charges - Payments ; return
8 -----
getnum < If:
Then: aa(1,.0)="1"
9 -----
< If:
Then: lookup vidctrl r=aa -np
10 -----
< If: not vidctrl
Then: show "Serious Problem! No Control file!";return
11 -----
< If:
Then: Account_code=vidctrl(1);vidctrl(1)=vidctrl(1)+1 ; write vidctrl
12 -----
< If:
Then: return
File: vidcust Processing: input
1 F8 -Block Func, F9 -Go To/Find
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.
    
```

Press ESC to save your work on this processing table.

You will see a prompt for "Check Syntax? (Y/N)". Enter Y, and fix any problems found by the syntax check.

Hardcopy the processing if you like when prompted. (Still, cross reference hardcopy is optional as well.)

```

DEFINE PROCESSING 4.8.05K409 DEMO
Check Syntax? (Y/N) Y
Token Table Size: 3700
Hardcopy? (Y/S/N) N
Cross Reference Hardcopy? (Y/N)
    
```

Now we must go into the "vidctrl" file through IUA and tell our program which unique number (Account_code) we want to start with.

Go into Inquire, Update, Add (IUA).

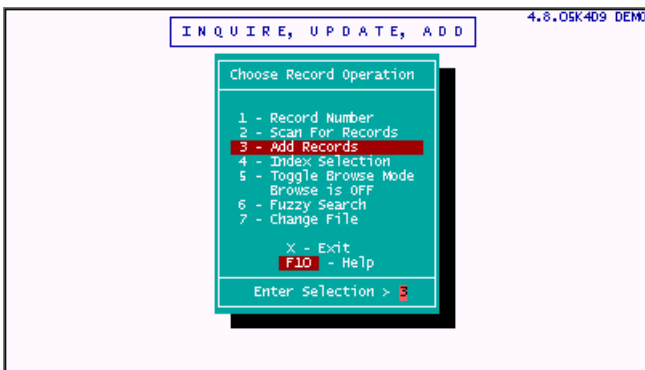
Choose "vidctrl"



Choose Screen.0

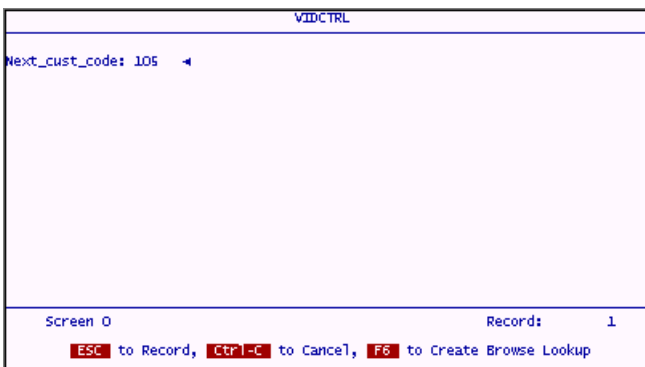


Select 3 - Add Records.



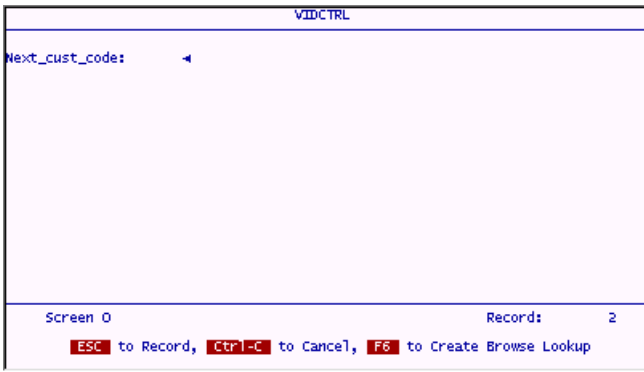
We have manually assigned 4 account numbers already, the highest of which was 104. We can start the next account number at 105.

Enter the following data:



Press ESC to save this record.

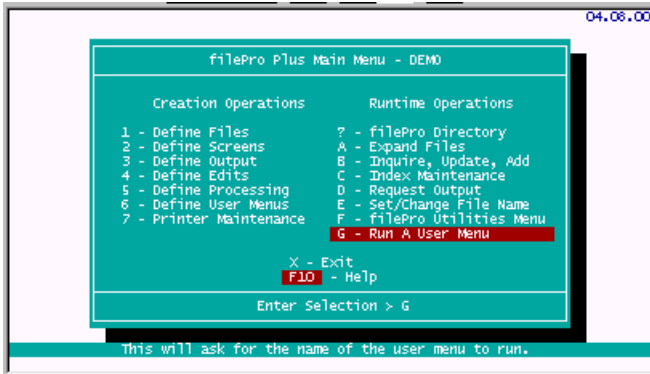
Since you selected Add Records Mode, filePro will present you with another blank record and wait for you to fill it. Do not enter anything on this record. **Press Control-C** to stop adding records.



Press X twice to return to the main menu.

We are ready to try the new process. We can use the menu we designed earlier to do this.

Select G - Run A User Menu.

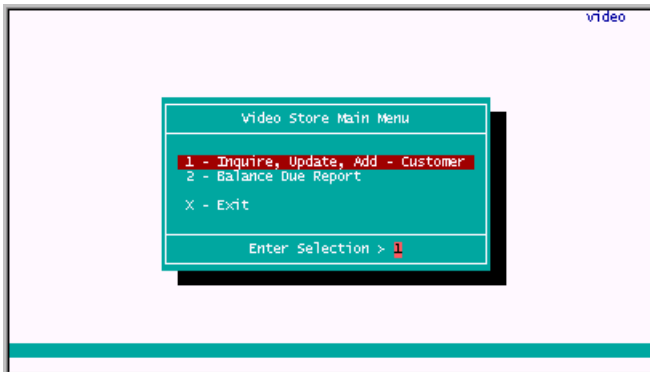


Choose "video".



The "Video Store Main Menu" will appear.

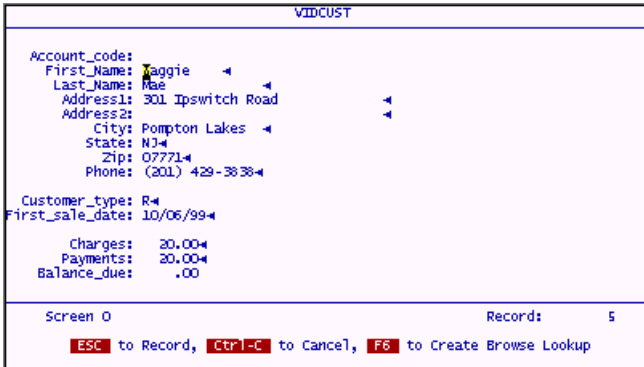
Select 1 - Inquire, Update, Add - Customer.



Select 3 - Add Records.



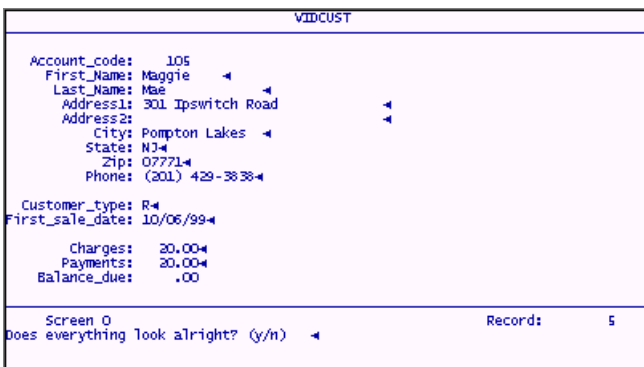
Add the following data:



Press **ESC** to save your work.

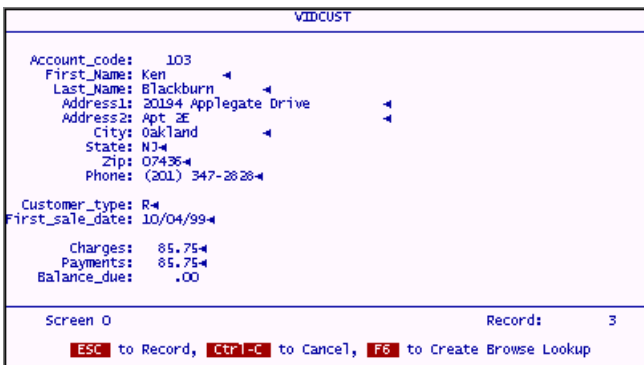
When the "blank" screen comes up, **BREAK** out with **Control-C**. Then go back to record #5 (pressing the UP Arrow should work to do this.)

You will see that the next Account_code of 105 has been properly added automatically by the "getnum" subroutine.



Leave this record and bring up record number 3.

Press **U** to update this record, but don't change anything.



Press **ESC** after you enter the Update Mode. You will see that the Account_code does NOT get changed. This is exactly the operation we want. New accounts (those with no Account_code) get new numbers, old accounts (those already having a code) stay as they are.

Tracking Receipts By Customer Account#

We are now going to define another file for the application. We need a file to track video rentals and the receipt of money for those rentals.

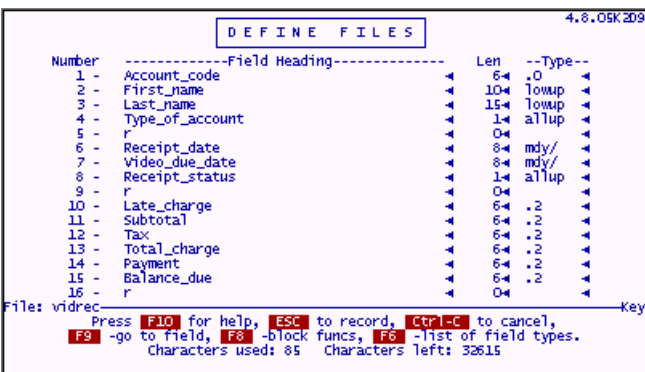
Go into Define Files , choose [NEW].

Enter the following name "vidrec".



Press Enter to continue. Do not give this file a Creation Password.

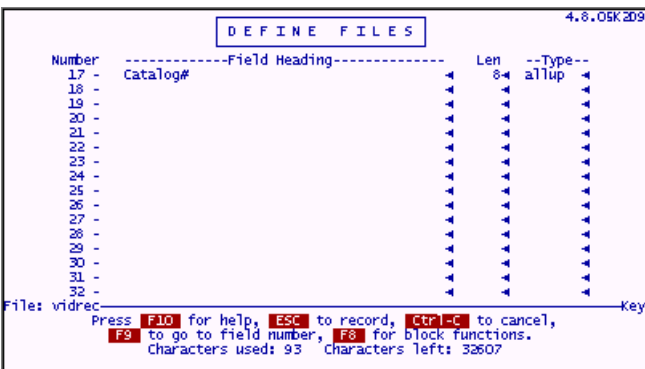
Enter the following data:



When you are done entering this data, press the NextPage key and put your cursor on field 17.

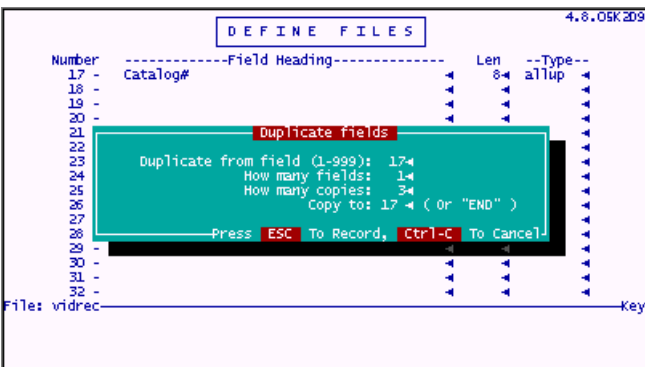
We are going to build 4 lines of detail for each record in this file. Since these lines are comprised of the same fields duplicated 4 times, the following procedure shows how to use the Block Function inside Define Files to quickly create duplicate fields.

Enter the following data, then return your cursor to line 17.



Press F8 to bring up the Block Functions dialog box. There is currently only 1 block function, it is duplicate fields.

Enter the following data:



Press **ESC** to record your work.

The system will build the desired duplicate fields as shown.

```
4.8.05K209
DEFINE FILES
Number  -----Field Heading-----  Len  --Type--
17 -   Catalog#                          8    allup
18 -   Catalog#                          8    allup
19 -   Catalog#                          8    allup
20 -   Catalog#                          8    allup
21 -
22 -
23 -
24 -
25 -
26 -
27 -
28 -
29 -
30 -
31 -
32 -
File: vidrec                                     Key
Press F10 for help, ESC to record, Ctrl-C to cancel,
F9 to go to field number, F8 for block functions.
Characters used: 117 Characters left: 32583
```

We will build the other parts of these 4 lines in the same manner.

Enter the following data on line 21. Put your cursor back on line 21 when you are done.

```
4.8.05K209
DEFINE FILES
Number  -----Field Heading-----  Len  --Type--
17 -   Catalog#                          8    allup
18 -   Catalog#                          8    allup
19 -   Catalog#                          8    allup
20 -   Catalog#                          8    allup
21 -   Description                        40    '
22 -
23 -
24 -
25 -
26 -
27 -
28 -
29 -
30 -
31 -
32 -
File: vidrec                                     Key
Press F10 for help, ESC to record, Ctrl-C to cancel,
F9 to go to field number, F8 for block functions.
Characters used: 157 Characters left: 32543
```

Press **F8** to bring up the Duplicate Fields dialog box again.

The box should be ready to go as is. Make sure it looks like the following.

```
4.8.05K209
DEFINE FILES
Number  -----Field Heading-----  Len  --Type--
17 -   Catalog#                          8    allup
18 -   Catalog#                          8    allup
19 -   Catalog#                          8    allup
20 -   Catalog#                          8    allup
21 -
22 -
23 -
24 -
25 -
26 -
27 -
28 -
29 -
30 -
31 -
32 -
File: vidrec                                     Key
Duplicate fields
Duplicate from field (1-999): 21
How many fields: 1
How many copies: 3
Copy to: 21 ( Or "END" )
Press ESC To Record, Ctrl-C To Cancel
```

Press **ESC** when you are done.

filePro adds the requested fields.

```
4.8.05K209
DEFINE FILES
Number  -----Field Heading-----  Len  --Type--
17 -   Catalog#                          8    allup
18 -   Catalog#                          8    allup
19 -   Catalog#                          8    allup
20 -   Catalog#                          8    allup
21 -   Description                        40    '
22 -   Description                        40    '
23 -   Description                        40    '
24 -   Description                        40    '
25 -
26 -
27 -
28 -
29 -
30 -
31 -
32 -
File: vidrec                                     Key
Press F10 for help, ESC to record, Ctrl-C to cancel,
F9 to go to field number, F8 for block functions.
Characters used: 277 Characters left: 32423
```

Follow the same procedure to make 4 fields for "Charges". Enter the first field on line 25 and then duplicate it 3 times.

Your screen should look like this when you are done:

```

4.8.05K209
  DEFINE FILES
Number  -----Field Heading-----  Len  --Type--
17 - Catalog#                          8  allup
18 - Catalog#                          8  allup
19 - Catalog#                          8  allup
20 - Catalog#                          8  allup
21 - Description                       40  '
22 - Description                       40  '
23 - Description                       40  '
24 - Description                       40  '
25 - Charge                            6  .2
26 - Charge                            6  .2
27 - Charge                            6  .2
28 - Charge                            6  .2
29 -
30 -
31 -
32 -
File: vidrec                                     Key
Press F10 for help, ESC to record, Ctrl-C to cancel,
F9 to go to field number, F8 for block functions.
Characters used: 301 Characters left: 32399

```

Press ESC to save your work.

Press X to finish the design of this file.

Enter the data as shown below:

```

4.8.05K209
  DEFINE FILES
Number  -----Field Heading-----  Len  --Type--
17 - Catalog#                          8  allup
18 - Catalog#                          8  allup
19 - Catalog#                          8  allup
20 - Catalog#                          8  allup
21 - Des
22 - Des
23 - Des
24 - Des
25 - Cha
26 - Cha
27 - Cha
28 - Cha
29 -
30 -
31 -
32 -
File: vidrec                                     Key
Create default formats
Create a screen 0 (Y/N): y
Create a default report (Y/N): N
Create automatic indexes (Y/N): N
Create qualifiers (Y/N): N
Press ESC To Record, Ctrl-C To Cancel

```

Note that filePro will warn you that only the first 20 fields could fit on the default screen. This is okay, we are going to discuss this next.

```

4.8.05K209
  DEFINE FILES
Number  -----Field Heading-----  Len  --Type--
17 - Catalog#                          8  allup
18 - Catalog#                          8  allup
19 - Catalog#                          8  allup
20 - Catalog#                          8  allup
21 - Description                       40  '
22 - Description                       40  '
23 - Description                       40  '
24 - Description                       40  '
25 - Charge                            6  .2
26 - Charge                            6  .2
27 - Charge                            6  .2
28 - Charge                            6  .2
29 -
30 -
31 -
32 -
File: vidrec                                     Key
NOTE: Only The First 20 Fields Could Fit On The Default Screen
Press ← To Continue

```

Go to Define Screens for the file "vidrec".

Choose Screen.0

```

VIDREC
Account_code: *1          Catalog#: *18
First_name: *2           Catalog#: *19
Last_name: *3            Catalog#: *20
Type_of_account: *4
r: *5
Receipt_date: *6
Video_due_date: *7
Receipt_status: *8
r: *9
Late_charge: *10
Subtotal: *11
Tax: *12
Total_charge: *13
Payment: *14
Balance_due: *15
r: *16
Catalog#: *17
File: vidrec                                     Color Screen: 0
Enter Selection >
F10 -Help, U -Update, S -Select Screen, F -Select File,
C -Copy Screen, H -Hardcopy, D -Delete Screens, X -Exit

```

Press C to copy this screen to a different name.

```

VIDREC

Account_code: *1          Catalog#: *18
First_name: *2           Catalog#: *19
Last_name: *3            Catalog#: *20
Type_of_account: *4
r: *5
Receipt_date: *6
Video_due_date: *7
Receipt_status: *8
r: *9
Late_charge: *10
Subtotal: *11
Tax: *12
Total_charge: *13
Payment: *14
Balance_due: *15
r: *16
Catalog#: *17
File: vidrec-----Color Screen: 0

Copy to...
Enter screen name:

```

Choose [NEW]

Enter the name 1.

```

VIDREC

Account_code: *1          Catalog#: *18
First_name: *2           Catalog#: *19
Last_name: *3            Catalog#: *20
Type_of_account: *4
r: *5
Receipt_date: *6
Video_due_date: *7
Receipt_status: *8
r: *9
Late_charge: *10
Subtotal: *11
Tax: *12
Total_charge: *13
Payment: *14
Balance_due: *15
r: *16
Catalog#: *17
File: vidrec-----Color Screen: 0

Copy to...
Enter screen name: 1

```

```

VIDREC

Account_code: *1          Catalog#: *18
First_name: *2           Catalog#: *19
Last_name: *3            Catalog#: *20
Type_of_account: *4
r: *5
Receipt_date: *6
Video_due_date: *7
Receipt_status: *8
r: *9
Late_charge: *10
Subtotal: *11
Tax: *12
Total_charge: *13
Payment: *14
Balance_due: *15
r: *16
Catalog#: *17
File: vidrec-----Color Screen: 1

Enter Selection >
F10 -Help, U -Update, S -Select Screen, F -Select File,
C -Copy Screen, H -Hardcopy, D -Delete Screens, X -Exit

```

Note that you will now be looking at, or standing "on", Screen 1.

There is no need to keep Screen 0 around anymore. We will delete it so it will not be accidentally chosen by our users.

To do this, **press D**. Select Screen 0, and **press ENTER** to mark it with a star. Pressing ENTER will unmark it also, this is a toggle. Once you have marked it, **press ESC** to actually delete it. You will be prompted for acknowledgment before the delete occurs.

```

VIDREC

Account_code: *1          Catalog#: *18
First_name: *2           Catalog#: *19
Last_name: *3            Catalog#: *20
Type_of_account: *4
r: *5
Receipt_date: *6
Video_due_date: *7
Receipt_status: *8
r: *9
Late_charge: *10
Subtotal: *11
Tax: *12
Total_charge: *13
Payment: *14
Balance_due: *15
r: *16
Catalog#: *17
File: vidrec-----Color Screen: 1

Are you sure you want to delete the selected screen(s)?
Press Y or N

*0
1
Delete screens...
Enter screen name:
ESC - delete marked screens, Ctrl-C - cancel.

```

We need to modify the screen to fit our needs. There are many functions inside filePro to help do this.

Press "U" for Update, put your cursor on line 1 of the screen and **press F4 twice**. This will pull up the lines of the screen to cover the default title and title graphic line.

Your screen should now look like this:

```

Account_code: *1          Catalog#: *18
First_name: *2           Catalog#: *19
Last_name: *3           Catalog#: *20
Type_of_account: *4
r: *5
Receipt_date: *6
Video_due_date: *7
Receipt_status: *8
r: *9
Late_charge: *10
Subtotal: *11
Tax: *12
Total_charge: *13
Payment: *14
Balance_due: *15
r: *16
Catalog#: *17

File: vidrec-----Color Screen: 1
F9 -Toggle Graphics, F6 -Display Fields, F5 -Resolve Fields
F10 -Help, F7 -Box Func's, F8 - Extended Func's
1, 1 Ctrl F10 -Color Help, ESC -Record, Ctrl-C -Cancel

```

Here is how to access the BoxFunctions which aid in screen design.

Put your cursor on position 11,4 . (Position of cursor is shown at bottom left corner of the screen.), and **press F7** .

```

Account_code: *1          Catalog#: *18
First_name: *2           Catalog#: *19
Last_name: *3           Catalog#: *20
Type_of_account: *4
r: *5
Receipt_date: *6
Video_due_date: *7
Receipt_status: *8
r: *9
Late_charge: *10
Subtotal: *11
Tax: *12
Total_charge: *13
Payment: *14
Balance_due: *15
r: *16
Catalog#: *17

File: vidrec-----Color Screen: 1
O -Draw Box, E -Erase Box, B -Blank Box, I -Invert Box
C -Copy Box, M -Move Box, O -Overlay Box, S -Save Box
11, 4 F -Color Fill, W -Fill And Drop Shadow, X -Exit, F10 -Help

```

A cross hair cursor will appear where your normal cursor used to be.

Now move your cursor to position 16,24 (the cross hair will not move). When your cursor is at 16,24 **press M** .

```

Account_code: *1          Catalog#: *18
First_name: *2           Catalog#: *19
Last_name: *3           Catalog#: *20
Type_of_account: *4
r: *5
Receipt_date: *6
Video_due_date: *7
Receipt_status: *8
r: *9
Late_charge: *10
Subtotal: *11
Tax: *12
Total_charge: *13
Payment: *14
Balance_due: *15
r: *16
Catalog#: *17

File: vidrec-----Color Screen: 1
Use the arrow keys to move the box where you want it.
H - Center horizontally, V - Center vertically, ← - Move box
11, 4

```

A box of "comer graphics" will be outlined from your starting position to your ending position. You can now move this area of the screen anywhere it will fit and press **ENTER** . This will literally move this section of the screen to the new position.

Use the Arrow keys to move this section of the screen to position 14,51 . (That is, put the top left corner of the box on position 14,51.)

```

Account_code: *1          Catalog#: *18
First_name: *2           Catalog#: *19
Last_name: *3           Catalog#: *20
Type_of_account: *4
r: *5
Receipt_date: *6
Video_due_date: *7
Receipt_status: *8
r: *9
Late_charge: *10
Subtotal: *11
Tax: *12
Total_charge: *13          r          1
Payment: *14
Balance_due: *15
r: *16
Catalog#: *17          L          J

File: vidrec-----Color Screen: 1
Use the arrow keys to move the box where you want it.
H - Center horizontally, V - Center vertically, ← - Move box
14, 51

```

When you're there, **press ENTER** .

The move operation will leave your screen looking like this.

```

Account_code: *1      Catalog#: *18
First_name: *2       Catalog#: *19
Last_name: *3        Catalog#: *20
Type_of_account: *4
r: *5
Receipt_date: *6
Video_due_date: *7
Receipt_status: *8
r: *9

Late_charge: *10
Subtotal: *11
Tax: *12
Total_charge: *13
Payment: *14
Balance_due: *15

r: *16
Catalog#: *17

File: vidrec-----Color Screen: 1
F9 -Toggle Graphics, F6 -Display Fields, F5 -Resolve Fields
F10 -Help, F7 -Box Func's, F8 -Extended Func's
14,51 Ctrl F10 -Color Help, ESC -Record, Ctrl-C -Cancel

```

There are several options for the Box Functions. A useful one is "blanking an area". To do this, select box functions by putting your cursor on position 1,33 , then pressing F7 . The cross hair will appear.

```

Account_code: *1      Catalog#: *18
First_name: *2       Catalog#: *19
Last_name: *3        Catalog#: *20
Type_of_account: *4
r: *5
Receipt_date: *6
Video_due_date: *7
Receipt_status: *8
r: *9

Late_charge: *10
Subtotal: *11
Tax: *12
Total_charge: *13
Payment: *14
Balance_due: *15

r: *16
Catalog#: *17

File: vidrec-----Color Screen: 1
D -Draw Box, E -Erase Box, B -Blank Box, I -Invert Box
C -Copy Box, M -Move Box, O -Overlay Box, S -Save Box
5,51 F -Color Fill, W -Fill And Drop Shadow, X -Exit, F10 -Help

```

While the cross hair is visible, put your cursor at 5,51 and then press B .

That area of the screen will be blanked (cleared).

```

Account_code: *1      Catalog#: *18
First_name: *2       Catalog#: *19
Last_name: *3        Catalog#: *20
Type_of_account: *4
r: *5
Receipt_date: *6
Video_due_date: *7
Receipt_status: *8
r: *9

Late_charge: *10
Subtotal: *11
Tax: *12
Total_charge: *13
Payment: *14
Balance_due: *15

r: *16
Catalog#: *17

File: vidrec-----Color Screen: 1
F9 -Toggle Graphics, F6 -Display Fields, F5 -Resolve Fields
F10 -Help, F7 -Box Func's, F8 -Extended Func's
5,51 Ctrl F10 -Color Help, ESC -Record, Ctrl-C -Cancel

```

Using these box functions and any other trick at your command, make your Screen 1 look like the screen shown below. You will need to use Reverse Video for the line item headings and for the protected field indicator of field 15. Do this with Alt-F1 (to tum it on) and Alt-F8 to tum it off. (HINT : The straight line is nothing but a very skinny box! Use the D option to draw this line.)

```

Account_code: *1      Type: *14      Address: *aa
First_name: *2       : *ab
Last_name: *3        : *ac
Phone: *ad

Catalog#  Description  Charge
*17      *21                *25
*18      *22                *26
*19      *23                *27
*20      *24                *28
Late_charge: *10
Subtotal: *11
Tax: *12
Total_charge: *13
Payment: *14
Balance_due: *15

Receipt_date: *6
Video_due_date: *7
Receipt_status: *8

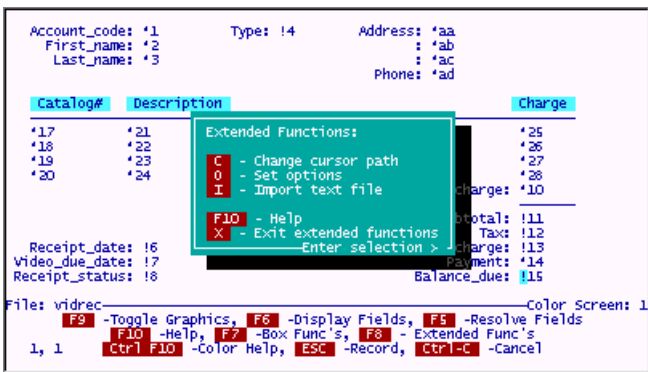
File: vidrec-----Color Screen: 1
F9 -Toggle Graphics, F6 -Display Fields, F5 -Resolve Fields
F10 -Help, F7 -Box Func's, F8 -Extended Func's
20,80 Ctrl F10 -Color Help, ESC -Record, Ctrl-C -Cancel

```

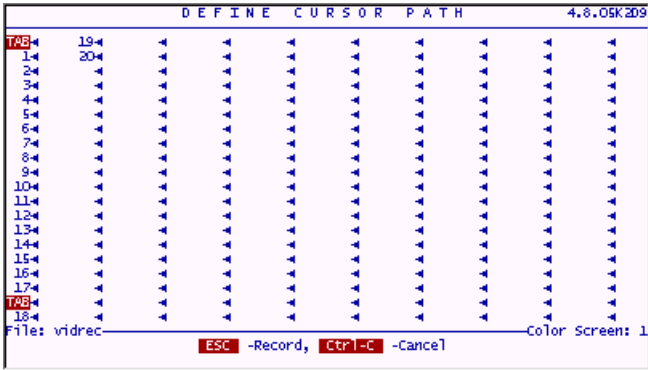
When you are done, press ESC to save your work.

Since we have modified the screen from the way it looked when we first copied it, we must rebuild the cursor path. This is the roadmap that indicates how we want the cursor to move through these fields on the users screen while they are in IUA.

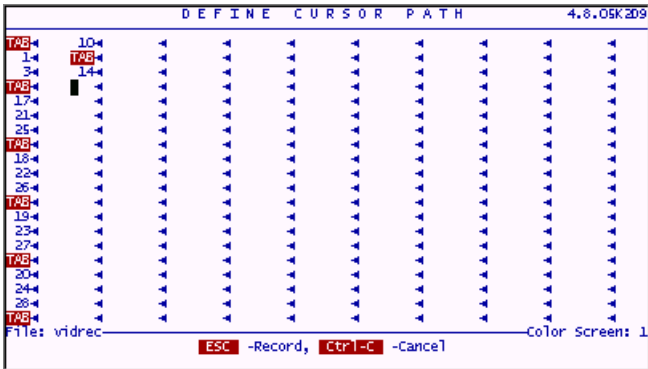
Press F8 to bring up the Extended Functions dialog box.



Press C for Cursor Path and the following screen will appear:



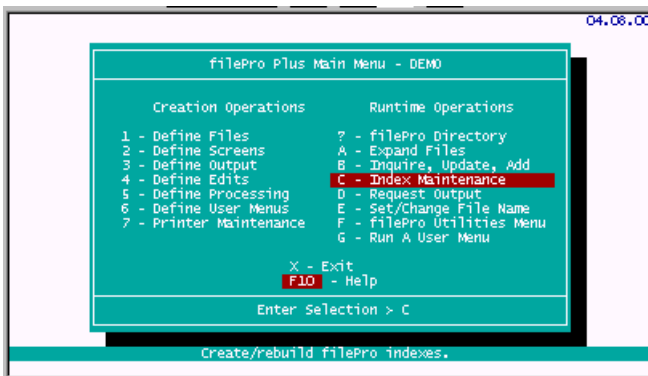
Correct the cursor path as follows. Use the F3 and F4 keys to push fields up and down on this screen. Wherever there is a TAB indicated, press the TAB key, and filePro will insert the word TAB for you.



Press ESC to save your work on the cursor path. Press ESC on the main screen to save the entire screen.

To access this file, we will need to build some automatic indexes for it.

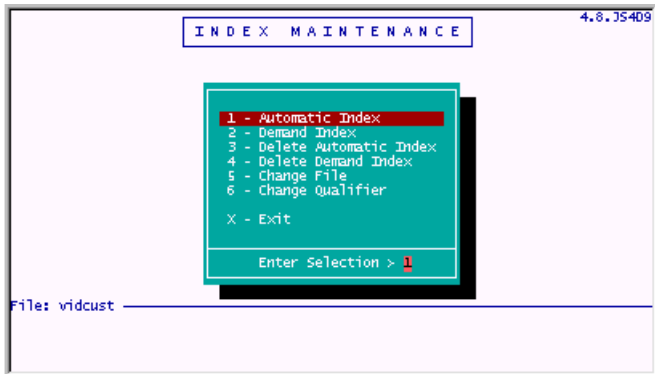
Select C - Index Maintenance.



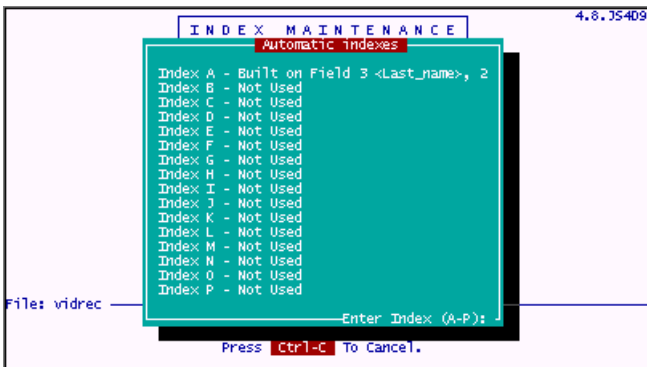
Choose "vidcust".



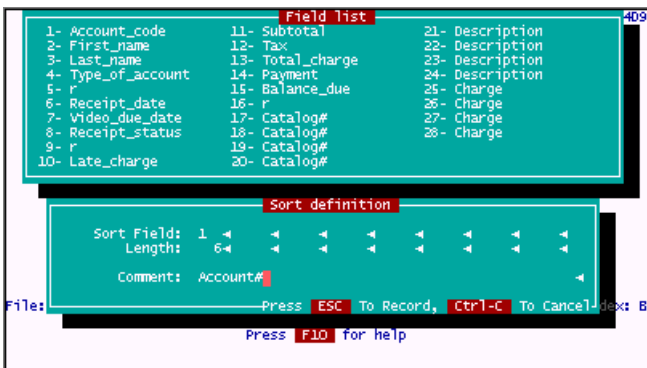
Select 1 - Automatic Index.



Select B.



Enter the following data:

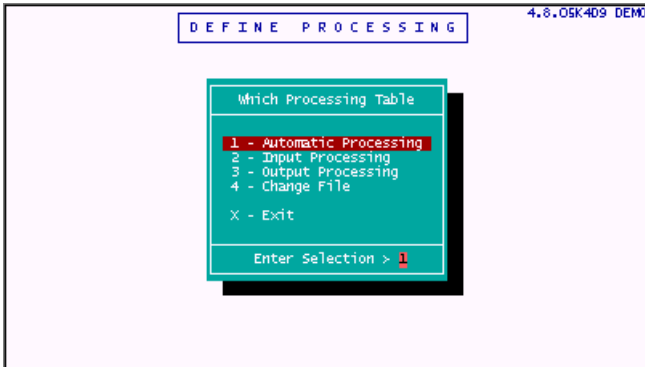


It's time to build some processing for the new file we've created, "vidrec". This file will hold receipts that will be "attached" to the customer file via the unique Account_code field. This means each time a new invoice is added, the Account_code (Account#) of the customer will be placed on the receipt record. Also, the customer's name and address can be pulled up automatically from the receipts file by using a lookup based on either the Account# or sometimes the Account Name.

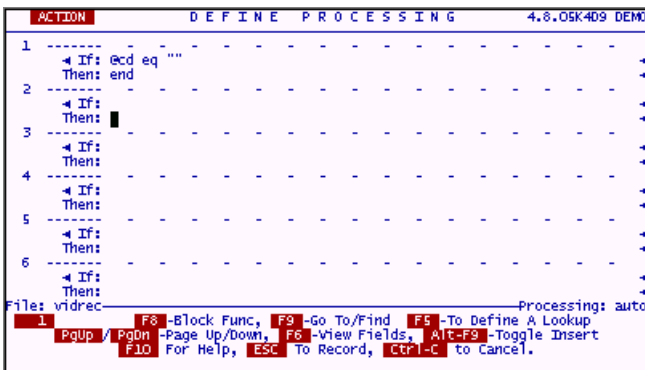
There will be several types of processing in the "vidrec" file; Automatic, Input, Trigger, and Output. We will start with Automatic. **Go into Define Processing** on the "vidrec" file.



Select 1- Automatic Processing.



First, enter line 1 as shown. Then, put your cursor on line 2 so we can build a lookup using the lookup wizard.



Press F5 to start the wizard.

Fill in the following data. This time we are going to give the lookup file name a short alias. In other words, we want to be able to refer to this file as a different name, but it still means this file. The alias we will use is "addr".



Here is the complete lookup. It is a Key field lookup. In other words we will find records in the lookup file based on a key field in this file (either a real field or a dummy field.) In this case, a real field, 1, is being used as the key.

Enter the data as shown on the next three screens. They show the lookup being built step-by-step.

```

DEFINE LOOKUP DEMO
Lookup From File: vidcust
Name Of Lookup Is: addr
How Is The Record To Be Found? k K - Key Field R - Record Number
F - Free Record Z - Fuzzy Search
What Index Is To Be Used? b F6 for list
What Field In 'vidrec' 1
Contains The Key?
Protect Record in Lookup File? N
1- Account_code 7- Video_due_date 13- Total_charge
2- First_name 8- Receipt_status 14- Payment
3- Last_name 9- r 15- Balance_due
4- Type_of_account 10- Late_charge 16- r
5- r 11- Subtotal 17- Catalog#
6- Receipt_date 12- Tax 18- Catalog#
Press F6 For Field Lengths and Edits
Press Ctrl-C To Cancel Changes

```

```

DEFINE LOOKUP DEMO
Lookup From File: vidcust
Name Of Lookup Is: addr
How Is The Record To Be Found? k K - Key Field R - Record Number
F - Free Record Z - Fuzzy Search
What Index Is To Be Used? b F6 for list
What Field In 'vidrec' 1
Contains The Key?
Protect Record in Lookup File? N
If Key Doesn't Match Exactly: X X - Key Must Match Exactly
G - Use Next Greater Match
L - Use Next Lower Match
Press Ctrl-C To Cancel Changes

```

```

DEFINE LOOKUP DEMO
Lookup From File: vidcust
Name Of Lookup Is: addr
How Is The Record To Be Found? k K - Key Field R - Record Number
F - Free Record Z - Fuzzy Search
What Index Is To Be Used? b F6 for list
What Field In 'vidrec' 1
Contains The Key?
Protect Record in Lookup File? N
If Key Doesn't Match Exactly: X X - Key Must Match Exactly
G - Use Next Greater Match
L - Use Next Lower Match
If Lookup Fails: n B - Blank The Field
N - Do Nothing
E - Report An Error
Create Browse Lookup? N
Press Ctrl-C To Cancel Changes

```

The final lookup line as constructed by the lookup wizard is as follows. As you can see, it is fairly simple and straightforward. As you get more comfortable with how lookups work, you may choose to simply type lookup lines in without using the lookup wizard.

```

ACTION DEFINE PROCESSING 4.8.05K409 DEMO
1 -----
  If: @cd eq ""
  Then: end
2 -----
  If:
  Then: lookup addr = vidcust k=l i=b -rx
3 -----
  If:
  Then:
4 -----
  If:
  Then:
5 -----
  If:
  Then:
6 -----
  If:
  Then:
File: vidrec Processing: auto
1 F6 -Block Func, F9 -Go To/Find, F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F8 -View Fields, Alt-F8 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

Once the lookup is performed, we can assign the values from the looked-up record to fields in this record. We will assign the values for address and phone number to "dummy fields". These dummy fields can be displayed on the screen without the need of storing them in the file. After all, the customer's address and phone are stored permanently in the "vidcust" file. Since we can look them up and use them any time we want, why store them in the computer on every single receipt record? It would be a waste of space. We will store the customer name in this file, since it will be valuable to be able to search the receipts file by name and to do this, the name must be stored on each record. But, we will never search this receipt file by address, or other even less important criteria.

The "not addr" on line 3 tells filePro what to do if the Account_code being looked for is not found in the "vidcust" file.

Make sure your table looks like this.

```

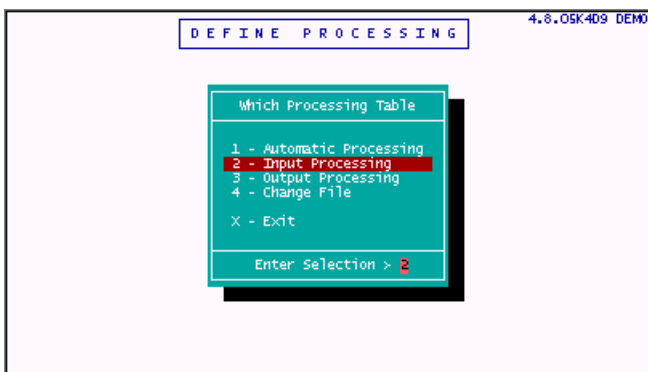
LABEL          DEFINE PROCESSING          4.8.05K409 DEMO
1 -----
  If: @cd eq ""
  Then: end
2 -----
  If:
  Then: lookup addr = vidcust k=1 i=b -nx
3 -----
  If: not addr 'refer to this file as the alias"name" addr, not vidrec
  Then: show "@Customer code not on file!" ; end
4 -----
  If: 'fill in the dummy variables on the screen with address
  Then: aa=addr (4) ; ab=addr (5) ; ac=addr (6) { " " < addr (7) < addr (8)
5 -----
  If: 'fill in the dummy variable for phone
  Then: ad=addr (9)
6 -----
  If: 'without this display, the user would not see these assignments
  Then: display ; end
File: vidrec                                     Processing: auto
1 F8 -Block Func, F9 -Go To/Find
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

Press ESC to save your work.

Next, we will enter some simple INPUT processing for the receipts file. Later on, we will elaborate on this processing, but for now all it needs to do is lookup the Account code and address of specified customers. This INPUT table will be almost an exact duplicate of the AUTO table. The lookup is required on both tables for the following reason. The AUTO table will perform the lookup for any record that is already on file. If the user is looking at full screen views of the receipt file, as each record comes up on the screen, the AUTO table will lookup the address and phone number that is associated with the Account # on the receipt and display it in dummy fields on the screen. The INPUT table will do the very same thing for new receipts as they are being entered. In fact, the lookup to retrieve address and phone# will be done when the user enters a valid Account #.

Select 2



Since the INPUT and AUTO table have very similar code to perform the lookup we are writing, we can grab the code from the AUTO table in one fell swoop and then just modify it slightly, rather than typing in all that code over again. Here is how to use the Block Functions to do this.

First, enter this much data on the screen.

```

ACTION          DEFINE PROCESSING          4.8.05K409 DEMO
1 -----
  If:
  Then: end
2 -----
  If:
  Then:
3 -----
  If:
  Then:
4 -----
  If:
  Then:
5 -----
  If:
  Then:
6 -----
  If:
  Then:
File: vidrec                                     Processing: input
1 F8 -Block Func, F9 -Go To/Find, F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

Put you cursor somewhere in element 3 and press F8 to call up the Block Functions. The screen will look like this.

```

ACTION          DEFINE PROCESSING          4.8.05K409 DEMO
1 -----
  If:
  Then: end
2 -----
  If:
  Then:
3 -----
  If:
  Then:
4 -----
  If:
  Then:
5 -----
  If:
  Then:
6 -----
  If:
  Then:
File: vidrec                                     Processing: input
F8 To Define Block, C To Copy, M To Move, H To Hardcopy
D To Delete, L To Load, S To Save
Press Ctrl-C To Cancel Block Function

```

Press L to Load some processing from another table into this table. Follow along with the pictures and enter rest of this operation as shown.

```

ACTION          DEFINE  PROCESSING          4.8.05K409 DEMO
-----
1  -----
   < If:
   Then: end
2  -----
ewif1 < If:
   Then:
3  -----
   < If:
   Then:
4  -----
   < If:
   Then:
5  -----
   < If:
   Then:
6  -----
   < If:
   Then:
File: vidrec-----Processing: input
Load To Line: 3
Enter Processing Name: auto

```

filePro will load a copy of the AUTO processing table starting at line 3.

```

ACTION          DEFINE  PROCESSING          4.8.05K409 DEMO
-----
1  -----
   < If:
   Then: end
2  -----
ewif1 < If:
   Then:
3  -----
   < If: @cd eq ""      'If the record is new (unsaved) END
   Then: end
4  -----
   < If:
   Then: lookup addr = vidcust k=1  i=b -nx
5  -----
   < If: not addr      'refer to this file as the alias"name" addr, not vidrec
   Then: show "@Customer code not on file!"; end
6  -----
   < If: 'fill in the dummy variables on the screen with address
   Then: aa=addr(4) ; ab=addr(5) ; ac=addr(6) { " , " < addr(7) < addr(8)
File: vidrec-----Processing: input
66  -----
   F8 -Block Func, F9 -Go To/Find
   PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
   F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

The AUTO table is very similar but not exactly what needs to be done on the INPUT table. Match your table to the screens following.

```

LABEL          DEFINE  PROCESSING          4.8.05K409 DEMO
-----
1  -----
   < If:
   Then: end
2  -----
ewif1 < If:
   Then:
3  -----
   < If: l eq ""
   Then: end
4  -----
   < If:
   Then: lookup addr = vidcust k=1  i=b -nx
5  -----
   < If: not addr      'refer to this file as the alias"name" addr, not vidrec
   Then: show "@Customer code not on file!"; display ; screen ,1
6  -----
   < If: 'fill in the dummy variables on the screen with address
   Then: aa=addr(4) ; ab=addr(5) ; ac=addr(6) { " , " < addr(7) < addr(8)
File: vidrec-----Processing: input
1  -----
   F8 -Block Func, F9 -Go To/Find
   PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
   F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

```

ACTION          DEFINE  PROCESSING          4.8.05K409 DEMO
-----
7  -----
   < If: 'fill in the dummy variable for phone
   Then: ad=addr(9)
8  -----
   < If: 'assignment to real fields is allowed on the INPUT table!
   Then: 2=addr(2) ; 3=addr(3) ; 4=addr(11)
9  -----
   < If: 'without this display. the user would not see these assignments!
   Then: display ; end
10 -----
   < If:
   Then:
11 -----
   < If:
   Then:
12 -----
   < If:
   Then:
File: vidrec-----Processing: input
1  -----
   F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
   PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
   F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

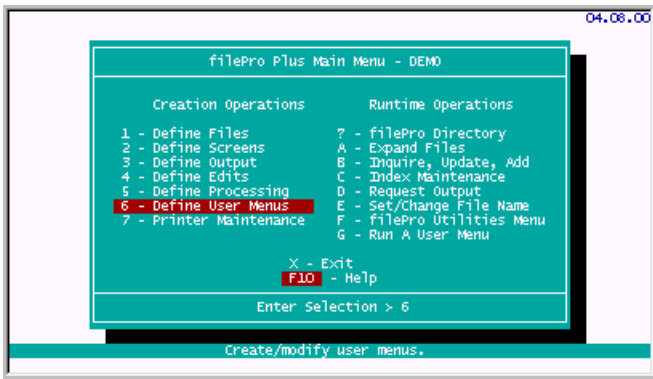
When you are done, press ESC to save your work.

Do a syntaxcheck and fix any problems.

Note that besides assigning the address and phone from the "vidcust" file to the dummy variables which will be displayed on the screen, this processing also assigns some of the customer file data to "real" fields in this file.

Before we try any of this programming, let's add the new file we created, the receipts file, to our existing user menu.

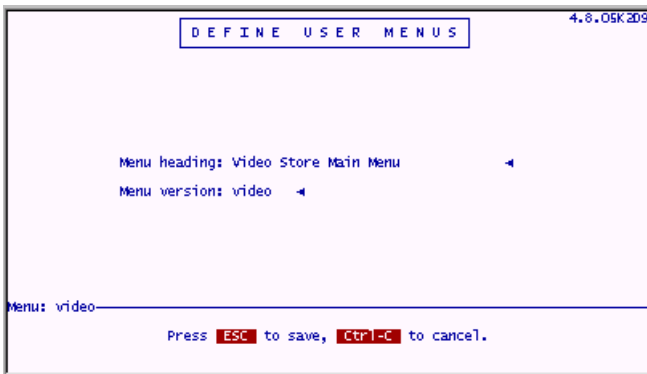
Select 6.



Choose "video". (Note that pressing "V" will jump you to the name as well, then when you press ENTER the rest of the name will be filled in automatically. All the filePro point-and-pick menus work like this.)

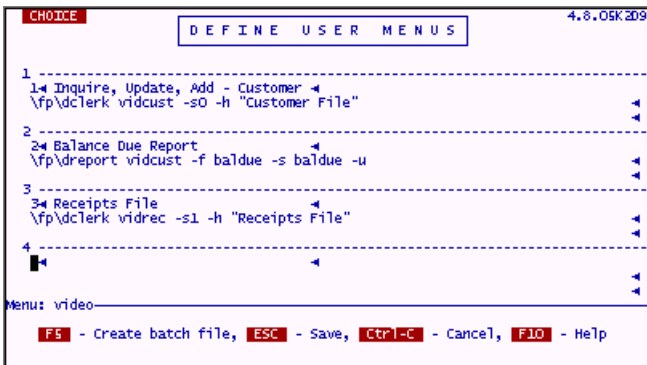


The following screen will appear:



Press ESC on this screen without making any changes.

Add the following (line 3) to this menu:



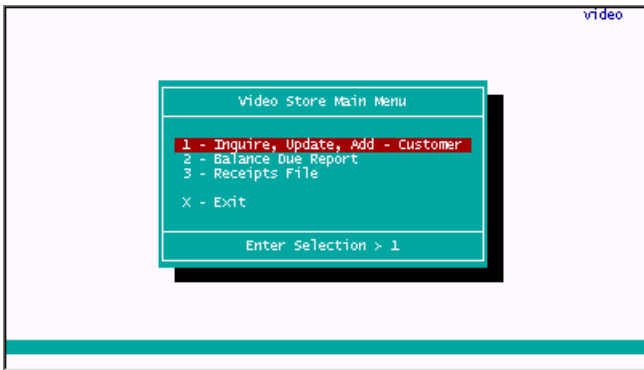
Press ESC to save your work.

It's finally time to test out all of this programming!

Run the user menu "video" by pressing "G" at the main menu and choosing it.

The menu now looks like this.

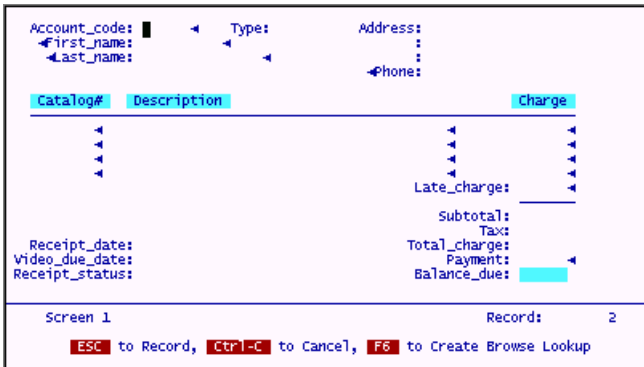
Select 3.



At the "clerk" menu, **Select 3** to add records.

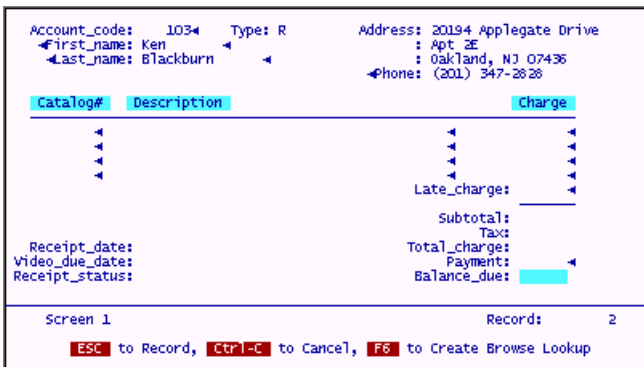
The screen will look like the following.

Enter the following, (Account# 103) and **press ENTER**.



The name, address, type of customer and phone will pop in.

The screen will look as follows:



Note that this information popped in immediately after you entered the Account# 103 and pressed ENTER. This is exactly what we programmed it to do. The code on the INPUT table says that when the cursor leaves field 1 the lookup should be performed. By pressing ENTER after putting in the 103 your cursor was "leaving" field 1 on its way to the next field in the cursor path. This is the specific "trigger" that caused the lookup processing code to activate. Because there was a "DISPLAY" command just before the "END" of this code, the screen fields are all refreshed to their most current values. We see all the assignments that were just made by this lookup.

Calculating Sales Tax and Totals

We are going to add some enhancements to this receipt file now. We need to make it automatically calculate the subtotal, the tax and the totals each record generates. First we will work with the tax field.

We'll use the control file we built earlier to hold system generated unique numbers to help us with the tax calculation. This time, we will add a field to that file which will hold a static (or relatively static) piece of information. In this case, we want to store the tax rate for our state. When a charge is made for these rental videos, we can calculate the appropriate tax amount and add it to the total charge.

Go into Define Files on the "vidctrl" file.

Add the following field (#2).

```

DEFINE FILES                                4.8.05K209
-----Field Heading-----
Number  Len  --Type--
1 - Next_cust_code  64  .0
2 - tax_#          44  .3
3 -
4 -
5 -
6 -
7 -
8 -
9 -
10 -
11 -
12 -
13 -
14 -
15 -
16 -
File: vidctrl
Press F10 for help, ESC to record, CTRL-C to cancel,
F9 to go to field number, F8 for block functions.
Previous record length: 6 Current record length: 10
  
```

Press ESC to save your work.

Press X to finish the file design and enter a Y to "Create a screen" prompt. If you forget to do this, the old Screen 0 does not have this newly added field on it, and you will not be able to enter the tax rate.

```

DEFINE FILES                                4.8.05K209
-----Field Heading-----
Number  Len  --Type--
1 - Next_cust_code  64  .0
2 - tax_#          44  .3
3 -
4 -
5 -
6 -
7 -
8 -
9 -
10 -
11 -
12 -
13 -
14 -
15 -
16 -
File: vidctrl
Key
  
```

Create default formats

Create a screen 0 (Y/N): y

Create a default report (Y/N): N

Create automatic indexes (Y/N): N

Create qualifiers (Y/N): N

Press ESC To Record, CTRL-C To Cancel

When you are done adding the field and creating the default screen, go into IUA on the "vidctrl" file. Choose Screen 0 and go to record #1 and update it as follows.

```

VIDCTRL
Next_cust_code: 106
tax_#: .055

Screen 0      Enter Selection >
D -Delete, H -Hardcopy, U -Update, X -Exit, F -Print Form, B -Browse
Record: 1
  
```

Press ESC to save your work. Return to the main menu.

Let's add the processing to calculate tax. We will do this calculation when adding or modifying the charges on any receipt. The calculation will be done on the INPUT processing table, however, we can grab the tax rate from the control file on the automatic table. We will arrange the code to do this lookup only one time for each IUA session and store the tax rate in a global dummy field that will be available throughout the whole session. First, the AUTO part of this processing combination.

Go into Define Processing on the "vidrec" file. Choose AUT0 processing.

Enter the following code. Hint: Push down line 1 first using F3 and enter the new line 1. Then add the subroutine "gettax" starting on line 8 shown on the next screen.

```

LABEL          DEFINE  PROCESSING          4,8,05K409 DEMO
1  -----
  ◀ If: tx eq ""
  Then: gosub gettax
2  -----
  ◀ If: @cd eq " " 'If record is new (unsaved) END
  Then: end
3  -----
  ◀ If:
  Then: lookup addr = vidcust k=1 i=b -nx
4  -----
  ◀ If: not addr 'refer to this file as the alias"name" addr, not vidrec
  Then: show "Customer code not on file!" ; end
5  -----
  ◀ If: 'fill in the dummy variables on the screen with address
  Then: aa=addr(4) ; ab=addr(5) ; ac=addr(6) { " " < addr(7) < addr(8)
6  -----
  ◀ If: 'fill in the dummy variable for phone
  Then: ad=addr(9)
File: vidrec                                     Processing: auto
1  F8 -Block Func, F9 -Go To/Find
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

Press NxtPage and finish the AUTO code.

Enter the following processing. (Lines 8-11)

```

LABEL          DEFINE  PROCESSING          4,8,05K409 DEMO
7  -----
  ◀ If: 'without this display, the user would not see these assignments
  Then: display ; end
8  -----
gettax ◀ If:
  Then: rn(1,0,g) = "1"
9  -----
  ◀ If:
  Then: lookup vidctrl r=rn -n
10 -----
  ◀ If: not vidctrl
  Then: show "No control file!!!" ; return
11 -----
  ◀ If:
  Then: tx(4,3,g)=vidctrl(2) ; return
12 -----
  ◀ If:
  Then:
File: vidrec                                     Processing: auto
1  F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

Press ESC to save your work.

Choose INPUT processing.

We will build another subroutine to do the "totals" calculation for each record. It will add up the charges for each line item and put that value into the "Subtotal" field. It will also calculate the tax. The subtotal and tax will be added together to show the total charges. Payments will be subtracted from total charges to calculate the Balance_Due.

This could all be easily done on one processing line, but it will be shown here spread out across several lines so that each part of the routine can be easily understood. There will be a mix-and-match of field names and field numbers to aid in showing these calculations as clearly as possible.

Enter the following code as shown.

```

LABEL          DEFINE  PROCESSING          4,8,05K409 DEMO
10 -----
totals ◀ If:
  Then: Subtotal = 25 + 26 + 27 + 28 + Late_charge
11 -----
  ◀ If:
  Then: Tax = Subtotal * tx
12 -----
  ◀ If:
  Then: Total_charge = Subtotal + Tax
13 -----
  ◀ If:
  Then: Balance_Due = Total_charge - Payment
14 -----
  ◀ If:
  Then: return
15 -----
  ◀ If:
  Then:
File: vidrec                                     Processing: input
1  F8 -Block Func, F9 -Go To/Find
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

This will take care of the actual calculations for all the totaling to be done on each record.

Now, we will write the code which tells the program "when" to do these calculations.

Incidentally, filePro will "close up" blank lines on a processing table when you save it. This does not hurt anything, assuming that you know the blank line was there and are aware that it will go away. Programmers often use blank lines to separate subroutines graphically while they are coding them and then let filePro close them up as the table is being stored. You can do this too, just be sure that the code from the routine on top will never fall "accidentally" to the routine below. In this case, there is a RETURN on line 14, there is no fear of this subroutine running into one below it. But if it is easier for you to enter code in separate clearly distinct pieces, do so. For example, you can enter the next "when-to-do-it" code starting on line 16. FilePro will close up the blank space when you save your work.

Enter the following code.

```

ACTION          DEFINE  PROCESSING  4.8.05K409 DEMO
15 -----
@wlf25 ◀ If:
Then:
17 -----
@wlf26 ◀ If:
Then:
18 -----
@wlf27 ◀ If:
Then:
19 -----
@wlf28 ◀ If:
Then:
20 -----
@wlf10 ◀ If:
Then:
21 -----
@wlf14 ◀ If:
Then: gosub totals ; display ; end
File: vidrec ----- Processing: input
61 -----
F8 -Block Func, F9 -Go To/Find
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

Press ESC to save your work. (Remember, next time you look, line 15 will be "sucked up" and line 16 will be living there.)

Once again, it is time to try out the program. Incidentally, this is not an abnormal way of building a filePro (or any) application program. You design a little bit and then test it out. Design a little more and test it out. As you go, various modules of the program become solidified and need less work or enhancement. It would be difficult to write a completely functional program without testing at various points along the way... not unless you are Mozart reborn and think it all up in your head first, then just write it all down.

We will enter a known account code, and then enter in some fake catalog numbers and charges. Notice that the charges and tax are correctly calculated at each appropriate "trigger".

Enter the account code 103 and press ENTER.

```

Account_code: 103◀ Type: R      Address: 20194 Applegate Drive
◀First_name: Ken      ◀      : Apt 2E
◀Last_name: Blackburn ◀      : Oakland, NJ 07436
◀Phone: (201) 347-2828
-----
Catalog#  Description                               Charge
-----
123A ◀    Tin Cup ◀                               4.00◀
199B ◀    Terminator II ◀                         4.00◀
X495 ◀    The X Files - Volume 1 ◀                 3.50◀
◀
Late_charge: 2.00◀
Subtotal: 13.50
Tax: .74
Total_charge: 14.24
Payment: 12.13◀
Balance_due: 2.11
-----
Receipt_date:
Video_due_date:
Receipt_status:
-----
Screen 1                               Record: 1
ESC to Record, Ctrl-C to Cancel, F6 to Create Browse Lookup

```

When you are done experimenting, leave IUA. It does not matter whether you save the record or not.

Did you notice that when you entered the Account# and the fields popped in, your cursor was left in the Last Name field instead of in the first Catalog# field? This is not very elegant. It would be better to have the cursor go directly into that field upon retrieving the account information.

To do this, Enter the following "fix" at line 9 of the INPUT table for "vidrec".

```

LABEL          DEFINE  PROCESSING  4.8.05K409 DEMO
7 -----
◀ If: 'fill in the dummy variable for phone
Then: ad=addr(9)
8 -----
◀ If: 'assignment to real fields is allowed on the INPUT table!
Then: 2=addr(2) ; 3=addr(3) ; 4=addr(11)
9 -----
◀ If: 'without this display, the user would not see these assignments!
Then: display ; screen ,17
10 -----
totals ◀ If:
Then: Subtotal = 25 + 26 + 27 + 28 + Late_charge
11 -----
◀ If:
Then: Tax = Subtotal * tx
12 -----
◀ If:
Then: Total_charge = Subtotal + Tax
File: vidrec ----- Processing: input
1 -----
F8 -Block Func, F9 -Go To/Find
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

This simple change tells filePro to put your cursor into field 17 on the current screen instead of letting it go into the next field on the cursor path (which is what happened when this was an END command.) You will see this new better behavior later when we do some more testing.

Simple Browse Lookup

Meanwhile, we have only been able to call up customers by their Account#'s. How often do your customers remember their account number? Not often. In a Video Rental Store, probably never! However, the computer and this program needs these unique numbers to bring up the proper records and attach receipts to the proper accounts. There is a way to let the user find the Account#'s by supplying the "name" of the customer. (They don't usually forget this.)

The browse lookup function of filePro is ideal for doing this task. It will let you put in a few characters of someone's last name and then show you all the closest matches in alphabetical order. Besides the name, other information about the customer can be displayed so that you can further identify the correct account. Once you are sure you have the right account selected, you call that information into the receipt record and move on to enter the items being rented.

Here is another important aspect of filePro (and programming in general). If you write your code in a well-considered style, you won't have to write as much of it. You will be able to make certain pieces of code do double and triple duty, even if they are not actually subroutines. We can do this very thing in this instance. The code to fill in the address and other customer info is already written. These assignments are found on line 6. All we have to do is give this line a "label" and we can direct another part of the program here.

Add the label "filscr" at line 6, while you are at it, make sure all the rest of your code matches this screen.

```
LABEL      DEFINE  PROCESSING      4.8.05K409 DEMO
4  -----
  < If:
  Then: lookup addr = vidcust k=1 i=b -nx
5  < If: not addr 'refer to this file as the alias" name" addr, not vidrec
  Then: show "Customer code not on file!" ; display ; screen ,1
6  -----
filscr < If: 'fill in the dummy variables on the screen with address
  Then: aa=addr(4) ; ab=addr(5) ; ac=addr(6) { " , " < addr(7) < addr(8)
7  -----
  < If: 'fill in the dummy variable for phone
  Then: ad=addr(9)
8  -----
  < If: 'assignment to real fields is allowed on the INPUT table!
  Then: z=addr(2) ; 3=addr(3) ; 4=addr(11)
9  -----
  < If: 'without this display, the user would not see these assignments!
  Then: display ; screen ,17
File: vidrec ----- Processing: input
7  -----
  F8 -Block Func, F9 -Go To/Find
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
  F10 For Help, ESC To Record, Ctrl-C to Cancel.
```

When you are done, go to line 22 and add the rest of the code to call up customer accounts by name.

First, enter some controlling code as follows.

```
LABEL      DEFINE  PROCESSING      4.8.05K409 DEMO
22 -----
  < If:
  Then: e=3 ; end
23 -----
  < If: 3 eq e and e ne "" 'name is already filled from earlier too
  Then:
24 -----
  < If: 3 eq ""
  Then: screen ,1
25 -----
  < If:
  Then:
26 -----
  < If:
  Then:
27 -----
  < If:
  Then:
File: vidrec ----- Processing: input
  F8 -Block Func, F9 -Go To/Find, F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
  F10 For Help, ESC To Record, Ctrl-C to Cancel.
```

When you are done, put your cursor on line 25 and press F5 to design a browse lookup using the lookup wizard.

Enter the data shown on the following screen.

```
DEFINE  LOOKUP      DEMO
-----
Lookup From File: vidcust
Name Of Lookup Is: addr
How Is The Record To Be Found? k K - Key Field R - Record Number
                               f F - Free Record Z - Fuzzy Search
What Index Is To Be Used? a A - F6 for List
What Field In 'vidrec' 3
Contains The Key?
Protect Record in Lookup File? N
If Key Doesn't Match Exactly: g X - Key Must Match Exactly
                               G - Use Next Greater Match
                               L - Use Next Lower Match
If Lookup Fails: n B - Blank The Field
                  N - Do Nothing
                  E - Report An Error
Create Browse Lookup? Y
Press Ctrl-C To Cancel Changes
```

Because you have put a Y in the "Create Browse Lookup" prompt, the wizard will give you a new screen to finish the design.

Complete the browse section of this lookup by filling in the prompts on the next screen.

Note that the fields from the file being looked into are displayed for reference. (You can even see their lengths and edits by pressing F5.)

```

DEFINE LOOKUP DEMO
Browse Header:
Browse Format:

1- Account_code      7- State            13- r
2- First_Name        8- Zip              14- Charges
3- Last_Name         9- Phone            15- Payments
4- Address1          10- r               16- Balance_due
5- Address2          11- Customer_type
6- City              12- First_sale_date
Press F5 For Field Lengths and Edits
Press Ctrl-C To Cancel Changes

```

Enter the following data as shown. (Prompts will appear sequentially. You may move backward and forward through them. You may even return to the first browse screen by pressing the UP Arrow while in the Browse Header field.)

```

DEFINE LOOKUP DEMO
Browse Header:
Name                Adress                Phone
Browse Format:
*2                 *4                 *9
Browse Window: Lines: 12 Row: Col:
Pop-Up Screen: Name: 0 Row: Col:
Only Show Records That Match Key? N Y-Yes, N-No, P-show Partial match
Browse Window Show option: P 0 - Show Only, Accept no keystrokes.
(Leave blank for default) K - Keep Browse Window on Screen.
P - Retain Position on re-execution.
Exit Keys:
(Leave blank for none)
Order of Records (Ascending, Descending): A
Position of First Record (Top,Middle,Bottom): M
Processing Label:
Press Ctrl-C To Cancel Changes

```

When you are finished, filePro creates the browse lookup code for you. It is important to note that this usually takes up two lines and extends beyond the screen to the right. You may use your cursor to scroll either line so as to view all the code. However, do NOT make changes to a browse lookup line from this processing table screen. Always use the lookup wizard which will create the lines properly. A misplaced space or punctuation mark will really mess things up.

Your screen should look like this:

```

LABEL DEFINE PROCESSING 4.8.05K409 DEMO
22 -----
ewef3 < If:
Then: e=3 ; end
23 -----
ewlf3 < If: 3 eq "" and e ne ""
Then: screen ,1
24 -----
< If:
Then: lookup addr = vidcust k=3 i=a -ng b="(brw=12 show=keep pop=0
25 -----
< If:
Then: Phone]*2 <3 *4
26 -----
< If:
Then:
27 -----
< If:
Then:
File: vidrec Processing: input
1 F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, ALT-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

The first thing we want to do is put a label on the browse lookup line itself. This will be useful later.

Put the label "brwNAME" on line 25.

Now finish the browse lookup by adding the following code.

```

LABEL DEFINE PROCESSING 4.8.05K409 DEMO
25 -----
brwNAME < If:
Then:
26 -----
< If:
Then:
27 -----
< If: @sk eq "BRKY"
Then: clearb ; 3 = "" ; display ; screen ,3
28 -----
< If: @sk eq "SAVE"
Then: goto brwNAME
29 -----
< If: not addr
Then: show "No names" ; clearb ; 3 = "" ; display ; screen ,3
30 -----
< If: 'this means they pressed ENTER, same as @sk eq "ENTR"
Then: clearb ; goto filscr
File: vidrec Processing: input
1 F8 -Block Func, F9 -Go To/Find
PgUp / PgDn -Page Up/Down, F6 -View Fields, ALT-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

This code below the browse lookup pair of lines (lines 27-30) tests what the user does while the browse is on the screen. The way this browse is written, there are three keys which are important. That is there are only three keys a user can press while the browse is up that give control back to the processing table. They are ESCAPE, BREAK and ENTER. The system maintained field @sk (special key) is filled with labels at various trigger points within filePro. This is one of those trigger points. If the user presses ESC, @sk will contain "SAVE". If the user presses BREAK, @sk will contain "BRKY" and if the user presses ENTER, @sk will be filled with "ENTR". As soon as any one of these keys is pressed the processing falls through to the line immediately below the browse lookup pair. Here we can act on whichever key was pressed and do appropriate things.

If the user pressed Control-C (BREAK), the browse is cleared (CLEARB) and the cursor is put back into a cleared field 3.

If ESC was pressed, we want nothing to happen. The best way to accomplish this is to send the processing directly back to the browse line. Remember, we labeled this "brwNAME".

If there are no records found that match the key of this browse lookup, the "if" condition on line 29 will be TRUE. A message will be shown to the user, the browse window is cleared, and the cursor is returned to a cleared field 3. This will all happen without user interaction, except for the acknowledgement of the SHOW warning message. (It is important to note that filePro will also put up a message about there being no records found for this browse lookup... that is part of the browse lookup function itself. This built-in warning message can be eliminated with the -s switch.)

If the user pressed ENTER while looking at the browse display, the processing will fall through to line 30 and the data from whichever record the highlighted bar was resting on will be loaded into the current record. This is done by simply doing a GOTO the "flscr" label we just added to the regular lookup used by the Account# search routine.

Once again, we get to try some of this code out.

Use the menu "video" and enter the receipts file.

Go into Add Records mode. Skip past the Account# field by pressing ENTER and put a "j" in the Last Name field.

```
Account_code:  Type:      Address:
First_name:   Type:      :
Last_name: j   Type:      :
Phone:        Type:      :

Catalog#  Description  Charge
-----
Late_charge:
Subtotal:
Tax:
Total_charge:
Payment:
Balance_due:

Receipt_date:
Video_due_date:
Receipt_status:

Screen 1 Record: 2
ESC to Record, Ctrl-C to Cancel, F6 to Create Browse Lookup
```

The following browse window will appear.

```
Account_code:  Type:      Address:
First_name:   Type:      :
Last_name: j   Type:      :
Phone:        Type:      :

Beginning of File
Name          Address      Phone
-----
Ken Blackburn 2014 Applegate Drive (201) 347-2828
John Jones    1521 Cypress Street (201) 443-2222
Maggie Hale  301 Ipswich Road (201) 429-3838
Mary Rodrigue 97 First Ave. (201) 928-3838
Karen Smith   111 Happy Lane (201) 427-9888
End of File

Receipt_date:
Video_due_date:
Receipt_status:
Total_charge:
Payment:
Balance_due:

Use Arrows PgDn PgUp to move cursor, ← to choose record, V to View Popup
```

A very nice built-in feature of a simple browse lookup is the View function.

By pressing V, the record under the highlighted bar is pulled up in full screen view as shown below. This view function can help you isolate a desired record since more information is available than will fit on a browse line.

```
VIDCUST

Account_code: 101
First_Name: John
Last_Name: Jones
Address: 1521 Cypress Street
Address2:
City: Hawthorne
State: NJ
Zip: 07506
Phone: (201) 443-2222

Customer_type: S
First_sale_date: 10/04/99

Charges: 104.00
Payments: 92.00
Balance_due: 12.00

Press any key to continue.
```

When the user presses ENTER from the full view screen, the popup is brought down and the user is returned to the browse window.

When the desired record is highlighted and the user presses ENTER, the browse is cleared and that record's data is brought into the current record and assigned just as if it were done with a regular lookup. It is important to note that the Account# which was not known, is now assigned to field 1 just as if we knew it all along.

```

Account_code: 101 Type: S Address: 1521 Cypress Street
First_name: John
Last_name: Jones
Phone: (201) 443-2222

Catalog# Description Charge
-----
Late_charge:
Subtotal:
Tax:
Total_charge:
Payment:
Balance_due:

Receipt_date:
Video_due_date:
Receipt_status:

Screen 1 Record: 2
ESC to Record, Ctrl-C to Cancel, F6 to Create Browse Lookup

```

When you are done experimenting with this browse lookup, exit IUA and return to the main menu. It is not important whether you save any records or not.

Just about everything needed to add a receipt is done now except a final touch. The date and status fields must be filled in by processing.

Go into Define Processing on the "vidrec" file. Choose INPUT processing.

Put your cursor on line 15 and press F3 3 times to open up some blank lines. Add the code shown below.

```

ACTION DEFINE PROCESSING 4.8.05K409 DEMO
13
  If:
  Then: Total_charge = Subtotal + Tax
14
  If:
  Then: Balance_Due = Total_charge - Payment
15
  If: 6 eq ""
  Then: 6=@td ; 7=@td+"3"
16
  If:
  Then: 8="O"
17
  If: Balance_due eq "0"
  Then: 8="C"
18
  If:
  Then: return
File: vidrec Processing: input
6 F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt+F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

This code fills the date field 6 with today's date, and fills the due date with today's date plus 3 days. (This should be enough time to watch any video.)

Because of the "if" condition on line 15, the date assignment and calculation will only be done if the date field is empty. Any new receipt will have a blank date and the process will happen. Subsequent updates of this record will not cause the date(s) to be changed.

The status field is also set during this totals subroutine since it is dependant on the Balance_Due which is calculated and assigned here. This is a nifty and very useful snippet of processing. It is clever because, it first sets the Balance_Due field to "O" (for OPEN) no matter what. Then, on the next line, if the Balance_Due is exactly equal to "0" (zero) it is flipped to a C (for CLOSED). Two of the simplest lines you will ever see, but absolutely foolproof for setting such a status flag. (This field is sometimes called a flag because it quickly flags a record's status without having to test a field (in this case the Balance_Due). You can sort and select records based on this field as well.)

Enter the "vidrec" file and try out this newly added code.

Enter the following data and you will see the resulting date/status calculations immediately upon moving through one of the money fields.

```

Account_code: 101 Type: S Address: 1521 Cypress Street
First_name: John
Last_name: Jones
Phone: (201) 443-2222

Catalog# Description Charge
-----
123MH Max Headroom 10.00
Late_charge:
Subtotal: 10.00
Tax: .55
Total_charge: 10.55
Payment:
Balance_due: 10.55

Receipt_date: 10/06/99
Video_due_date: 10/09/99
Receipt_status: O

Screen 1 Record: 2
ESC to Record, Ctrl-C to Cancel, F6 to Create Browse Lookup

```

Feel free to experiment with this program as it stands now. When you are done, return to the main menu.

To make the receipts file "vidrec" much more usable, we will add some automatic indexes to it.

Go into Index Maintenance on the "vidrec" file.

Build Automatic Indexes on the Account_code field, the Last_Name and First Name fields, and the status field.

The Automatic Indexes screen should look like this when you are done.

INDEX MAINTENANCE

Automatic indexes

Index A - Built on Field 1 <Account_code>
Index B - Built on Field 3 <Last_name>, 2
Index C - Built on Field 8 <Receipt_status>
Index D - Not Used
Index E - Not Used
Index F - Not Used
Index G - Not Used
Index H - Not Used
Index I - Not Used
Index J - Not Used
Index K - Not Used
Index L - Not Used
Index M - Not Used
Index N - Not Used
Index O - Not Used
Index P - Not Used

File: vidrec

Enter Index (A-P):

Press **Ctrl-C** To Cancel.

The @key Trigger

There is another small enhancement we can make to the receipts file which will help users enter data quickly.

More importantly, it will disclose the usefulness of another filePro "trigger", @key processing. (pronounced "at key") It is processing that occurs whenever the user presses a designated key while the cursor is at the Enter Selection prompt. This is exactly the same as the keys filePro displays at the bottom of each record in IUA. H-Hardcopy, B-Browse, U-Update, etc. You can design keys like this of your own. If you pick one of the keys already used internally by filePro like the H key (Hardcopy), the filePro function will be disabled and the processing you write for this key will be done instead.

We will design an @key that lets the user take a payment quickly. Normally, the user would press U to update an existing receipt, then they would have to press many ENTERS or TABS to get to the Payment field where they would enter the amount. The following single line of code makes this much easier. It will put the user directly into the Payment field. When the amount is entered, the user presses ESC as normal and the process ends. No need to push the cursor all through the other fields on the screen just to take a payment.

Go to the INPUT processing table of "vidrec".

Enter the following code as shown.

```
LABEL          DEFINE PROCESSING          4.8.05K409 DEMO
31  < If: not addr
    Then: show "No names" ; clearb ; 3="" ; display ; screen ,3
32  < If:
    Then: 'this means they pressed ENTER, same as @sk eq "ENTR"
    Then: clearb ; goto filscr
33  @keyP < If:
    Then: screen ,14 ; end
34  < If:
    Then:
35  < If:
    Then:
36  < If:
    Then:
File: vidrec                                     Processing: input
F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.
```

Press ESC to save your work.

Try this out by ENTERING IUA in the "vidrec" file.

Stand on any existing filled record and while your cursor is blinking at the Enter Selection prompt, press P. You will see the following.

```
Account_code: 101 Type: S Address: 1521 Cypress Street
First_name: John
Last_name: Jones Hawthorne, NJ 07506
Phone: (201) 443-2222

Catalog# Description Charge
12345 Max Headroom 10.00
Late_charge:
Subtotal: 10.00
Tax: .55
Total_charge: 10.55
Payment:
Balance_due: 10.55

Receipt_date: 10/06/99
Video_due_date: 10/09/99
Receipt_status: 0

Screen 1 Record: 2
ESC to Record, Ctrl-C to Cancel, F6 to Create Browse Lookup
```

When you are done entering an amount, press ESC to store the payment.

This is a nice addition, but how can we inform the user that it's available?

Simple, we can use yet another type of "trigger" processing, @entsel. This is processing that happens just before the cursor is put at the Enter Selection prompt. It is very useful for many things, but especially good for putting up prompts, since that is when the regular filePro prompts appear!

Go to the INPUT processing table on the "vidrec" file.

Adjust the @keyP line and add the other code shown below. Besides the @entsel code, another @key process, @keyU is defined. The idea here, is to put the prompt P in reverse video on the screen just as if it were a regular filePro prompt, and take it away whenever the user enters Update Mode. Update Mode can now be reached by either pressing U (the usual filePro key) or P our newly added @key. If either of these is pressed, the prompt is taken off the screen, since it would do no good to press a P while in the Payment field or any other field for that matter. For the @keyU, we must also duplicate the standard filePro action of putting the user into Update Mode. This is done by executing the RESTART command. It does exactly that, puts the cursor in the first field on the cursor path and resets the processing pointer to the top of the INPUT table.

```
LABEL          DEFINE PROCESSING          4.8.05K409 DEMO
31  < If: not addr
    Then: show "No names" ; clearb ; 3="" ; display ; screen ,3
32  < If:
    Then: 'this means they pressed ENTER, same as @sk eq "ENTR"
    Then: clearb ; goto filscr
33  @keyP < If:
    Then: screen ,14 ; end
34  @entsel < If:
    Then: show "Press \r P \r=Payment" ; end
35  @keyU < If:
    Then: show "" ; restart
36  < If:
    Then:
File: vidrec                                     Processing: input
F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.
```

Press ESC to save your work.

Test this code by going into IUA on the "vidrec" file. Go to any existing and filled record. You will see the new P prompt along with all the others, like this.

```
Account_code: 101 Type: S Address: 1521 Cypress Street
First_name: John Last_name: Jones : Hawthorne, NJ 07506
Phone: (201) 443-2222

Catalog# Description Charge
123MH Max Headroom 10.00
Late_charge:
Subtotal: 10.00
Tax: .55
Total_charge: 10.55
Payment: 10.55
Balance_due: .00

Receipt_date: 10/06/99
Video_due_date: 10/09/99
Receipt_status: C

Screen 1 Enter Selection > Record: 2
Press P -Payment
D -Delete, H -Hardcopy, U -Update, X -Exit, F -Print Form, B -Browse
```

When you press either P to take a payment, or U to update the record, the prompt will be removed from sight along with all the others that filePro removes for you.

This is a good start on a useful application, but there is far more to do, and learn. See you in the Finish Design!

Add New Customers While In The Receipts File

If most of the work is being done in the receipts file, i.e., entering charges, taking payments, etc, it is a waste of time to force the user to back out of this file and enter the customer file just to add a new account. This becomes even more of a problem when such a program is used at the Point Of Sale where time and speed of operation are the critical measurement of how good a program works.

There is no need to leave one filePro file simply to add a record in another one. The whole operation, including getting the next unique account number, can be done from just about anywhere inside filePro.

From this point on, since this is the Advanced Guide, it will be assumed that you know how to enter Define Processing on whichever file is listed. Do the work on whichever file is specified. These names can be found on the bottom left and right of each processing screen. These basic instructions will be left out of this section of the tutorial guides.

Add the following lines of code (25 through 56) to the "vidrec/input" table. Modify any lines that already exist to match the following four screens.

```

LABEL          DEFINE PROCESSING          4.8.05K409 DEMO
-----
25  < If:
    Then: show "Enter a period (.) to add a new customer"
26  < If:
    Then: e=3 ; end
27  < If:
    Then: show ""
28  < If: 3 co "."
    Then: 3=""; gosub addnew
29  < If: 1 ne ""
    Then: screen ,17
30  < If: 3 eq "" and e ne "" 'name already filled from earlier lookup
    Then: end
File: vidrec          Processing: input
1  F8 -Block Func, F9 -Go To/Find
   PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
   F10 For Help, ESC To Record, Ctrl-C to Cancel.
  
```

```

LABEL          DEFINE PROCESSING          4.8.05K409 DEMO
-----
39  < If:
    Then: show "Press \r P \r=Payment" ; end
40  < If:
    Then: show "" ; restart
41  < If:
    Then:
42  < If:
    Then: show "Press \r \K4 \r to record, \r \KY \r to Cancel"
43  < If:
    Then: lookup new = vidcust r=free -ep
44  < If:
    Then: popup update new,0 ; clearp
File: vidrec          Processing: input
1  F8 -Block Func, F9 -Go To/Find, F5 -To Define A Lookup
   PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
   F10 For Help, ESC To Record, Ctrl-C to Cancel.
  
```

```

LABEL          DEFINE PROCESSING          4.8.05K409 DEMO
-----
45  < If: @sk eq "BRKY"
    Then: write new ; delete new ; screen ,3
46  < If: 'contents of popup screen are posted out to the lookup file
    Then: 'automatically at this line, we need to get the next account#
47  < If: 'and post it to the new looked up record manually
    Then: 'this is done with virtually the same lookup done in vidcust
48  < If:
    Then: lookup num = vidctrl r=rn -np
49  < If: not num
    Then: show "No control file!!!" ; write new ; delete new ; screen ,3
50  < If:
    Then: new(1)=num(1); num(1)=num(1)+"1"; write num
File: vidrec          Processing: input
1  F8 -Block Func, F9 -Go To/Find, F5 -To Define A Lookup
   PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
   F10 For Help, ESC To Record, Ctrl-C to Cancel.
  
```

```

LABEL          DEFINE PROCESSING          4.8.05K409 DEMO
-----
51  < If:
    Then: 'now we can fill our current record, real fields and dummies
52  < If:
    Then: 1=new(1); 2=new(2); 3= new(3); 4 = new(11)
53  < If:
    Then: aa=new(4); ab=new(5); ac=new(6) {" "<new(7)<new(8); ad=new(9)
54  < If:
    Then: write new
55  < If:
    Then: display; return
56  < If:
    Then:
File: vidrec          Processing: input
1  F8 -Block Func, F9 -Go To/Find, F5 -To Define A Lookup
   PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
   F10 For Help, ESC To Record, Ctrl-C to Cancel.
  
```

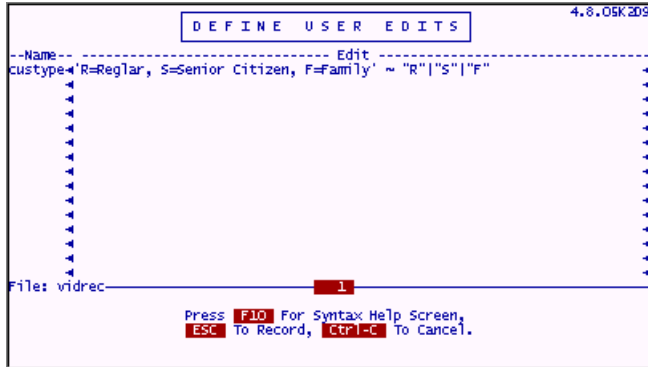
The code above will activate when the user's cursor leaves field 3 (Last Name) while there is a period in it. This is a user-defined trigger if you will that makes use of the

when-leaving field trigger to do its work. If the field contains a period, the program looks up a free record in the customer file "vidcust" and pops up Screen.0 from that file. After the user is finished entering the new account information, this data is stored on the free record in the customer file and the next available (and unique) Account# is assigned to it.

Except for the **POPUP UPDATE** command on line 44, we have seen all this code in one variation or another before. The **POPUP UPDATE** puts the user on the screen designated from the lookup file. When the user is finished entering data and presses **ESC**, filePro writes all the fields on this screen to the looked up record automatically. The only thing left for us to do before we give control back to the user is obtain the next unique Account# and assign it to this record's field 1. At this point, the new customer account becomes available for use by the file the user is actually standing in, "vidrec" (the receipts file).

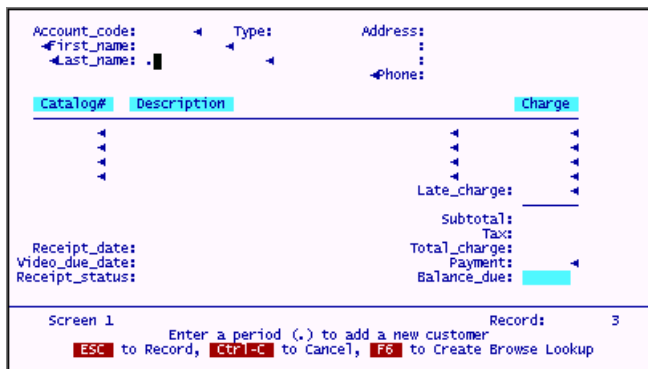
It is necessary to put the same user edit that resides in the "vidcust" file into the "vidrec" file. This way, when the user enters the account type for the new customer, the field will only allow the same codes that are allowed in the customer file.

Go to Define Edits for the "vidrec" file and add the following.



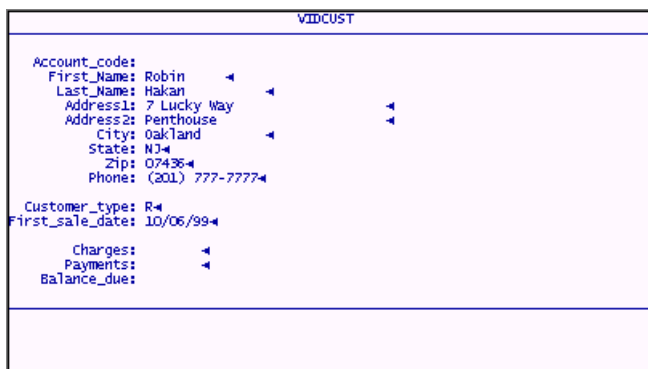
Go into the "vidrec" file and try out this new feature.

While standing on a new record, enter a period in the Last Name field and press **ENTER**.



A blank customer info screen will appear.

Enter the following data.



Press **ESC** to save the new customer record.

The following will appear on your screen. The newly added customer data and its specific Account# have **popped** into place. You could immediately add Catalog#'s to this receipt. We will not do this now.

```

Account_code: 106 Type: R Address: 7 Lucky Way
First_name: Robin Penthouse
Last_name: Hakan Oakland, NJ 07436
Phone: (201) 777-7777

Catalog# Description Charge
-----
Late_charge:
Subtotal:
Tax:
Total_charge:
Payment:
Balance_due:

Receipt_date:
Video_due_date:
Receipt_status:

Screen 1 Record: 3
Press ESC to record, Ctrl-C to Cancel
ESC to Record, Ctrl-C to Cancel, F6 to Create Browse Lookup

```

If you were to search for this customer in the "vidcust" file, you would find that is has been correctly added in that file.

Posting

The next enhancement to be added is a posting routine. We have a customer file "vidcust", and we want to store all the sales made to this customer in this file, as well as payments. An added benefit of doing this is that we can immediately see, by going to this file what the customer's current Balance_Due is.

For this posting routine to be robust and work under all situations, adding new receipts, modifying old receipts, deleting receipts, we need to make use of a small trick. We are going to take a "snapshot" of every record before the user does anything to it. This way, we can compare the snapshot of the "before" image with the way the record is left "after" the user is done updating it, and post any and all changes to the customer file.

The best place to take the "before" picture is the AUTO processing table, since this table always runs just after the record is retrieved from the disk and just before the user sees it on his screen.

We only need to snap a picture of two fields, the charges and the payments. If either or both of these change, we will send (post) those changes over to the customer file.

Enter the following code on line 2. This will save the "before" values of the charges and payments into "oc" and "op".

```
ACTION          DEFINE  PROCESSING          4.8.05K409 DEMO
-----
1  -----
   If: tx eq ""
   Then: gosub gettax
2  -----
   If:
   Then: oc(6,.2)=l3 ; op(6,.2)=l4
3  -----
   If: @cd eq " " 'If record is new (unsaved) END
   Then: end
4  -----
   If:
   Then: lookup addr = vidcust k=l i=b -nx
5  -----
   If: not addr 'refer to this file as the alias"name" addr, not vidrec
   Then: show @Customer code not on file!"; end
6  -----
   If: 'fill in the dummy variables on the screen with address
   Then: aa=addr(4) ; ab=addr(5) ; ac=addr(6) { " , " < addr(7) < addr(8)
File: vidrec -----Processing: auto
26  -----
   F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
   PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
   F10 For Help, ESC To Record, Ctrl-C to Cancel.
```

We will put the guts of the posting routine on the INPUT table for "vidrec".

First, add the following code on line 1.

```
ACTION          DEFINE  PROCESSING          4.8.05K409 DEMO
-----
1  -----
   If:
   Then: gosub post
2  -----
   If:
   Then: end
3  -----
wif1 If:
   Then:
4  -----
   If: l eq ""
   Then: end
5  -----
   If:
   Then: lookup addr = vidcust k=l i=b -nx
6  -----
   If: not addr 'refer to this file as the alias"name" addr, not vidrec
   Then: show @Customer code not on file!"; display ; screen ,l
File: vidrec -----Processing: input
11  -----
   F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
   PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
   F10 For Help, ESC To Record, Ctrl-C to Cancel.
```

Add the actual "post" subroutine as shown below.

```
ACTION          DEFINE  PROCESSING          4.8.05K409 DEMO
-----
57 post -----
   If:
   Then: lookup pst = vidcust k=l i=b -nxp
58 -----
   If: not pst
   Then: show @odd, customer not in customer file!"; return
59 -----
   If:
   Then: pst(14) = pst(14) + l3 - oc
60 -----
   If:
   Then: pst(15) = pst(15) + l4 - op
61 -----
   If:
   Then: pst(16) = pst(14) - pst(15)
62 -----
   If:
   Then: write pst ; return
File: vidrec -----Processing: input
19  -----
   F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
   PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
   F10 For Help, ESC To Record, Ctrl-C to Cancel.
```

An explanation of this code in English is fairly simple. When the user presses ESC to store the receipt record, the first thing to run is the "post" subroutine. It simply goes out to the appropriate customer record (using the unique account number for the lookup) and adds the difference between what is in the Charges field now (13) and what used to be in the Charges field ("oc") to the Charges field on the customer record. The same is true for the payments field. The difference between what is in the Payments field now (14) and what used to be in the Payments field ("op") is added to the Payments field in the customer record. Then the Balance_Due field of the customer record is recalculated.

This code will work whether the receipt is new or old. If the receipt is new, the snapshot value of charges and payments will be "0". The amount posted to the customer file will be the new value minus "0", which is the new value. If the receipt is old and the values are just being modified, only the difference will post. It always works under all possible situations.

Remember that each time the AUTO table runs, all regular dummy fields are cleared. The code we put on the AUTO table will reassign these dummies each time a new record is brought to the screen.

Try out this code in the receipts file.

Enter the following data.

```
Account_code: 106 Type: R Address: 7 Lucky Way
First_name: Robin Penthouse
Last_name: Hakan Oakland, NJ 07436
Phone: (201) 777-7777

Catalog# Description Charge
AA555 First Contact 5.00
Late_charge:
Subtotal: 5.00
Tax: .28
Total_charge: 5.28
Payment:
Balance_due: 5.28

Receipt_date: 10/06/99
Video_due_date: 10/09/99
Receipt_status: 0

Screen 1 Record: 2
ESC to Record, Ctrl-C to Cancel, F6 to Create Browse Lookup
```

When you are done, store this receipt.

Go to the customer file "vidcust" and see if the posting worked. You should see the following.

```
VIDCUST

Account_code: 106
First_name: Robin
Last_name: Hakan
Address1: 7 Lucky Way
Address2: Penthouse
City: Oakland
State: NJ
Zip: 07436
Phone: (201) 777-7777

Customer_type: R
First_sale_date: 10/06/99

Charges: 5.28
Payments:
Balance_due: 5.28

Screen 0 Enter Selection > Record: 6
D-Delete, H-Hardcopy, U-Update, X-Exit, F-Print Form, B-Browse
```

To finish the "post" routine, we must add a routine that will "back out" a receipt's previous postings if the entire receipt is deleted with the (D)elele key. This is an even simpler operation. Add the following code as an @key routine that does a lookup to the customer file and subtracts the charges on the receipt from the charges on the customer record, and subtracts the payments on the receipt from the payments on the customer record, then recalculates the Balance_Due on the customer record. Once the postings have been "reversed" or "backed out", the routine actually deletes the receipt record itself.

Enter the code from the following two screens into "vidrec/input".

```
LABEL DEFINE PROCESSING 4.8.05K409 DEMO
63 -----
keyd If:
Then: beep
64 -----
If:
Then: input popup q(1,yesno) "Are you SURE you want to delete this?"
65 -----
If: q ne "y"
Then: end
66 -----
If:
Then: gosub unpost
67 -----
If:
Then: delete; end
68 -----
unpost If:
Then: lookup del = vidcust k=1 i=b -m>p
File: vidrec Processing: input
1 F8 -Block Func, F9 -Go To/Find
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.
```

```
LABEL DEFINE PROCESSING 4.8.05K409 DEMO
69 -----
If: not del
Then: show "odd, customer not in the customer file.;"return
70 -----
If:
Then: del(14)=del(14) - 13
71 -----
If:
Then: del(15)=del(15) - 14
72 -----
If:
Then: del(16)=del(16) - del(15)
73 -----
If:
Then: write del ; return
74 -----
If:
Then:
File: vidrec Processing: input
1 F8 -Block Func, F9 -Go To/Find
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.
```

Try this code by deleting the receipt you just added. When you press D on this record, you will see:

```

Account_code: 106 Type: R Address: 7 Lucky Way
First_name: Robin Penthouse
Last_name: Hakan : Oakland, NJ 07436
Phone: (201) 777-7777

Catalog# Description Charge
AA555 Fir 5.00
Are you SURE you want to delete this?
Late_charge:
Subtotal: 5.00
Tax: .28
Total_charge: 5.28
Payment:
Receipt_date: 10/06/99 Video_due_date: 10/09/99
Receipt_status: 0 Balance_due: 5.28

Screen 1 Press P=Payment Records: 2
ESC to Record, Ctrl-C to Cancel, F6 to Create Browse Lookup

```

Answer Y to this prompt and then exit out of the receipts file.

Go to the customer file and you should see that the record has been correctly updated.

```

VIDCUST

Account_code: 106
First_Name: Robin
Last_Name: Hakan
Address1: 7 Lucky Way
Address2: Penthouse
City: Oakland
State: NJ
Zip: 07436
Phone: (201) 777-7777
Customer_type: R
First_sale_date: 10/06/99
Charges: .00
Payments: .00
Balance_due: .00

Screen 0 Enter Selection > Records: 6
D-Delete, H-Hardcopy, U-Update, X-Exit, F-Print Form, B-Browse

```


Adding A Catalog File

We have been operating our Video Store at a severe disadvantage. Each time a tape is rented, we have to put in its description and price. Obviously, this would never do in a POS (Point of Sale) situation. We need to add some processing that will let us keep our inventory in a file and call up its information (description/price for each item) as needed and instantly.

We'll call this file "vidcat" and refer to it as the Catalog file. It does not have to be a complicated file. In fact, it will probably be the simplest filePro file you ever define.

Go into Define Files and choose [NEW]. Enter the name "vidcat" and do not give it any password.

Enter the following data.

```
4.8.05K209
DEFINE FILES
Number  -----Field Heading-----  Len  --Type--
 1 -  Catalog#                          8    allup
 2 -  Description                       40   lowup
 3 -  Charge                            6    .2
 4 -
 5 -
 6 -
 7 -
 8 -
 9 -
10 -
11 -
12 -
13 -
14 -
15 -
16 -
File: vidcat
Press F10 for help, ESC to record, Ctrl-C to cancel,
F9 to go to field number, F8 for block functions.
Characters used: 54 Characters left: 32646
```

Press ESC to store your work.

Press X to finish designing the file.

Enter the following in the options dialog popup.

```
4.8.05K209
DEFINE FILES
Number  -----Field Heading-----  Len  --Type--
 1 -  Catalog#                          8    allup
 2 -  Description                       40   lowup
 3 -  Charge                            6    .2
 4 -
 5 -
 6 -
 7 -
 8 -
 9 -
10 -
11 -
12 -
13 -
14 -
15 -
16 -
File: vidcat
Key

Create default formats
Create a screen 0 (Y/N): y
Create a default report (Y/N): n
Create automatic indexes (Y/N): n
Create qualifiers (Y/N): n
Press ESC To Record, Ctrl-C To Cancel
```

To make things really easy to code and work very smoothly, we must adjust the "vidrec" file a little.

Go into Define Files for "vidrec" and modify the following fields to look like this. (We are making them into associated field groups.)

```
4.8.05K209
DEFINE FILES
Number  -----Field Heading-----  Len  --Type--
17 -  a0) Catalog#                      8    allup
18 -  a0) Catalog#                      8    allup
19 -  a0) Catalog#                      8    allup
20 -  a0) Catalog#                      8    allup
21 -  b0) Description                    40   '
22 -  b0) Description                    40   '
23 -  b0) Description                    40   '
24 -  b0) Description                    40   '
25 -  c0) Charge                        6    .2
26 -  c0) Charge                        6    .2
27 -  c0) Charge                        6    .2
28 -  c0) Charge                        6    .2
29 -
30 -
31 -
32 -
File: vidrec
Key
Press F10 for help, ESC to record, Ctrl-C to cancel,
F9 to go to field number, F8 for block functions.
Previous record length: 301 Current record length: 301
```

Press ESC to store your work.

Press X to finish the design of this file.

IMPORTANT: Do not put a Y in any option on this popup.

```

DEFINE FILES                                4.8.05K409
-----Field Heading-----Len--Type--
17 - a0) Catalog#          8  allup
18 - a0) Catalog#          8  allup
19 - a0) Catalog#          8  allup
20 - a0) Catalog#          8  allup
21 - b0)                    '  '
22 - b0)                    '  '
23 - b0)                    '  '
24 - b0)                    '  '
25 - c0)                    '  '
26 - c0)                    '  '
27 - c0)                    '  '
28 - c0)                    '  '
29 -                        '  '
30 -                        '  '
31 -                        '  '
32 -                        '  '
File: vidrec                               Key

```

Create default formats

Create a screen O (Y/N): N

Create a default report (Y/N): N

Create automatic indexes (Y/N): N

Create qualifiers (Y/N): N

Press ESC To Record, Ctrl-C To Cancel

The screen in the receipts file must now be changed. The charge and description for each item will now be automatically "popped in" by a lookup to the Catalog file. These fields must be changed to protected fields. The application will fill them now.

Make the receipts screen look like this.

```

Account_code: '1      Type: 14      Address: 'aa
First_name: 12
Last_name: '3
Phone: 'ad

Catalog#  Description  Charge
'17      !21            !25
'18      !22            !26
'19      !23            !27
'20      !24            !28
Late_charge: '10

Subtotal: !11
Tax: !12
Total_charge: !13
Payment: '14
Balance_due: !15

Receipt_date: !6
Video_due_date: !7
Receipt_status: !8

File: vidrec                               Color Screen: 1

```

Enter Selection >

F10 -Help, U -Update, S -Select Screen, F -Select File,
C -Copy Screen, H -Hardcopy, D -Delete Screens, X -Exit

There are changes needed in the processing table for the Receipts file as well.

Remove the @wlf labels for fields 25 through 28. The cursor will no longer pass through these fields, as they are now protected.

```

LABEL          DEFINE PROCESSING          4.8.05K409 DEMO
-----
19 -----
@wlf10 < If:
Then:
20 -----
@wlf14 < If:
Then: gosub totals ; display ; end
21 -----
spare < If:
Then:
22 -----
@wef3 < If:
Then: show "Enter a period (.) to add a new customer"
23 -----
< If:
Then: e=3 ; end
24 -----
@wlf3 < If:
Then: show ""
File: vidrec                               Processing: input
1 F8 -Block Func, F9 -Go To/Find, F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

Continue adding the new code for the Catalog file lookup at line 70. Add the processing from the following two screens.

```

LABEL          DEFINE PROCESSING          4.8.05K409 DEMO
-----
69 -----
< If:
Then: write del ; return
70 -----
@wefa0 < If:
Then: e=a0 ; end
71 -----
@wlf14 < If:
Then: dim cat[4]:17 ; dim des[4]:21 ; dim chg[4]:25
72 -----
< If: a0 eq e
Then: end
73 -----
clrline < If: a0 eq ""
Then: des[eaf]=""; chg[eaf]=""; gosub totals ; display ; screen , (efd)
74 -----
< If:
Then: lookup itm=vidcat k=a0 i=a -mx
File: vidrec                               Processing: input
1 F8 -Block Func, F9 -Go To/Find
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

```

LABEL          DEFINE PROCESSING          4.8.05K409 DEMO
75 -----
  If: not itm
  Then: show "Catalog item not on file!"; a0="" ; goto clrline
76 -----
  If:
  Then: des[caf]=itm(2) ; chg[caf]=itm(3) ; gosub totals; display ; end
77 -----
  If:
  Then:
78 -----
  If:
  Then:
79 -----
  If:
  Then:
80 -----
  If:
  Then:
File: vidrec -----Processing: input
  F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
  PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt+F9 -Toggle Insert
  F10 -For Help, ESC -To Record, Ctrl-C -to Cancel.

```

In order for the code to work, an index is needed on the catalog# field in the "vidcat" file. Go to **Index Maintenance** and add the following.

```

INDEX MAINTENANCE          4.8.35409
Automatic indexes
Index A - Built on Field 1 <Catalog#>
Index B - Not Used
Index C - Not Used
Index D - Not Used
Index E - Not Used
Index F - Not Used
Index G - Not Used
Index H - Not Used
Index I - Not Used
Index J - Not Used
Index K - Not Used
Index L - Not Used
Index M - Not Used
Index N - Not Used
Index O - Not Used
Index P - Not Used
File: vidcat -----Enter Index (A-P):
Press Ctrl-C To Cancel.

```

Add the Catalog file to the "video" menu as shown.

```

ACTION          DEFINE USER MENUS          4.8.05K209
1 -----
1- Inquire, Update, Add - Customer
  \fp\dclerk vidcust -s0 -h "Customer File"
2 -----
2- Balance Due Report
  \fp\dreport vidcust -f baldue -s baldue -u
3 -----
3- Receipts File
  \fp\dclerk vidrec -s1 -h "Receipts File"
4 -----
4- Video Catalog File
  \fp\dclerk vidcust -s0 -h "Catalog File"
Menu: video
  F5 - Create batch file, ESC - Save, Ctrl-C - Cancel, F10 - Help

```

Use this menu choice to enter the Catalog file through IUA.

Choose **Add Records mode** . Add the following item

```

VIDCAT
Catalog#: A123
Description: Dirty Dancing
Charge: 4.00
Screen 0          Enter Selection > |          Record: 1
  D -Delete, H -Hardcopy, U -Update, X -Exit, F -Print Form, B -Browse

```

Press **ESC** to store your work.

While you are in Add Records mode, add the following Catalog items as well. We will need them for testing purposes.

Record #	Cat#	Description	Charge
2	ABC1	Serial	6.00
3	1001	Crossroads	15.00
4	1002	Around The World In 80 Days	4.00
5	1383M	Mission: Impossible	4.00
6	444JM	Fiddler On The Roof	6.00
7	3999	Total Recall	4.00
8	7744	French Kiss	5.00
9	888COU	Pure Country	4.00
10	6222G	Ghost	3.00

***** END OF FILE *****

ARROWS -Change Record, ← -Select Record, U -Update Record
S -Set Selection, Z -fuzzy search, R -Reset C -Continuous On
F -Format, n -Hardcopy, X -Exit

Finally, we are ready to try this out.

Add the following receipt and enter the following in the first Catalog# field.

```

Account_code: 106 Type: R Address: 7 Lucky Way
First_name: Robin Penthouse
Last_name: Hakan Oakland, NJ 07436
Phone: (201) 777-7777

Catalog# Description Charge
1002
Late_charge:
Subtotal:
Tax:
Total_charge:
Payment:
Balance_due:

Receipt_date:
Video_due_date:
Receipt_status:

Screen 1 Record: 4
ESC to Record, Ctrl-C to Cancel, F6 to Create Browse Lookup

```

You should see the following information pop in.

```

Account_code: 106 Type: R Address: 7 Lucky Way
First_name: Robin Penthouse
Last_name: Hakan Oakland, NJ 07436
Phone: (201) 777-7777

Catalog# Description Charge
1002 Around The World In 80 Days 4.00
Late_charge:
Subtotal: 4.00
Tax: .22
Total_charge: 4.22
Payment:
Balance_due: 4.22

Receipt_date: 10/07/99
Video_due_date: 10/10/99
Receipt_status: 0

Screen 1 Record: 4
ESC to Record, Ctrl-C to Cancel, F6 to Create Browse Lookup

```

Note that the calculations for subtotal, tax and total still happen correctly.

This is all part and parcel of the special when-leaving code we put in to do the lookup into the Catalog file and retrieve the description and charge. It is special because it uses "associated fields" and "arrays". However, it is not difficult to explain or understand how these features work. The @When Entering and @When Leaving Trigger a0) on lines 70 and 71 happen when the cursor leaves "any" of the fields in the a0) group. When this happens, filePro knows "which" instance of the associated field group was actually left, this is a0). FilePro also assigns the number of this instance to the system maintained field @af. If the cursor just left the second a0) field, @af will be filled with a "2". If the cursor left the last a0) field, @af would be filled with a "4". Because we know this number, we can act upon the same numbered Description and Charge fields along the same line item in their turn.

To help do this, an "array" is built on each of these associated field groups. Arrays can be "overlaid" onto real fields starting at any field number. When you build an array, you use the DIM command. (You DIMension the array to have so many elements.) In our case, we are building three arrays, each with 4 elements. These arrays are overlaid or "aliased" to the real fields that represent Catalog#'s, Descriptions and Charges. The arrays are given names that help identify them to us. If we refer (on the processing table) to chg["2"], we mean the second element in that array, or the second field from the one on which it was overlaid. If we change chg["2"], it is the same as changing the real field just as if we had done it using the field number itself. They both are the same field. In other words, using field 26 or chg["2"] is exactly the same thing.

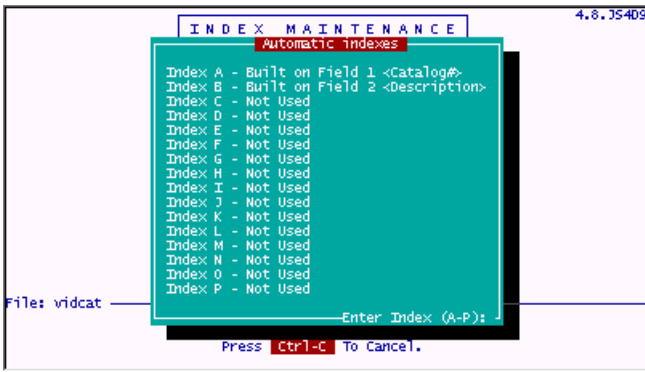
The power of arrays lies in the fact that we can use an expression for the element number in the array. In other words, we can use chg[aa], where field aa is equal to "3", and be referring to chg["3"] or field 27. This is exactly what is being done in the Catalog# lookup. We know which associated field instance the cursor left (1 through 4) because filePro fills @af with this number for us automatically. We can then manipulate the other fields across this line item by referring to them as des[@af] and chg[@af]. These fields will be the same distance down in their respective arrays as the field the cursor left in the Catalog array.

The combination of arrays and associated fields let us write the code once, and use it on four different lines, all because we can say @wlf(a0) (associated field) and because we can alias a named array to real fields.

However, even with all this elegant coding, there is a small fly in the ointment. How often would you think the Catalog# of any particular video gets "rubbed off" or "scribbled over". Probably, more often than not! It is important that we be able to lookup these Catalog items by their name (description) as well as their number. This will require some more code, but not much, just a browse lookup.

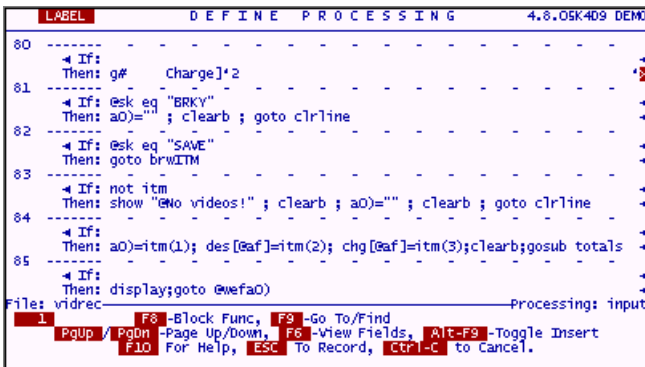
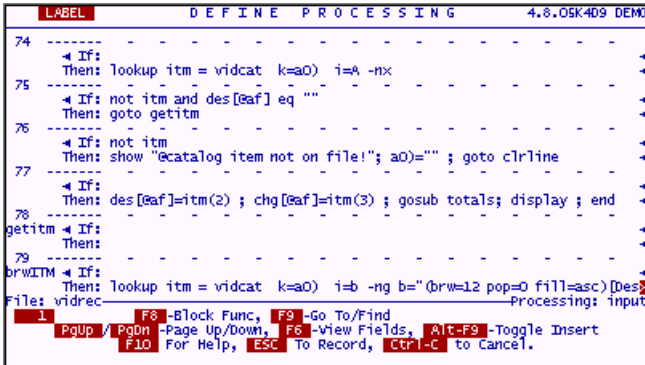
It will also require a new index in the Catalog file built on Description.

Go into Index Maintenance on the "vidcat" file and add the following.



To look into the Catalog file based on description, we only need to add a simple browse lookup.

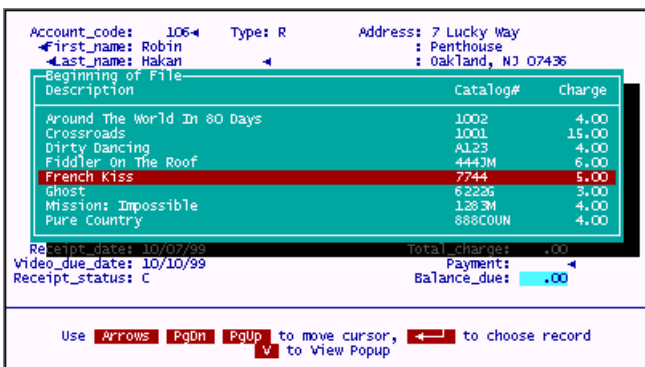
Modify the "vidrec/input" processing table to include the following two screens.



Try out this code now.

Enter the following in Add Records mode in the "vidrec" file.

Call up Account# 106 , and then enter an "f" in the first Catalog# field as follows.



You will see the following. The display starts from the first "f" on file and extends alphabetically from there in both directions.

```

Account_code: 106 Type: R Address: 7 Lucky Way
First_name: Robin : Penthouse
Last_name: Hakan : Oakland, NJ 07436
Phone: (201) 777-7777

Catalog# Description Charge
7744 French Kiss 5.00
Late_charge:
Subtotal: 5.00
Tax: .28
Total_charge: 5.28
Payment:
Balance_due: 5.28

Receipt_date: 10/07/99
Video_due_date: 10/10/99
Receipt_status: 0

Screen 1 Records: 3
ESC to Record, Ctrl-C to Cancel, F6 to Create Browse Lookup

```

Move your highlighted bar down to this record and press ENTER.

```

Account_code: 106 Type: R Address: 7 Lucky Way
First_name: Robin : Penthouse
Last_name: Hakan : Oakland, NJ 07436
Phone: (201) 777-7777

Beginning of File
Description Catalog# Charge
Around The World In 80 Days 1002 4.00
Crossroads 1001 15.00
Dirty Dancing A123 4.00
Fiddler On The Roof 4443M 6.00
French Kiss 7744 5.00
Ghost 62225 3.00
Mission: Impossible 1283M 4.00
Pure Country 888COUN 4.00

Receipt_date: 10/07/99 Total_charge: .00
Video_due_date: 10/10/99 Payment:
Receipt_status: C Balance_due: .00

Use Arrows PgDn PgUp to move cursor, ← to choose record
V to View Popup

```

You will see the following pop into place.

```

Account_code: 106 Type: R Address: 7 Lucky Way
First_name: Robin : Penthouse
Last_name: Hakan : Oakland, NJ 07436
Phone: (201) 777-7777

Catalog# Description Charge
7744 French Kiss 5.00
Late_charge:
Subtotal: 5.00
Tax: .28
Total_charge: 5.28
Payment:
Balance_due: 5.28

Receipt_date: 10/07/99
Video_due_date: 10/10/99
Receipt_status: 0

Screen 1 Records: 3
ESC to Record, Ctrl-C to Cancel, F6 to Create Browse Lookup

```

Feel free to experiment with this lookup. Note that if you do enter a valid Catalog#, the browse lookup is bypassed and the correct data is simply retrieved onto the line item.

The Sort/Select Processing Table (-v processing)

The sort/selection processing provided by filePro is one of its most powerful features. This function allows you to make one pass at the file before running the actual output format. During this first pass, you can select only those records desired based on virtually any criteria or processing. You can also override the default sorting order of the specified output format from within this table. Sorting can be performed on dummy fields, expressions as well. This can not be done in output processing.

Probably the main benefit of Sort/Select tables are their ability to provide users with an ad-hoc look and feel. You can prompt the user for values, and use these values to control all aspects of how the output is completed. This is much different than running an output format with a simple (or even complex) selection set.

Because you are making a first complete pass of the records, you can accumulate values on the sort/selection table and use these "totals" during the actual report pass. For example, you can accumulate the total of a particular field during the sort/select phase and show the percentage of each record against this total on the output phase. This would be impossible (well, difficult at best) using only output processing as you do not know the total of all the records as you are stepping down through the records to produce your output.

Some people call sort/selection tables V tables or -V tables, because this is how you implement these tables. On the command line or menu action line, you would do something like this:

```
.../dreport filename -foutputformat -v sortselecttable -a -u
```

This tells filePro to run the -v table first against all records and hand the selected records over to the output phase.

The SELECT command is the single unique most important function on any sort/select table. If you want to include any record in the group handed to the output phase, you must SELECT this record on the -v table. This means the process must actually move through a processing element where the "if" condition is TRUE and there is a SELECT on the "then" line. Records not so SELECTED do not show up in the output.

Here is a simple sort/selection table. Enter the code from the following two screens in the "vidrec" file on a new output processing table called "seldates".

```

      LABEL                D E F I N E   P R O C E S S I N G
-----
1  -----
   < If: bd ne ""
   Then: goto sel
2  -----
askbd < If:
   Then: input popup bd(8,mdy/,g) "What starting date? (mm/dd/yy) "
3  -----
   < If: @sk eq "BRKY"
   Then: exit
4  -----
   < If: bd eq ""
   Then: goto askbd
5  -----
asked < If:
   Then: input popup ed(8,mdy/,g) "what ending date? (mm/dd/yy) "
6  -----
   < If: @sk eq "BRKY"
   Then: exit
File: VIDREC                1 -----Processing: SELDATES
      F8 -Block Func,  F9 -Go To/Find
      PgUp / PgDn -Page Up/Down,  F6 -View Fields,  Alt-F9 -Toggle Insert
      F10 For Help,  ESC To Record,  BREAK to Cancel.
```

```

L A B E L           D E F I N E   P R O C E S S I N G
-----
7 -----
  ◀ If: ed eq ""
  Then: goto asked
8 -----
  ◀ If: ed lt bd
  Then: show "@End date can't be earlier than begin date!";goto asked
9 -----
sel ◀ If: Receipt_date ge bd and Receipt_date le ed
  Then: select ; end
10 -----
  ◀ If:
  Then: end
11 -----
  ◀ If:
  Then:
12 -----
  ◀ If:
  Then:
File: VIDREC----- 1 -----Processing: SELDATES
      F8 -Block Func, F9 -Go To/Find
      PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.

```

In order to use any dummy fields assigned on a sort/selection table on your output processing table, they must be global dummy fields, and they must be placed on the AUTO table as well. The AUTO table makes them available to, or passes them through to the output processing table.

Build a new output table in "vidrec" called "autorepl". Enter the following data on it.

```

L A B E L           D E F I N E   P R O C E S S I N G
-----
1 -----
  ◀ If:
  Then: bd(8,mdy/,g) ; ed(8,mdy/,g)
2 -----
  ◀ If:
  Then:
3 -----
  ◀ If:
  Then:
4 -----
  ◀ If:
  Then:
5 -----
  ◀ If:
  Then:
6 -----
  ◀ If:
  Then:
File: VIDREC----- 1 -----Processing: autorepl
      F8 -Block Func, F9 -Go To/Find
      PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.

```

We are going to use the BALDUE report we built earlier in the "vidcust" file. We are going to use it, however, in the "vidrec" file. Here is how to copy that format out of "vidcust" into "vidrec". Just mimic the steps on the next several screens.

First, bring up "vidcust/baldue" in Define Output.

```

          D E F I N E   O U T P U T   F O R M A T S
-----
10      20      30      40      50      60      70      80
.....|
HEADING / TITLE LINES
Balance Due Report                               Date: *@td
                                                Time: *@tm
                                                Page: <@pn

Name                               Phone                               Balance Due
-----|-----|-----
DATA LINES
*2      <3      *9      *16

T O T A L   L I N E S
G R A N D   T O T A L
Grand Total: =16
E N D   O F   F O R M
File: VIDCUST-----Format: BALDUE
Enter Selection >
F10 -Help, U -Update, S -Select Format, F -Select File,
C -Copy Format, H -Hardcopy, D -Delete Formats, X -Exit

```

While it is on the screen, press F to change files and choose "vidrec". Press ENTER.


```

DEFINE OUTPUT FORMATS
10      20      30      40      50      60      70      80
.....|.....|.....|.....|.....|.....|.....|.....|
HEADING / TITLE LINES
Balance Due Report                               Date: *@td
                                                Time: *@tm
                                                Page: <@pn

BRODY STATES
CLASS TEST
FPCUST VIDCAT
FPINVC VIDCTRL
FPPROD VIDCUST
FPSTIAX VIDREC
INVC XINV
JOHN XIRLN

Phone Balance Due
-----|-----|-----|
DATA LINES
*9 *16

TOTAL LINES
GRAND TOTAL
Grand Total: =16
END OF FORM

File: VIDCUST-----Format: BALDUE

Select file...
Enter file name:

```

You will be brought into the "vidrec" file carrying this format, but it will have no name.

```

DEFINE OUTPUT FORMATS
10      20      30      40      50      60      70      80
.....|.....|.....|.....|.....|.....|.....|.....|
HEADING / TITLE LINES
Balance Due Report                               Date: *@td
                                                Time: *@tm
                                                Page: <@pn

Name Phone Balance Due
-----|-----|-----|
DATA LINES
*2 <3 *9 *16

TOTAL LINES
GRAND TOTAL
Grand Total: =16
END OF FORM

File: VIDREC-----
Enter Selection >
F10 -Help, U -Update, S -Select Format, F -Select File,
C -Copy Format, H -Hardcopy, D -Delete Formats, X -Exit

```

Press C and [NEW] to Copy this format to a new name in this file.

```

DEFINE OUTPUT FORMATS
10      20      30      40      50      60      70      80
.....|.....|.....|.....|.....|.....|.....|.....|
HEADING / TITLE LINES
Balance Due Report                               Date: *@td
                                                Time: *@tm
                                                Page: <@pn

Name Phone Balance Due
-----|-----|-----|
DATA LINES
*2 <3 *9 *16

TOTAL LINES
GRAND TOTAL
Grand Total: =16
[NEW] END OF FORM

File: VIDREC-----

Copy format to...
Enter format name:

```

Give it the name "receipts" and press ENTER. You will see the following. The format is now named and ready for use.

```

      DEFINE OUTPUT FORMATS
10      20      30      40      50      60      70      80
.....|
HEADING / TITLE LINES
Balance Due Report                               Date: *@td
                                                Time: *@tm
                                                Page: <@pn

Name                               Phone                               Balance Due
-----|-----|-----
DATA LINES
*2      <3      *9      *16

TOTAL LINES
GRAND TOTAL
Grand Total: =16
END OF FORM

File: VIDREC-----Format: RECEIPTS
Enter Selection >
F10 -Help, U -Update, S -Select Format, F -Select File,
C -Copy Format, H -Hardcopy, D -Delete Formats, X -Exit

```

The fields in this file, "vidrec" are different from those in "vidcust" where this format came from, so you will have to modify the sort of this format to be applicable here. Do this, by pressing U to update the format, then F8 for the options dialog, then S for Sort criteria. Enter the following data.

```

Field list
1- Account_code      11- Subtotal      21- b0) Description
2- First_name       12- Tax           22- b0) Description
3- Last_name        13- Total_charge  23- b0) Description
4- Type_of_account  14- Payment       24- b0) Description
5- r                15- Balance_due   25- c0) Charge
6- Receipt_date     16- r            26- c0) Charge
7- Video_due_date   17- a0) Catalog# 27- c0) Charge
8- Receipt_status   18- a0) Catalog# 28- c0) Charge
9- r                19- a0) Catalog#
10- Late_charge     20- a0) Catalog#

DATA LINES
Sort definition
Sort Field: 6
Length: 8
Descending?
Subtotal Field? x

Press ESC To Record, BREAK To Cancel

Press F10 for Help

```

Modify the output format to look like this.

```

      DEFINE OUTPUT FORMATS
10      20      30      40      50      60      70      80
.....|
HEADING / TITLE LINES
Receipts For                               Date: *@td
*bd      to *ed                             Time: *@tm
                                                Page: <@pn

Name                               Charges Payments Balance Due
-----|-----|-----
DATA LINES
*2      <3      *13      *14      *15

TOTAL LINES
SUBTOTAL BY RECEIPT_DATE
Subtotal for *@sf      =13      =14      =15

GRAND TOTAL
Grand Total =13      =14      =15
END OF FORM

File: VIDREC-----Format: RECEIPTS
Enter Selection >
F10 -Help, U -Update, S -Select Format, F -Select File,
C -Copy Format, H -Hardcopy, D -Delete Formats, X -Exit

```

Put this new report on the "video" menu.

Enter the following data.

ACTION

D E F I N E U S E R M E N U S

```
3 -----
3 3 Receipts File          ◀
  \fp\dclerk vidrec -sl -h "Receipts File"
                                     ▶
4 -----
4 4 Video Catalog File    ◀
  \fp\dclerk vidcat -s0 -h "Catalog File"
                                     ▶
5 -----
5 5 Receipts Report (by date) ◀
  \fp\dreport vidrec -f receipts -v seldates -a -y autorepl -u
                                     ▶
6 -----
  ◀                               ▶
                                     ▶
Menu: VIDEO-----
F5 - Create batch file, ESC - Save, BREAK - Cancel, F10 - Help
```

Use this menu to try the new report.
Choose a date range that will find some of the receipts you've been entering. First, the starting date.

Receipts Report

What starting date? (mm/dd/yy) ◀

Then, the ending date.

Receipts Report

what ending date? (mm/dd/yy) ◀

Press **BREAK** To Exit.

The report should look something like this.

Name	Charges	Payments	Balance Due
Ken Blackburn	20.05		20.05
Subtotal for 04/28/97	20.05	.00	20.05
Robin Hakan	30.60		30.60
Maggie Mae	5.28		5.28
Ken Blackburn	4.22		4.22
Subtotal for 04/29/97	40.10	.00	40.10
Grand Total	60.15	.00	60.15

In many cases, there is a powerful way to enhance sort/selection processing. You can speed up the selection of records from large files GREATLY by using the "lookup-" (lookup dash) feature of filePro.

Change the processing for "vidrec/vdates" to look like the following 4 screens.

```
*****  
ACTION          D E F I N E   P R O C E S S I N G  
-----  
1  -----  
    < If: bd ne ""  
    Then: goto sel  
2  -----  
askbd < If:  
    Then: input ("7","25") bd(8,mdy/,g) "What starting date? (mm/dd/yy) "  
3  -----  
    < If: @sk eq "BRKY"  
    Then: exit  
4  -----  
    < If: bd eq ""  
    Then: goto askbd  
5  -----  
asked < If:  
    Then: input ("13","25") ed(8,mdy/,g) "What ending date? (mm/dd/yy) "  
6  -----  
    < If: @sk eq "BRKY"  
    Then: exit  
File: VIDREC Processing: VDATES  
F8 -Block Func, F9 -Go To/Find  
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert  
F10 For Help, ESC To Record, BREAK to Cancel.
```

```
LABEL          D E F I N E   P R O C E S S I N G  
-----  
7  -----  
    < If: ed eq ""  
    Then: goto asked  
8  -----  
    < If: ed lt bd  
    Then: show "@End date can't be earlier than begin date!";goto asked  
9  -----  
    < If:  
    Then: ky=bd  
10 -----  
sel < If: ky eq "12/31/99"  
    Then: end  
11 -----  
    < If: Receipt_date lt bd  
    Then: lookup - k=ky i=c -ng  
12 -----  
    < If: not -  
    Then: end  
File: VIDREC Processing: VDATES  
F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup  
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert  
F10 For Help, ESC To Record, BREAK to Cancel.
```

```

LABEL                D E F I N E   P R O C E S S I N G
-----
13 -----
  ◀ If: Receipt_date gt ed
  Then: gosub getlast
14 -----
  ◀ If: Receipt_date ge bd and Receipt_date le ed
  Then: gosub getit
15 -----
  ◀ If:
  Then: end
16 -----
getit ◀ If:
      Then:
17 -----
  ◀ If: Balance_due gt "0"
  Then: select
18 -----
  ◀ If:
  Then: return
File: VIDREC-----1-----Processing: VDATES
      F8 -Block Func,  F9 -Go To/Find  F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down,  F6 -View Fields,  Alt-F9 -Toggle Insert
      F10 For Help,  ESC To Record,  BREAK to Cancel.

```

```

LABEL                D E F I N E   P R O C E S S I N G
-----
19 -----
getlast◀ If:
      Then: ky="12/31/99"
20 -----
  ◀ If:
  Then: lookup - k=ky i=c -nl
21 -----
  ◀ If:
  Then: return
22 -----
  ◀ If:
  Then:
23 -----
  ◀ If:
  Then:
24 -----
  ◀ If:
  Then:
File: VIDREC-----1-----Processing: VDATES
      F8 -Block Func,  F9 -Go To/Find  F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down,  F6 -View Fields,  Alt-F9 -Toggle Insert
      F10 For Help,  ESC To Record,  BREAK to Cancel.

```

Change the menu item for this report to read as follows.

```

ACTION                D E F I N E   U S E R   M E N U S
-----
3 -----
3◀ Receipts File
  \fp\dclerk vidrec -sl -h "Receipts File"
4 -----
4◀ Video Catalog File
  \fp\dclerk vidcat -s0 -h "Catalog File"
5 -----
5◀ Receipts Report (by date)
  \fp\dreport vidrec -f receipts -v vdates -ic -a -y autorepl -u
6 -----
  ◀
Menu: VIDEO-----
      F5 - Create batch file,  ESC - Save,  BREAK - Cancel,  F10 - Help

```

Try this code out, however, you will not see ANY speed difference with only a handful of records. When you run such a "lookup dash" table against a file with many thousands of records, you will be astonished at the speed increase in selecting records over conventional methods.

Receipts Report

What starting date? (mm/dd/yy) 04/27/97

What ending date? (mm/dd/yy) 05/03/97

Press BREAK To Exit.

A brief description of how lookup dash works on the sort/select table is this. Lookup dash "moves" you to the record indicated by the lookup key. The first test tells filePro to move you to the first record that meets the criteria or the next highest match. Since we are using an index to run this output, each record will come in order and the process will continue SELECTING records that fall within the criteria. As soon as the first record that does not match the criteria is reached, filePro is told to do a lookup dash to the very end of the index. It does this by using a key that is the highest possible value for the index and then finding an equal or next lowest match to this. By doing this, you are guaranteed to find the very last record in any index. Since this record will NOT match the original criteria, the process just ends and there are no more records to process. By cutting out only the records that match the criteria from an entire index, vast savings in time are obtained.

Let's build another sort/selection table. On this one, we will accumulate a total of the Balance_Due fields selected so it can be used on the output processing table. We will also query the user as to the status of records for which the report is to be pulled, (O)pen or (C)losed.

Enter the code from the next 3 screens onto an output table in "vidrec" called "vstatus".

```

L A B E L           D E F I N E   P R O C E S S I N G
-----
1  -----
   < If: st ne ""
   Then: goto sel
2  -----
askst < If:
   Then: input ("7","25") st(1,allup,g) "What status type? (C/O) "
3  -----
   < If: @sk eq "BRKY"
   Then: exit
4  -----
   < If: st eq ""
   Then: goto askst
5  -----
   < If: st ne "0" and st ne "C"
   Then: goto askst
6  -----
   < If:
   Then: ky=st
File: VIDREC-----1-----Processing: VSTATUS
      F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.

```

```

L A B E L           D E F I N E   P R O C E S S I N G
-----
7  -----
sel < If: st eq "~"
   Then: end
8  -----
   < If: Receipt_status lt st
   Then: lookup - k=ky i=d -ng
9  -----
   < If: Receipt_status gt st
   Then: gosub getlast
10 -----
   < If: Receipt_status eq st
   Then: gosub getit
11 -----
   < If:
   Then: end
12 -----
getit < If: bd eq ""
   Then: bd(8,mdy/,g)=Receipt_date ; ed(8,mdy/,g)=Receipt_date
File: VIDREC-----1-----Processing: VSTATUS
      F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.

```

```

LABEL                D E F I N E   P R O C E S S I N G
-----
13 -----
    ◀ If: Receipt_date lt bd
    Then: bd=Receipt_date
14 -----
    ◀ If: Receipt_date gt ed
    Then: ed=Receipt_date
15 -----
    ◀ If:
    Then: tt(7,.2,g)=tt+"1" ; select ; return
16 -----
getlast◀ If:
    Then: ky="~"
17 -----
    ◀ If:
    Then: lookup - k=ky i=d -nl
18 -----
    ◀ If:
    Then: return
File: VIDREC-----1-----Processing: VSTATUS
    F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
    PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
    F10 For Help, ESC To Record, BREAK to Cancel.

```

Add the following to the "vidrec/autorepl" table, so that the variable "st" can be passed to the output processing table.

```

CONDITION           D E F I N E   P R O C E S S I N G
-----
1 -----
    ◀ If:
    Then: bd(8,mdy/,g) ; ed(8,mdy/,g)
2 -----
    ◀ If:
    Then: st(1,,g);tt(7,.2,g)
3 -----
    ◀ If:
    Then:
4 -----
    ◀ If:
    Then:
5 -----
    ◀ If:
    Then:
6 -----
    ◀ If:
    Then:
File: VIDREC-----1-----Processing: AUTOREPL
    F8 -Block Func, F9 -Go To/Find
    PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
    F10 For Help, ESC To Record, BREAK to Cancel.

```

Put the following on the "video" menu, so we can test this report.

```

ACTION                D E F I N E   U S E R   M E N U S
-----
3 -----
3◀ Receipts File
  \fp\dclerk vidrec -sl -h "Receipts File"
4 -----
4◀ Video Catalog File
  \fp\dclerk vidcat -s0 -h "Catalog File"
5 -----
5◀ Receipts Report (by date)
  \fp\dreport vidrec -f receipts -v vdates -ic -a -y autorepl -u
6 -----
6◀ Receipts Report (by status)
  \fp\dreport vidrec -f receipts -v vstatus -id -a -y autorepl -u
Menu: VIDEO-----
    F5 - Create batch file, ESC - Save, BREAK - Cancel, F10 - Help

```

Before running this output, we must modify the format "vidrec/receipts" and its processing table to calculate and display the percentage each record's Balance_Due is of the total.

First, add the following to "vidrec/receipts".

```

LABEL                D E F I N E   P R O C E S S I N G
-----
1  -----
   ◀ If:
   Then: pp(6,.2)=(15/tt) * "100"
2  -----
   ◀ If:
   Then: p(7)=pp4"%"
3  -----
   ◀ If:
   Then: end
4  -----
   ◀ If:
   Then:
5  -----
   ◀ If:
   Then:
6  -----
   ◀ If:
   Then:
File: VIDREC-----1-----Processing: RECEIPTS
      F8 -Block Func,  F9 -Go To/Find
      PgUp / PgDn -Page Up/Down,  F6 -View Fields,  Alt-F9 -Toggle Insert
      F10 For Help,  ESC To Record,  BREAK to Cancel.

```

Now modify the report format itself to show the percentage.

```

      D E F I N E   O U T P U T   F O R M A T S
-----
      10      20      30      40      50      60      70      80
.....|-----|-----|-----|-----|-----|-----|-----|
      H E A D I N G / T I T L E   L I N E S
Receipts For                               Date: *@td
*bd      to *ed                             Time: *@tm
                                           Page: <@pn

Name                                         Charges  Payments  Balance Due  %ofTotal
-----
      D A T A   L I N E S
*2      <3                               *13      *14      *15      *p
      T O T A L   L I N E S
      S U B T O T A L   B Y   R E C E I P T   D A T E
      Subtotal for *@sf      =13      =14      =15
      G R A N D   T O T A L
      Grand Total      =13      =14      =15
      E N D   O F   F O R M
File: VIDREC-----Format: RECEIPTS
      Enter Selection >
      F10 -Help,  U -Update,  S -Select Format,  F -Select File,
      C -Copy Format,  H -Hardcopy,  D -Delete Formats,  X -Exit

```

Use the menu item we added previously, to try the report.

```

      R E Q U E S T   O U T P U T
-----

```

```

What status type? (C/O) ◀

```

```

Press BREAK To Exit.

```

It should look something like this.

Receipts For
04/28/97 to 04/29/97

Date: 04/30/97
Time: 22:55:43
Page: 1

Name	Charges	Payments	Balance Due	%ofTotal
Ken Blackburn	20.05		20.05	35.85%
Subtotal for 04/28/97	20.05	.00	20.05	
Robin Hakan	30.60		30.60	54.71%
Robin Hakan	5.28		5.28	9.44%
Subtotal for 04/29/97	35.88	.00	35.88	
Grand Total	55.93	.00	55.93	

A Virtual Work File - A Place To Stand

Aristotle once said, "Give me a place to stand and a lever big enough, and I could move the Earth." The concept is possibly true and whether or not it is, there is meaning here for filePro application developers. The lever is filePro, big enough to do almost any task, and the knowing "place to stand" is the key to making it work. Up until this point in the tutorials, we have been standing in one real file or another and adding, modifying and deleting records from that file or others. There are drawbacks to this scenario, but there are also things you can do about them.

The most fundamental concern for application developers is data integrity. Add to this, efficiency and usability and the problems of design are big enough to start with; but bring in the aspects and needs of a "multi-user" system, and the problems begin to really mount up. The simplest fact that two users can not be allowed to update the same record at the same time can cause everything from little hassles to wide spread calamity. In the case of our little Video Store system, multi-user problems could arise at many stages. To virtually eliminate this, we can use "virtual" files. What are these? Simple, they are nothing more than an Aristotelian "place to stand". A dummy file, from which, the application user can reach out and access all other real filePro files in the most non-invasive way. In other words, working from this "virtual" or "dummy" file one can have the most access while benefiting from the least impact caused by mandatory "record locking". All this to say, we are going to build a fake filePro file and stand in it to add customers, receipts (and catalog items) to the system from there. By doing this, we won't be tying up the real records with various filePro functions and commands that would otherwise give us serious problems.

Consider these three situations.

1) Our Video Store app works from within the customer file "vidcust". A customer brings some videos up to a register for rental. The store clerk begins updating the customer's account record, adding receipts, taking payments, etc. While he is doing this, the customer's wife goes up to a register on the other side of the store and wants to buy a DVD. The second register operator can not update the family account record, because it is already in use by the first register operator. Not a good situation.

2) Our Video Store app works from within the receipts file "vidrec". This is a little better, since multiple receipts can be added simultaneously to the same customer account. Assuming, that is, the customer record is only locked just long enough to do critical postings. The problem is solved, right? No. What if during the day, the store manager needs to run a report showing all receipts up to the minute. If any receipt record is being updated, the report will freeze on that record and wait for it to be unlocked. Not a good situation.

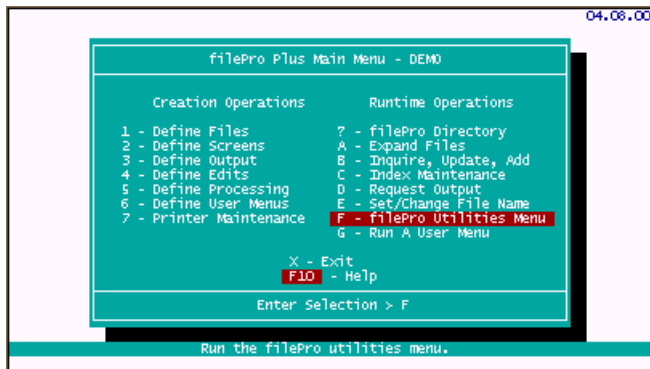
3) After using the "How Do I" section of this manual, you start adding some of the great features you've learned about, like the MENU function, to your programs. You don't realize that the MENU command is being run from within the INPUT processing table and therefore has the record open and locked! Nobody else can access this record for updating, or even read past it with a report, while the record is in this state. The MENU created through processing, looks so much like a regular filePro menu that one of your users thinks nothing of leaving this menu on the screen while he goes to lunch. Not a good situation, and all you were trying to do was make use of some of the powerful tools within filePro.

All these types of multi-user problems can be avoided by building a "work" file. This just means making a filePro file that you can use to "stand on" while you operate an application. We will do this by copying the "vidcust" file to a new file called "vidwork" and rewriting the application we have written so far to work from within that file. The actual procedure will be to enter this work file in Add Records mode, thereby landing on a free record. While on this free record, we can do all the operations necessary to run our program, add customers, receipts, etc. When we are done running the application, we delete the record we were standing on and back out of the file. No record in any real filePro file is locked for more than the time it takes to post to it (microseconds) and still our application works just fine.

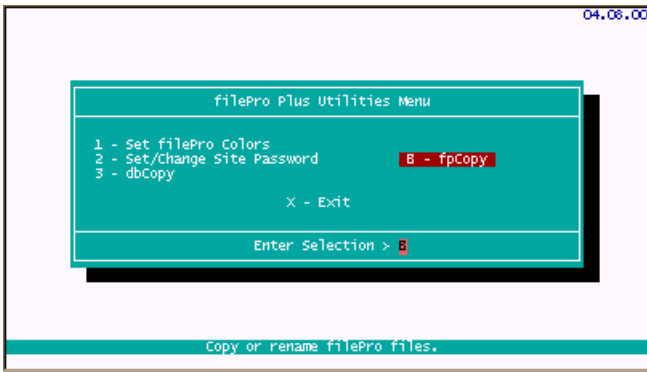
Here is how to turn this application into a better program by using a "virtual" work file.

First, we will need to use "fpcopy" to duplicate the "vidcust" file.

Select F



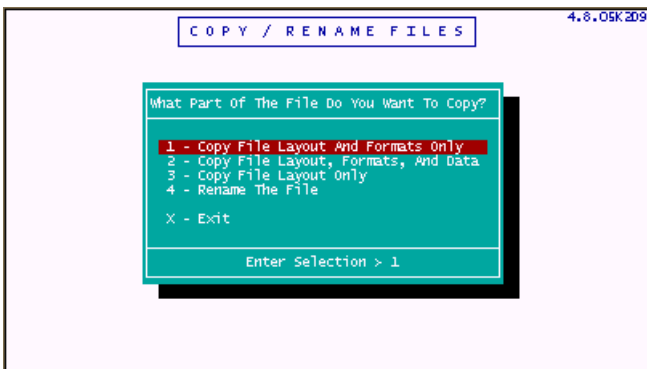
Select B



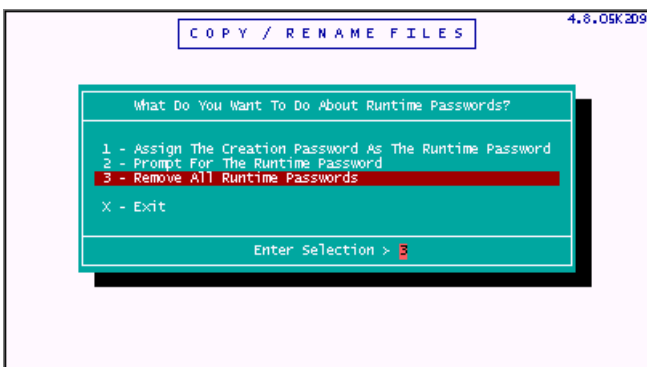
Enter the following data.



Select 1



Select 3



Go into Define Screens on the file "vidwork".

Copy Screen.0 to Screen.1 and delete Screen.0.

Modify the new Screen.1 to look like this.

```

VIDECUST
Account_code: '1
First_Name: !2
Last_Name: '3
Address1: !4
Address2: !5
City: !6
State: !7
Zip: !8
Phone: !9
Customer_type: !11
First_sale_date: !12
Charges: !14
Payments: !15
Balance_due: !16
File: vidwork-----Color Screen: 1
Enter Selection >
F10 -Help, U -Update, S -Select Screen, F -Select File,
C -Copy Screen, H -Hardcopy, D -Delete Screens, X -Exit

```

We can modify a screen from the "vidrec" file for use in the new "vidwork" file.

Remain in Define Screens. **Change to the "vidrec" file** by pressing **F** and choosing **"vidrec"**.

Press S to select screens and **choose Screen.1**.

```

Account_code: '1      Type: !4      Address: 'aa
First_name: '2       : 'ab
Last_name: '3        : 'ac
                        Phone: 'ad

Catalog#  Description  Charge
-----
'17      '21          '25
'18      '22          '26
'19      '23          '27
'20      '24          '28
                        Late_charge: '10
Subtotal: !11
Tax: !12
Total_charge: !13
Payment: '14
Balance_due: !15
Receipt_date: !6
Video_due_date: !7
Receipt_status: !8
File: vidrec-----Color Screen: 1
Enter Selection >
F10 -Help, U -Update, S -Select Screen, F -Select File,
C -Copy Screen, H -Hardcopy, D -Delete Screens, X -Exit

```

Change back to the "vidwork" file by pressing **F** and **choosing "vidrec"**.

Select C for Copy and then choose **[NEW]**. Give this currently "unamed" screen the name **"rec"**.

```

Account_code: '1      Type: !4      Address: 'aa
First_name: '2       : 'ab
Last_name: '3        : 'ac
                        Phone: 'ad

Catalog#  Description  Charge
-----
'17      '21          '25
'18      '22          '26
'19      '23          '27
'20      '24          '28
                        Late_charge: '10
Subtotal: !11
Tax: !12
Total_charge: !13
Payment: '14
Balance_due: !15
Receipt_date: !6
Video_due_date: !7
Receipt_status: !8
File: vidwork-----Color Screen: rec
Enter Selection >
F10 -Help, U -Update, S -Select Screen, F -Select File,
C -Copy Screen, H -Hardcopy, D -Delete Screens, X -Exit

```

Your screen should look like this.

```

Account_code: '1      Type: !4      Address: 'aa
First_name: !2       : 'ab
Last_name: '3        : 'ac
                        Phone: 'ad

Catalog#  Description  Charge
-----
'17      !21          !25
'18      !22          !26
'19      !23          !27
'20      !24          !28
                        Late_charge: '10
Subtotal: !11
Tax: !12
Total_charge: !13
Payment: '14
Balance_due: !15
Receipt_date: !6
Video_due_date: !7
Receipt_status: !8
File: vidwork-----Color Screen: rec
Enter Selection >
F10 -Help, U -Update, S -Select Screen, F -Select File,
C -Copy Screen, H -Hardcopy, D -Delete Screens, X -Exit

```

Modify this screen to look as follows.

```

Account_code: '1      Type: 14      Address: 14
First_name: 12      : 15
Last_name: 13      : 1rv
                  Phone: 19

Catalog#  Description  Charge
-----
*ra      lre          lri
*rb      lrf          lrj
*rc      lrg          lrk
*rd      lrh          lr1
                  Late_charge: *rm
                  Subtotal: lrn
                  Tax: lro
Receipt_date: lrs    Total_charge: lrp
Video_due_date: lrt  Payment: *rq
Receipt_status: lru  Balance_due: lrr

File: vidwork-----Color Screen: rec
Enter Selection >
F10 -Help, U -Update, S5 -Select Screen, F7 -Select File,
C -Copy Screen, H -Hardcopy, D -Delete Screens, X -Exit

```

We will add the processing for the "vidwork" file in two stages for testing purposes.

Add the 49 lines of code from the following 9 screens to the "vidwork/input" processing table.

```

LABEL          DEFINE  PROCESSING  4.8.05K409 DEMO
-----
1 -----
  < If:
  Then: delete ; end
2 -----
@wef1 < If: tx eq ""
  Then: gosub gettax
3 -----
  < If:
  Then: cls("22")
4 -----
  < If: 1 eq ""
  Then: showctr("22") "Enter Acct# or press \r \KZ \r to use Acct Name."
5 -----
  < If:
  Then: esljend
6 -----
gettax < If:
  Then: r(1,.0,g)="1"
File: vidwork-----Processing: input
1 -----
  F8 -Block Func, F9 -Go To/Find
  PgUp / PgDn -Page Up/Down, F6 -View Fields, ALT-F9 -Toggle Insert
  F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

```

LABEL          DEFINE  PROCESSING  4.8.05K409 DEMO
-----
7 -----
  < If:
  Then: lookup vidctr1 r=r -n
8 -----
  < If: not vidctr1
  Then: show "No control file!!!"; return
9 -----
  < If:
  Then: tx(4,.3,g)=vidctr1(2) ; return
10 -----
@wif1 < If:
  Then:
11 -----
  < If: 1 eq ""
  Then: gosub clrall;end
12 -----
  < If:
  Then: lookup addr = vidcust k=1 i=b -nx
File: vidwork-----Processing: input
1 -----
  F8 -Block Func, F9 -Go To/Find, F5 -To Define A Lookup
  PgUp / PgDn -Page Up/Down, F6 -View Fields, ALT-F9 -Toggle Insert
  F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

```

LABEL          DEFINE  PROCESSING  4.8.05K409 DEMO
-----
13 -----
  < If: not addr
  Then: show "Customer code not on file!" ; gosub clrall ; screen ,1
14 -----
  < If:
  Then: copyin addr
15 -----
  < If:
  Then: display ; end
16 -----
@wef3 < If:
  Then: cls("22")
17 -----
  < If: 3 eq ""
  Then: showctr("22") "Enter a period (.) to add a new customer."
18 -----
  < If: 3 eq ""
  Then: showctr("23") "Enter start of name, or \r \KZ \r to use Acct#."
File: vidwork-----Processing: input
1 -----
  F8 -Block Func, F9 -Go To/Find, F5 -To Define A Lookup
  PgUp / PgDn -Page Up/Down, F6 -View Fields, ALT-F9 -Toggle Insert
  F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

```

LABEL          DEFINE PROCESSING          4.8.05K409 DEMO
19 -----
  < If:
  Then: e=3; end
20 -----
@wlf3 < If:
  Then: showctr("22") ""
21 -----
  < If: 3 co ""
  Then: 3="";gosub c1rall; gosub adnew; screen 1,3
22 -----
  < If: 3 eq e and e ne ""
  Then: end
23 -----
  < If: 3 eq ""
  Then: gosub c1rall ; end
24 -----
  < If:
  Then: gosub prompt
File: vidwork-----Processing: input
1  F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt+F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

(The full lookup wizard screens for lines 25-26 are shown after this screen. Then, the processing screens continue.

```

LABEL          DEFINE PROCESSING          4.8.05K409 DEMO
25 -----
brwNAME < If:
  Then: lookup_addr = vidcust k=3 i=a -ng b="" (brw=L2 show=pkeep pop=0
26 -----
  < If:
  Then: Phone]*2 <3 *4
27 -----
  < If: ask eq "BRKY"
  Then: c1earb ; 3 = "" ; display ; screen ,3
28 -----
  < If: ask eq "SAVE"
  Then: goto brwNAME
29 -----
  < If: not addr
  Then: show "No names" ; c1earb ; 3=""; display ; screen ,3
30 -----
  < If:
  Then: copyin addr ; c1earb
File: vidwork-----Processing: input
1  F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt+F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

```

DEFINE LOOKUP DEMO
-----
Lookup From File: vidcust
Name of Lookup Is: addr
How Is The Record To Be Found? k K - Key Field R - Record Number
F - Free Record Z - Fuzzy Search
What Index Is To Be Used? a F6 for list
What Field In 'vidrec' 3
Contains The Key?
Protect Record in Lookup File? N
If Key Doesn't Match Exactly: g
X - Key Must Match Exactly
G - Use Next Greater Match
L - Use Next Lower Match
If Lookup Fails: n B - Blank The Field
N - Do Nothing
E - Report An Error
Create Browse Lookup? Y
Press Ctrl-C To Cancel Changes

```

```

DEFINE LOOKUP DEMO
-----
Browse Header:
Name Adress Phone
Browse Format: *2 *4 *9
Browse Windows: Lines: 12 Row: Col:
Pop-Up Screen: Name: 0 Row: Col:
Only Show Records That Match Key? N Y-Yes, N-No, P-show Partial match
Browse Window Show option: P 0 - Show Only. Accept no keystrokes.
(Leave blank for default) K - Keep Browse Window on Screen.
P - Retain Position on re-execution.
Exit Keys:
(Leave blank for none)
Order of Records (Ascending, Descending): A
Position of First Record (Top,Middle,Bottom): M
Processing Label:
Press Ctrl-C To Cancel Changes

```

```

LABEL          DEFINE PROCESSING          4.8.05K409 DEMO
31 -----
   If:
   Then: display ; screen ,3
32 -----
address If:
   Then: cls("22"); gosub prompt2
33 -----
   If:
   Then: lookup new = vidcust r=free -ep
34 -----
   If:
   Then: popup update new,"add" ; clearp
35 -----
   If: @sk eq "BRKY"
   Then: write new ; delete new ; screen ,3
36 -----
   If:
   Then: lookup num = vidctrl r=r -np
File: vidwork-----Processing: input
1 F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

```

LABEL          DEFINE PROCESSING          4.8.05K409 DEMO
37 -----
   If: not num
   Then: show "No control file!!!" ; write new ; delete new ; screen ,3
38 -----
   If:
   Then: new(1)=num(1); num(1)=num(1)+1; write num
39 -----
   If:
   Then: new(16)=new(14)-new(15)
40 -----
   If:
   Then: 1=new(1); 2=new(2); 3= new(3); 4 = new(4)
41 -----
   If:
   Then: 5=new(5); 6=new(6) ; 7=new(7); 8=new(8); 9=new(9)
42 -----
   If:
   Then: 11=new(11); 12=new(12); 14=new(14); 15=new(15); 16=new(16)
File: vidwork-----Processing: input
1 F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

```

LABEL          DEFINE PROCESSING          4.8.05K409 DEMO
43 -----
   If:
   Then: write new
44 -----
   If:
   Then: return
prompt If:
   Then: cls("22")
46 -----
   If:
   Then: showctr("23") "Use ARROWS, NxtPag/PrvPg to scroll. Press \K2 "
47 -----
prompt2 If:
   Then: showctr("22") "Press \r K4 \r to record, \r \KY \r to cancel"
48 -----
   If:
   Then: return
File: vidwork-----Processing: input
1 F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

```

LABEL          DEFINE PROCESSING          4.8.05K409 DEMO
49 -----
clrall If:
   Then: dim clrit[16]:1 ; clear clrit ; display ; return
50 -----
   If:
   Then:
51 -----
   If:
   Then:
52 -----
   If:
   Then:
53 -----
   If:
   Then:
54 -----
   If:
   Then:
File: vidwork-----Processing: input
1 F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

Put this Work File on the menu "video".

Enter the following data.

```

5 -----
5  Receipts Report (by date)      ◀
  \fp\dreport vidrec -f receipts -v vdates -ic -a -y autorepl -u      ▶
6 -----
6  Receipts Report (by status)   ◀
  \fp\dreport vidrec -f receipts -v vstatus -id -a -y autorepl -u    ▶
7 -----
7  Work File                     ◀
  \fp\dclerk vidwork -sl -xa -d                                       ▶
8 -----
◀
▶
▶
Menu: VIDEO
F5 - Create batch file,  ESC - Save,  BREAK - Cancel,  F10 - Help

```

Notice that the command line includes `-xa` and `-d`. These switches tell filePro to go immediately into Add Records mode (`-xa`) and to clear the standard filePro prompts (H-Hardcopy, B-Browse, U-Update... etc.) from the bottom of the screen.

The processing code we put in "vidwork/input" does essentially the same thing as was first done in the receipts file. It lets us lookup a specific customer's Account# and pull in the customer info. It lets us browse the customer file by name and pull in the customer info and Account#. It lets us add a new customer. This table also brings the "taxrate" lookup from the automatic processing onto this table, since this program has no need to run any automatic processing.

The only unique and new concept on this table is line 1. Since we are in Add Records mode when running this application, we will be sitting on the first available free record. When the user presses ESC, this line will DELETE the record we are standing on and END. The Add Records mode will put us on the next available free record again, which is the one we just deleted! We can stay in one place "treading water" in this file as long as need be. So can any number of other users on their own individual "place to stand" temporary record. This is because while we are standing on our record, it is locked and can not be given to anyone else. Other users coming into this file in Add Records mode get their own record on which to "tread water". Once you become familiar with this idea, it will become a valuable and useful mechanism in your filePro toolkit.

It's time to try this code.

Use the menu selection to go into the virtual file "vidwork".

First try the Account# lookup. Enter Account# 103, you should see something like the following pop into place.

```

Video Store
-----
Account_code: 103◀
First_Name: Ken
Last_Name: Blackburn ◀
Address: 20194 Applegate Drive
Address: Apt. 2E
City: Oakland
State: NJ
Zip: 07436
Phone: (201) 347-2828

Customer_type: R
First_sale_date: 02/12/97

Charges: 102.63
Payments: 102.63
Balance_due: .00

```

Now try the name lookup. Enter the first few letters of Hakan.

7

You will see the following browse window.

Video Store		
Beginning of File		
Name	Address	Phone
Ken Blackburn	20194 Applegate Drive	(201) 347-2828
Robin Hakan	7 Lucky Way	(201) 777-7777
John Jones	1521 Cypress Street	(201) 443-2222
Maggie Mae	301 Ipswitch Road	(201) 429-3838
Mary Rodrigues	97 First Ave.	(201) 928-3838
Karen Smith	111 Happy Lane	(201) 427-9888
End of File		

Press **ESC** to Record, **BREAK** to Cancel
Use ARROWS and NxtPg/PrvPg to scroll. Press **←** to select.

Press **ENTER** and the customer information is popped into place.

```

Video Store
-----
Account_code: 106
First_Name: Robin
Last_Name: Hakan
Address: 7 Lucky Way
Address: Penthouse
City: Oakland
State: NJ
Zip: 07436
Phone: (201) 777-7777

Customer_type: R
First_sale_date: 04/28/97

Charges: 40.10
Payments: .00
Balance_due: 40.10

```

We have not lost the "add new customer" feature. Even though we are standing on a virtual file, the code still works properly from here.

Put a period in the Last_Name field to activate the "add new customer" routine. Enter the following data.

```

Add New Customer Screen

First_Name: William
Last_Name: Stoute
Address: 123 Billings Drive
Address: Suite 22
City: Paterson
State: NJ
Zip: 07555
Phone: 427-2828

Customer_type: S
first_sale_date: 05/03/97

```

Press **ESC** to Record, **BREAK** to Cancel

The next unique Account# is added automatically, and the account is ready to have receipts added to it.

Video Store

```
Account_code: 115
First_Name: William
Last_Name: Stoute
Address: 123 Billings Drive
Address: Suite 22
City: Paterson
State: NJ
Zip: 07555
Phone: 427-2828

Customer_type: S
First_sale_date: 05/03/97

Charges:
Payments:
Balance_due: .00
```

A quick check of the real customer file shows that the record has been stored here permanently.

VIDCUST

```
Account_code: 115
First_Name: William
Last_Name: Stoute
Address: 123 Billings Drive
Address: Suite 22
City: Paterson
State: NJ
Zip: 07555
Phone: 427-2828

Customer_type: S
First_sale_date: 05/03/97

Charges:
Payments:
Balance_due: .00
```

Screen 0 Enter Selection > Record: 7

D-Delete, H-Hardcopy, U-Update, X-Exit, F-Print Form, B-Browse

We will add all the rest of the processing for the virtual "vidwork" file now.

Much of this table is the same as the processing we wrote for "vidrec/input", except that lookup assignments are made to screens made up of dummy fields. At the appropriate times, these dummy fields are written to whichever real file they represent. By using dummy fields on screens defined in this file, "vidwork", we gain the ability to popup these screens and then perform @when (leaving and entering) processing on these fields. This is not possible when popping up a screen from "another" file. @when processing can only be done on fields in the current file (the file on which you are standing). If we were not able to do such @when processing, we could not popup a receipt screen and then do a lookup for either Account#'s or Catalog#'s while on that screen. There would be no "trigger" processing available.

The technique of doing POPUP UPDATE on a screen full of dummy fields, (assigning them from a lookup done just before the POPUP), and then rewriting any changes in these fields to their counterpart real fields in the looked up record, is cumbersome to say the least. But, since it gives us back the @when trigger capability, the extra work (normally done automatically by a regular "real" field POPUP UPDATE) is well worth it.

Enter the code from the following (umpteen) screens into "vidwork/input". Many of the lines are duplicated from what was there before, but some have changed slightly, so the whole "new" table is shown here. Be sure your table matches this one on all lines and do not just add in the new lines to your old table. (In other words, check line by line for discrepancies between our old code and this new table..)

```

1 -----
  < If:
  Then: delete ; end
2 -----
@wef1 < If: tx eq ""
      Then: gosub gettax
3 -----
  < If:
  Then: cls("22")
4 -----
  < If: l eq ""
  Then: showctr("22") "Enter Acct# or press \r \KZ \r to use Acct Name."
5 -----
  < If:
  Then: e=1;end
6 -----
gettax < If:
      Then: r(1,.0,g)="1"
File: vidwork-----1-----Processing: input
      F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
      PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.

```

```

7 -----
  < If:
  Then: lookup vidctrl r=r -n
8 -----
  < If: not vidctrl
  Then: show "@No control file!!!";return
9 -----
  < If:
  Then: tx(4,.3,g)=vidctrl(2) ; return
10 -----
@wlf1 < If:
      Then:
11 -----
  < If: l eq ""
  Then: gosub clrall;end
12 -----
  < If:
  Then: lookup addr = vidcust k=1 i=b -nx
File: vidwork-----1-----Processing: input
      F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
      PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.

```

```

13 -----
  < If: not addr
  Then: show "@Customer code not on file!" ; gosub clrall ; screen ,1
14 -----
  < If:
  Then: copyin addr
15 -----
  < If:
  Then: display ; end
16 -----
@wef3 < If:
      Then: cls("22")
17 -----
  < If: 3 eq ""
  Then: showctr("22") "Enter a period (.) to add a new customer."
18 -----
  < If: 3 eq ""
  Then: showctr("23") "Enter start of name, or \r \KZ \r to use Acct#."
File: vidwork-----1-----Processing: input
      F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
      PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.

```

```

19 -----
   < If: 3 ne ""
   Then: showctr("22") "Press \r \KE \r for receipts on file."
20 -----
   < If:
   Then: e=3;end
21 -----
@wlf3 < If: fa="dontleavefield"
   Then: fa="";screen ,3
22 -----
   < If:
   Then: showctr("22") ""
23 -----
   < If: 3 co "."
   Then: 3="";gosub clrall;gosub addnew;screen 1,3
24 -----
   < If: 3 eq e and e ne ""
   Then: end
File: vidwork-----1-----Processing: input
      F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.

```

```

25 -----
   < If: 3 eq ""
   Then: gosub clrall ; end
26 -----
   < If:
   Then: gosub prompt
27 -----
brwNAME< If: 'This browse lookup code is on the following two screens*****
   Then: lookup addr = vidcust k=3 *****
28 -----
   < If: 'Enter these two lines from *****
   Then: 'the following two screensl *****
29 -----
   < If: @sk eq "BRKY"
   Then: clearb ; 3=""; display ; screen ,3
30 -----
   < If: @sk eq "SAVE"
   Then: goto brwNAME
File: VIDWORK-----1-----Processing: INPUT
      F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.

```

DEFINE LOOKUP

```

Lookup From File: vidcust <
Name Of Lookup Is: addr <
How Is The Record To Be Found? k< K - Key Field R - Record Number
                          F - Free Record Z - Fuzzy Search
What Index Is To Be Used? a< F6 for list
What Field In 'VIDWORK' 3 <
Contains The Key?
Protect Record in Lookup File? N<
If Key Doesn't Match Exactly: g< X - Key Must Match Exactly
                          G - Use Next Greater Match
                          L - Use Next Lower Match
If Lookup Fails: n< B - Blank The Field
                          N - Do Nothing
                          E - Report An Error
Create Browse Lookup? Y<
Press BREAK To Cancel Changes

```

D E F I N E L O O K U P

```

Browse Header:
Name           Address           Phone           ◀
Browse Format:
*2           <3           *4           *9           ▶
Browse Window: Lines: 18◀   Row:   ▶   Col:   ▶
Pop-Up Screen: Name: 0   ▶   Row:   ▶   Col:   ▶

Only Show Records That Match Key? N◀
Browse Window Show option: P◀ 0 - Show Only. Accept no keystrokes.
(Leave blank for default)   K - Keep Browse Window on Screen.
                           P - Retain Position on re-execution.

Exit Keys:
(Leave blank for none)
Order of Records (Ascending, Descending): A◀
Position of First Record (Top,Middle,Bottom): M◀
Processing Label:   ▶

Are these entries correct? Y◀
Press BREAK To Cancel Changes
    
```

LABEL D E F I N E P R O C E S S I N G 04.05.06D3

```

31 -----
   ◀ If: not addr
   Then: show "@No names" ; clearb ; 3="" ; display ; screen ,3
32 -----
   ◀ If:
   Then: copyin addr ; clearb
33 -----
   ◀ If:
   Then: display ; screen ,3
34 -----
address ◀ If: 'this saves a line of code
   Then: cls("22") ; gosub prompt2
35 -----
   ◀ If:
   Then: lookup new = vidcust r=free -ep
36 -----
   ◀ If:
   Then: gosub promptI
File: vidwork----- 1 -----Processing: input
      F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.
    
```

LABEL D E F I N E P R O C E S S I N G 04.05.06D3

```

37 -----
   ◀ If:
   Then: popup update new,"add" ; clearp
38 -----
   ◀ If: @sk eq "BRKY"
   Then: write new ; delete new ; screen ,3
39 -----
   ◀ If:
   Then: lookup num = vidctrl r=r -np
40 -----
   ◀ If: not num
   Then: show "@No control file!!" ; write new ; delete new ; screen ,3
41 -----
   ◀ If:
   Then: new(1)=num(1);num(1)=num(1)+"1";write num
42 -----
   ◀ If:
   Then: new(16)=new(14)-new(15)
File: vidwork----- 1 -----Processing: input
      F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.
    
```

```

43 -----
    ◀ If:
    Then: 1=new(1);2=new(2);3=new(3);4=new(4)
44 -----
    ◀ If:
    Then: 5=new(5);6=new(6);7=new(7);8=new(8);9=new(9)
45 -----
    ◀ If:
    Then: 11=new(11);12=new(12);14=new(14);15=new(15);16=new(16)
46 -----
    ◀ If:
    Then: write new
47 -----
    ◀ If:
    Then: return
48 -----
@wefra ◀ If:
    Then: dim cat[4]:ra ; dim des[4]:re ; dim chg[4]:ri
File: vidwork-----1-----Processing: input
      F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
      PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.

```

```

49 -----
    ◀ If:
    Then: e=ra ; end
50 -----
@wefrb ◀ If:
    Then: e=rb ; end
51 -----
@wefrc ◀ If:
    Then: e=rc ; end
52 -----
@wefrd ◀ If:
    Then: e=rd ; end
53 -----
@wlfra ◀ If:
    Then: i(1,.0)="1";fd=ra;goto finlook
54 -----
@wlfrb ◀ If:
    Then: i="2";fd=rb;goto finlook
File: vidwork-----1-----Processing: input
      F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
      PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.

```

```

55 -----
@wlfrc ◀ If:
    Then: i="3";fd=rc;goto finlook
56 -----
@wlfrd ◀ If:
    Then: i="4";fd=rd
57 -----
finlook◀ If: cat[i] eq e
    Then: end
58 -----
clrline◀ If: cat[i] eq ""
    Then: des[i]=" " ; chg[i]=" " ; gosub totals ; display ; screen ,( @fd)
59 -----
    ◀ If:
    Then: lookup itm=vidcat k=fd i=a -nx
60 -----
    ◀ If: not itm
    Then: goto getitm
File: vidwork-----1-----Processing: input
      F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
      PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.

```

LABEL

D E F I N E P R O C E S S I N G

```

61 -----
    ◀ If:
    Then: des[i]=itm[2] ; chg[i]=itm[3] ; gosub totals ; display ; end
62 -----
getitm ◀ If:
    Then: cls("22","3") ; gosub prompt3
63 -----
brwITM ◀ If:
    Then: lookup itm = vidcat k=fd i=b -ng b="(brw=18 show=keep fill=asc)
64 -----
    ◀ If:
    Then: Charge)*2 *1
65 -----
    ◀ If: @sk eq "BRKY"
    Then: cat[i]=" " ; clearb ; gosub promptI ; display ; screen ,(@fd)
66 -----
    ◀ If: @sk eq "SAVE"
    Then: goto brwITM
File: vidwork----- 1 -----Processing: input
    F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
    F10 For Help, ESC To Record, BREAK to Cancel.

```

D E F I N E L O O K U P

```

Lookup From File: vidcat ◀
Name Of Lookup Is: itm ◀
How Is The Record To Be Found? k◀ K - Key Field R - Record Number
                                F - Free Record Z - Fuzzy Search
What Index Is To Be Used? b◀ F6 for list
What Field In 'VIDWORK' fd ◀
Contains The Key?
Protect Record in Lookup File? N◀
If Key Doesn't Match Exactly: g◀ X - Key Must Match Exactly
                                G - Use Next Greater Match
                                L - Use Next Lower Match
If Lookup Fails: n◀ B - Blank The Field
                    N - Do Nothing
                    E - Report An Error
Create Browse Lookup? Y◀
Press BREAK To Cancel Changes

```

D E F I N E L O O K U P

```

Browse Header:
Description          Catalog#   Charge   ◀
Browse Format:
*2                  *1       *3       ◀
Browse Window: Lines: 18◀ Row: ◀ Col: ◀
Pop-Up Screen: Name: 0 ◀ Row: ◀ Col: ◀
Only Show Records That Match Key? N◀
Browse Window Show option: P◀ O - Show Only. Accept no keystrokes.
(Leave blank for default) K - Keep Browse Window on Screen.
                          P - Retain Position on re-execution.
Exit Keys: ◀
(Leave blank for none)
Order of Records (Ascending, Descending): A◀
Position of First Record (Top,Middle,Bottom): M◀
Processing Label: ◀
Are these entries correct? Y◀
Press BREAK To Cancel Changes

```

LABEL D E F I N E P R O C E S S I N G

```

67 -----
   < If: not itm
   Then: show "@No videos!" ; clearb;gosub promptI;cat[i]="";goto clrline <
68 -----
   < If:
   Then: cat[i]=itm(1) ; des[i]=itm(2) ; chg[i]=itm(3) <
69 -----
   < If:
   Then: clearb ; gosub promptI ; gosub totals ; display ; end <
70 -----
@wlfm < If:
   Then:
71 -----
@wlfmq < If:
   Then: gosub totals ; display ; end <
72 -----
prompt < If:
   Then: cls("22","3") <
File: vidwork-----1-----Processing: input
      F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
      PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.

```

LABEL D E F I N E P R O C E S S I N G

```

73 -----
   < If:
   Then: showctr("23")"ARROWS, NxtPg/PrvPg to scroll. \r \KZ \r to select." <
74 -----
prompt2 < If:
   Then: showctr("22") "Press \r \K4 \r to Record, \r \KY \r to Cancel" <
75 -----
   < If:
   Then: return <
76 -----
promptR < If:
   Then: cls("22","3") <
77 -----
   < If:
   Then: showctr("23") "\r A \r-Add, \r D \r-Delete, \r X \r-Exit" <
78 -----
prompt3 < If:
   Then: showctr("24")"ARROWS/NxtPg/PrvPg-Scroll, or \r \KZ \r to Select." <
File: vidwork-----1-----Processing: input
      F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
      PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.

```

LABEL D E F I N E P R O C E S S I N G 04.05.06D3

```

79 -----
   < If:
   Then: return <
80 -----
promptD < If:
   Then: cls("22") <
81 -----
   < If:
   Then: showctr ("23") "\r A \r-Add, \r X \r-Exit" <
82 -----
   < If:
   Then: return <
83 -----
promptI < If:
   Then: cls("22") <
84 -----
   < If:
   Then: s="Fill in Screen. \r \K4 \r to Record, \r \KY \r to cancel." <
File: vidwork-----1-----Processing: input
      F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
      PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.

```



```

85 -----
    ◀ If:
      Then: showctr("22") s
86 -----
    ◀ If:
      Then: return
87 -----
clrall ◀ If:
      Then: dim clrtr[16]:1 ; clear clrtr ; display ; return
88 -----
totals ◀ If:
      Then: 'this subroutine does all the money calculations
89 -----
    ◀ If:
      Then: rn = ri + rj + rk + rl + rm
90 -----
    ◀ If:
      Then: ro = rn * tx
File: vidwork-----1-----Processing: input
      F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
      PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.

```

```

91 -----
    ◀ If:
      Then: rp = rn + ro
92 -----
    ◀ If:
      Then: rr = rp - rq
93 -----
    ◀ If: rs eq ""
      Then: rs=@td ; rt=@td+3"
94 -----
    ◀ If:
      Then: ru="0"
95 -----
    ◀ If: rr eq "0"
      Then: ru="C"
96 -----
    ◀ If:
      Then: return
File: vidwork-----1-----Processing: input
      F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
      PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.

```

```

97 -----
@wuk3 ◀ If:
      Then: fa(1)="dontleavefield"
98 -----
    ◀ If:
      Then: cls("22")
99 -----
    ◀ If:
      Then: fl(1)=" "
100 -----
    ◀ If:
      Then: input popup q(1,yesno) "Show only Open receipts? (y/n) "
101 -----
    ◀ If: @sk eq "BRKY"
      Then: screen ,3
102 -----
    ◀ If: q eq "N" 'flag will not be set if q="Y" or q="" (RETURN)
      Then: fl="1"
File: vidwork-----1-----Processing: input
      F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
      PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.

```

```

103 -----
startb ◀ If:
      Then:
104 -----
      ◀ If:
      Then: lookup tst = vidrec k=1 i=a -nx
105 -----
      ◀ If: not tst
      Then: fn="none";goto moreb
106 -----
      ◀ If:
      Then: fn=""
107 -----
moreb ◀ If: fn eq "none"
      Then: gosub promptD;goto morebb
108 -----
      ◀ If:
      Then: gosub promptR
File: vidwork-----1-----Processing: input
      F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.
  
```

```

109 -----
morebb ◀ If:
      Then: ba="(brw=18 show=keep prc=prcRCT xkeys=ADX mlen=6 fill=asc,top)"
110 -----
      ◀ If:
      Then: bb="[RctDate DueDate Charges Payment BalDue Catalog#'s"
111 -----
      ◀ If:
      Then: bb=bb&"-on-the-receipt]"
112 -----
      ◀ If:
      Then: bc="*6 *7 *13 *14 *15 *17 *18 "
113 -----
      ◀ If:
      Then: bc=bc&"*19 *20"
114 -----
brwRCT ◀ If:
      Then: lookup rct=vidrec k=1 i=a -nxmp -s b=(ba&bb&bc)
File: vidwork-----1-----Processing: input
      F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.
  
```

```

115 -----
      ◀ If: @sk eq "BRKY"
      Then: clearb ; screen 1,3
116 -----
      ◀ If: @sk eq "SAVE"
      Then: goto brwRCT
117 -----
      ◀ If: @bk eq "A"
      Then: clearb ; fk(3)="add" ; gosub add ; goto startb
118 -----
      ◀ If: not rct
      Then: clearb;screen 1,3
119 -----
      ◀ If: @bk eq "D"
      Then: gosub delit ; clearb ; goto startb
120 -----
      ◀ If: @bk eq "X"
      Then: clearb ; screen 1,3
File: vidwork-----1-----Processing: input
      F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.
  
```

LABEL D E F I N E P R O C E S S I N G

```

121 -----
    ◀ If:
    Then: clearb ; gosub modify ; gosub promptR ; goto brwRCT
122 -----
prcRCT ◀ If: fl eq "1"
    Then: end
123 -----
    ◀ If: rct(15) eq "0"
    Then: DROP
124 -----
    ◀ If:
    Then: end
125 -----
add ◀ If:
    Then: lookup rct=vidrec r=free -ep
126 -----
modify ◀ If:
    Then: gosub getvars
File: vidwork-----1-----Processing: input
    F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
    PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
    F10 For Help, ESC To Record, BREAK to Cancel.

```

LABEL D E F I N E P R O C E S S I N G

```

127 -----
    ◀ If:
    Then: gosub promptI
128 -----
    ◀ If:
    Then: popup update -, "rec"
129 -----
    ◀ If: @sk eq "BRKY" and fk="add"
    Then: fk="" ; write rct ; delete rct ; return
130 -----
    ◀ If: @sk eq "BRKY"
    Then: return
131 -----
    ◀ If:
    Then: gosub printit ; gosub post ; clearp ; RETURN
132 -----
delit ◀ If:
    Then: gosub getvars
File: vidwork-----1-----Processing: input
    F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
    PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
    F10 For Help, ESC To Record, BREAK to Cancel.

```

LABEL D E F I N E P R O C E S S I N G 04.05.06D3

```

133 -----
    ◀ If:
    Then: cls("22")
134 -----
    ◀ If:
    Then: beep ; popup -, "rec"
135 -----
    ◀ If:
    Then: input q "Are you \r SURE \r you want to delete this? (y/n) "
136 -----
    ◀ If: q ne "Y"
    Then: clearp ; return
137 -----
    ◀ If:
    Then: gosub postd
138 -----
    ◀ If:
    Then: clearp
File: vidwork-----1-----Processing: input
    F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
    PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
    F10 For Help, ESC To Record, BREAK to Cancel.

```

```

139 -----
    ◀ If:
      Then: return
140 -----
getvars◀ If:
      Then: oc(6,.2)=rct(13);op(6,.2)=rct(14)
141 -----
    ◀ If:
      Then: ra(8,allup)=rct(17);rb(8,allup)=rct(18);rc(8,allup)=rct(19)
142 -----
    ◀ If:
      Then: rd(8,allup)=rct(20);re(40)=rct(21);rf(40)=rct(22);rg(40)=rct(23)
143 -----
    ◀ If:
      Then: rh(40,allup)=rct(24);ri(6,.2)=rct(25);rj(6,.2)=rct(26)
144 -----
    ◀ If:
      Then: rk(6,.2)=rct(27);rl(6,.2)=rct(28);rm(6,.2)=rct(10)
File: vidwork-----1-----Processing: input
      F8 -Block Func, F9 -Go To/Find F5 -To Define & Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.
  
```

```

145 -----
    ◀ If:
      Then: rn(6,.2)=rct(11);ro(6,.2)=rct(12);rp(6,.2)=rct(13)
146 -----
    ◀ If:
      Then: rq(6,.2)=rct(14);rr(6,.2)=rct(15);rs(8,mdy/)=rct(6)
147 -----
    ◀ If:
      Then: rt(8,mdy/)=rct(7);ru(1,allup)=rct(8)
148 -----
    ◀ If:
      Then: rv(25)=6{" "<7<8
149 -----
    ◀ If:
      Then: return
150 -----
printit◀ If:
      Then: cls("22")
File: vidwork-----1-----Processing: input
      F8 -Block Func, F9 -Go To/Find F5 -To Define & Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.
  
```

```

151 -----
    ◀ If:
      Then: input popup q "Print this receipt now? (y/n) \r \KZ \r=YES "
152 -----
    ◀ If: q eq "N"
      Then: return
153 -----
    ◀ If:
      Then: form "receipt" ; RETURN
154 -----
post ◀ If:
      Then: 'write out the changes to the receipt file
155 -----
    ◀ If:
      Then: rct(1)=1;rct(2)=2;rct(3)=3;rct(4)=11
156 -----
    ◀ If:
      Then: rct(17)=ra;rct(18)=rb;rct(19)=rc;rct(20)=rd
File: vidwork-----1-----Processing: input
      F8 -Block Func, F9 -Go To/Find F5 -To Define & Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.
  
```

LABEL D E F I N E P R O C E S S I N G

```

157 -----
    ◀ If:
    Then: rct(21)=re;rct(22)=rf;rct(23)=rg;rct(24)=rh
158 -----
    ◀ If:
    Then: rct(25)=ri;rct(26)=rj;rct(27)=rk;rct(28)=rl;rct(10)=rm
159 -----
    ◀ If:
    Then: rct(11)=rn;rct(12)=ro;rct(13)=rp;rct(14)=rq;rct(15)=rr
160 -----
    ◀ If:
    Then: rct(6)=rs;rct(7)=rt;rct(8)=ru
161 -----
    ◀ If: 'ensure the write happens now
    Then: write rct
162 -----
    ◀ If: 'write out the changes to the customer file
    Then: lookup hst = vidcust k=1 i=B -nxp
File: vidwork-----1-----Processing: input
    F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
    F10 For Help, ESC To Record, BREAK to Cancel.

```

LABEL D E F I N E P R O C E S S I N G

```

163 -----
    ◀ If: not hst
    Then: show "@Impossible. Customer not in customer file!";return
164 -----
    ◀ If:
    Then: hst(14) = hst(14) + rct(13) - oc
165 -----
    ◀ If:
    Then: hst(15) = hst(15) + rct(14) - op
166 -----
    ◀ If:
    Then: hst(16) = hst(14) - hst(15)
167 -----
    ◀ If: 'ensure the write
    Then: write hst
168 -----
    ◀ If: 'posting this info to the temporary stand-on record needs no write
    Then: 14=hst(14);15=hst(15);16=hst(16)
File: vidwork-----1-----Processing: input
    F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
    F10 For Help, ESC To Record, BREAK to Cancel.

```

LABEL D E F I N E P R O C E S S I N G

```

169 -----
    ◀ If:
    Then: return
170 -----
postD ◀ If: 'write out deleteion to customer file
    Then: lookup hst = vidcust k=1 i=B -nxp
171 -----
    ◀ If: not hst
    Then: show "@Impossible. Customer not in customer file!";return
172 -----
    ◀ If:
    Then: hst(14) = hst(14) - rct(13)
173 -----
    ◀ If:
    Then: hst(15) = hst(15) - rct(14)
174 -----
    ◀ If:
    Then: hst(16) = hst(14) - hst(15)
File: vidwork-----1-----Processing: input
    F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
    F10 For Help, ESC To Record, BREAK to Cancel.

```

```

LABEL          DEFINE PROCESSING
-----
175 -----
    ◀ If: 'ensure the write
      Then: write hst
176 -----
    ◀ If: 'posting this info to the temporary stand-on record needs no write
      Then: l4=hst(14);l5=hst(15);l6=hst(16)
177 -----
    ◀ If: 'overlay a one element array on top of the receipt record, making
      Then: dim thisrct[1](300):rct(1) 'sure it has the right # of characters
178 -----
    ◀ If:
      Then: lookup vidrecd r=free -sp
179 -----
    ◀ If: 'overlay a one element array on a free record in a holding file
      Then: dim safe[1](300):vidrecd(1) 'that has the same size/shape as "rct"
180 -----
    ◀ If: 'assign the free record the entire contents of the record being
      Then: safe["1"]=thisrct["1"] 'deleted in one fell swoop
File: vidwork----- 1 ----- Processing: input
      F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.

```

```

CONDITION     DEFINE PROCESSING
-----
181 -----
    ◀ If: 'write out the holding file record
      Then: write vidrecd
182 -----
    ◀ If: 'Now delete the receipt record itself. The copy for safekeeping
      Then: delete rct 'remains in the holding file (vidrecd) until removed
183 -----
    ◀ If:
      Then: return
184 -----
    ◀ If:
      Then:
185 -----
    ◀ If:
      Then:
186 -----
    ◀ If:
      Then:
File: vidwork----- 1 ----- Processing: input
      F8 -Block Func, F9 -Go To/Find F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
      F10 For Help, ESC To Record, BREAK to Cancel.

```

There are several enhancements to the Video Store app in this table.

A new trigger has been used, **@wuk** processing (at when user key). This means, do some processing when the user is sitting in a particular field and presses one of 4 special "user-keys". These user-keys are F8, F9, F3 and F4. We are implementing **@wuk3** and showing the prompt F8 as the key which can be pressed while in this field. Pressing this key brings up all the receipts on file for the Account# we have on our screen. This browse lookup has several "exit keys" attached to it which means the user can press any of these keys while the browse window is up, and the designated action will occur. They are called "exit keys" because, by their nature they suggest exiting the browse window to accomplish some task. We have defined exit keys to Delete the receipt record under the highlighted bar, to Add a new receipt directly from the browse screen, and defined that X will clear the browse window and return the user to the first screen.

The new browse into the receipts file (label brwRCT) is also built a little differently than normal browse lookups. It is built by filling some variables with the different parts of the browse lookup and then using these variables on the browse line itself. This is a very valuable trick to use when the browse data lines you want to show are too long to be built using the browse lookup wizard.

The brwRCT browse also makes use of "browse processing". This is a special kind of processing that happens just before each looked up record is put in the browse window. In our case, we are using this processing stub to test whether the record about to be put on the screen has a Balance_Due equal to "0", if it does, the DROP command causes it to be left out of the display. We only see records on which money is owed.

The "delit" subroutine calls another subroutine "postd" which uses arrays to delete a receipt record and also save a copy of the receipt in a "safe" file. This prevents users from deleting records and leaving no trace. By overlaying an array on the receipt and overlaying an array on a free lookup into the "safe" file (an exact duplicate of the receipt file), the processing can copy the record from one file to the other with one command, rather than a series of many field assignments.

New Features In An Old Application

Try it all out now! But before you do, go to the filePro Directory (? on the main menu) and delete the records in the receipts file "vidrec". This may leave some unsupported charges, payments and balances in the customer file, but that's okay for our purposes.

Once you have cleared the "vidrec" file. Use the menu "video" to enter the virtual "work" file.

Load the following customer Account#. Use either method (Account# or Name).

You should see the following.

```
Video Store
-----
Account_code: 106
First_Name: Robin
Last_Name: Hakan
Address: 7 Lucky Way
Address: Penthouse
City: Oakland
State: NJ
Zip: 07436
Phone: (201) 777-7777

Customer_type: R
First_sale_date: 04/28/97

Charges: 40.10
Payments: .00
Balance_due: 40.10
```

Press **F8** for receipts on file.

Press F8 to see the receipts on file for this customer.

Since there are no receipts on file (we just cleared them all), you should see the following.

```
Video Store
-----
RctDate DueDate Charges Payment BalDue Catalog#'s-on-the-receipt
```

A -Add, **X** -Exit

Press A to add a new receipt.

You will see the following.

```
Account Code: 106 Type: R Address: 7 Lucky Way
First Name: Robin : Penthouse
Last Name: Hakan : Oakland, NJ 07436
Phone: (201) 777-7777

Catalog# Description Charge
-----
1002
Late Charge:
Subtotal:
Tax:
Receipt Date: Total charge:
Video Due Date: Payment:
Receipt Status: Balance Due:
```

Fill in Screen. **ESC** to Record, **BREAK** to cancel.

Enter the Catalog#s shown. You should see the descriptions and charges popping in appropriately.

```

Account Code: 106 Type: R Address: 7 Lucky Way
First Name: Robin : Penthouse
Last Name: Hakan : Oakland, NJ 07436
Phone: (201) 777-7777

Catalog# Description Charge
-----
1002 ◀ Around The World In 80 Days 4.00
A123 ◀ Dirty Dancing 4.00
█ ◀
◀

Late Charge: ◀
Subtotal: 8.00
Tax: .44
Total charge: 8.44
Payment: ◀
Balance Due: 8.44

Receipt Date: 09/01/97
Video Due Date: 09/04/97
Receipt Status: 0
  
```

Fill in Screen. **ESC** to Record, **BREAK** to cancel.

Now try finding a Catalog item by the first few letters of its description. Enter an **f** on line 3 .

```

Account Code: 106 Type: R Address: 7 Lucky Way
First Name: Robin : Penthouse
Last Name: Hakan : Oakland, NJ 07436
Phone: (201) 777-7777

Catalog# Description Charge
-----
1002 ◀ Around The World In 80 Days 4.00
A123 ◀ Dirty Dancing 4.00
f ◀
◀

Late Charge: ◀
Subtotal: 8.00
Tax: .44
Total charge: 8.44
Payment: ◀
Balance Due: 8.44

Receipt Date: 09/01/97
Video Due Date: 09/04/97
Receipt Status: 0
  
```


Fill in Screen. **ESC** to Record, **BREAK** to cancel.

You should see a browse of the Catalog file. The highlighted bar will be on the first record starting with f.

```

Account Beginning of File
First Description Catalog# Charge
Last
Around The World In 80 Days 1002 4.00
Crossroads 1001 15.00
Dirty Dancing A123 4.00
Fiddler On The Roof 444JM 6.00
French Kiss 7744 5.00
A12 Ghost 6222G 3.00
F Mission: Impossible 1283M 4.00
Pure Country 888COUN 4.00
Serial ABC1 6.00
Total Recall 3999 4.00

Rec
Video
Recei End of File
  
```

ARROWS/NxtPg/PrvPg-Scroll, or  to Select.

Move the highlighted bar down to the next item, which is what you were looking for and press **ENTER**. You should see the following pop in.


```

Account Code: 106 Type: R Address: 7 Lucky Way
First Name: Robin : Penthouse
Last Name: Hakan : Oakland, NJ 07436
Phone: (201) 777-7777

Catalog# Description Charge
1002 ◀ Around The World In 80 Days 4.00
A123 ◀ Dirty Dancing 4.00
7744 ◀ French Kiss 5.00
◀

Late Charge: ◀
Subtotal: 13.00
Tax: .72
Total charge: 13.72
Receipt Date: 09/01/97 Payment: ◀
Video Due Date: 09/04/97 Balance Due: 13.72
Receipt Status: 0

```

Fill in Screen. **ESC** to Record, **BREAK** to cancel.

Press **ESC** to save your work.

You will be prompted as to whether you want to print the receipt or not. The default is Yes. (Just pressing ENTER will print the receipt.)

```

Account Code: 106 Type: R Address: 7 Lucky Way
First Name: Robin : Penthouse
Last Name: Hakan : Oakland, NJ 07436
Phone: (201) 777-7777

Catalog# Description Charge
1002 ◀ 4.00
A123 ◀ Print this receipt now? (y/n) ◀=YES ◀ 4.00
7744 ◀ 5.00
◀

Late Charge: ◀
Subtotal: 13.00
Tax: .72
Total charge: 13.72
Receipt Date: 09/01/97 Payment: ◀
Video Due Date: 09/04/97 Balance Due: 13.72
Receipt Status: 0

```

You will be returned to the receipts browse window for this customer. You should see the data for this receipt across the line.

You will be returned to the receipts browse window for this

```

Video Store
RctDate DueDate Charges Payment BalDue Catalog#'s-on-the-receipt
09/01/97 09/04/97 13.72 13.72 1002 A123 7744

End of File

```

A -Add, **D** -Delete, **X** -Exit
 ARROWS/NxtPg/PrvPg-Scroll, or **◀** to Select.

Press **X** to exit this screen.

You are returned to the main customer record. The newly added receipt was added to the Charges, Payments and Balance_Due fields as it should have been.

```

Account_code: 106◀
First_Name: Robin
Last_Name: Hakan ◀
Address: 7 Lucky Way
Address: Penthouse
City: Oakland
State: NJ
Zip: 07436
Phone: (201) 777-7777

Customer_type: R
First_sale_date: 04/28/97

Charges: 53.82
Payments: .00
Balance_due: 53.82
    
```

Press **F8** for receipts on file.

customer. Let's try modifying a receipt.

Press **F8** again to go back to the receipts browse for this

Choose the receipt from the browse screen by pressing **ENTER**.

The receipt will appear in full screen view. **Modify** it, by removing the 2nd line item. (space over it).

```

Account Code: 106 Type: R Address: 7 Lucky Way
First Name: Robin : Penthouse
Last Name: Hakan : Oakland, NJ 07436
Phone: (201) 777-7777

Catalog# Description Charge
-----
1002 ◀ Around The World In 80 Days 4.00
7744 ◀ French Kiss 5.00
Late Charge: ◀
Subtotal: 9.00
Tax: .50
Total charge: 9.50
Payment: ◀
Receipt Date: 09/01/97 Balance Due: 9.50
Video Due Date: 09/04/97
Receipt Status: 0
    
```

Fill in Screen. **ESC** to Record, **BREAK** to cancel.

When done, press **ESC** to save your work.

You will be brought back to the receipts browse. The receipt will appear updated.

Press **X** to return to the customer screen.

You will see that the customer record has been updated as to the modification you just made. The system is working perfectly.

Video Store

Account_code: 106◀
First_Name: Robin
Last_Name: Hakan ◀
Address: 7 Lucky Way
Address: Penthouse
City: Oakland
State: NJ
Zip: 07436
Phone: (201) 777-7777

Customer_type: R
First_sale_date: 04/28/97

Charges: 49.60
Payments: .00
Balance_due: 49.60

Press **ES** for receipts on file.

Press **ESC** to clear this customer and obtain a blank starting screen. (This is where the program simply deleted the virtual record you are standing on, and gave it right back to you again.)

Your screen should look like this.

Video Store

Account_code: █ ◀
First_Name:
Last_Name: ◀
Address:
Address:
City:
State:
Zip:
Phone:

Customer_type:
First_sale_date:

Charges:
Payments:
Balance_due: █

Enter Acct# or press **←** to use Acct Name.

The last feature we need to try is the "delete a whole receipt" routine. To test this we need a few receipts on file for a customer. Go add 3 quick receipts for Account# 106. Choose any Catalog item(s) you wish. Don't waste time on this. We are just going to delete them!

Once you have added the receipts, choose one of them from the receipts browse window by putting your highlighted bar on the want deleted and **press D**.

The receipt will be brought to the screen in full screen view and you will be prompted for acknowledgment.

Your screen should look something like this.

Account Code: 105 Type: R Address: 301 Ipswitch Road
 First Name: Maggie :
 Last Name: Mae : Pompton Lakes, NJ 07771
 Phone: (201) 429-3838

Catalog#	Description	Charge
6222G	◀ Ghost	3.00
ABCl	◀ Serial	6.00
	◀	
	◀	
	Late Charge: ▶	
	Subtotal:	9.00
	Tax:	.50
Receipt Date: 50.6097	Total charge:	9.50
Video Due Date: 09/04/97	Payment: ▶	
Receipt Status: 30.60	Balance Due:	9.50

Are you **SUPP** you want to delete this? (y/n) ◀

Enter a Y to delete the receipt.

When you are returned to the receipts browse window, the one you deleted will no longer be there.

Your screen should look something like this.

Video Store

RctDate	DueDate	Charges	Payment	BalDue	Catalog#'s-on-the-receipt
09/01/97	09/04/97	12.66	12.66	1283M	888COUN 3999
09/01/97	09/04/97	8.44	8.44	1002	A123

—End of File—

A -Add, **D** -Delete, **X** -Exit
 ARROWS/NxtPg/PrvPg-Scroll, or **←** to Select.

You will remember from the code you entered that we are not actually deleting this receipt. We are copying it first to a "safe" file called "vidrecd" and then deleting it from the real receipts file. The user will want to view these deleted records from time to time, so to finish this application, this file should be put on the user menu "video".

LONG DESCRIPTION

D E F I N E U S E R M E N U S

```

7 -----
7◀ Work File                               ▶
  \fp\dclerk vidwork -sl -xa -d           ▶
8 -----
8◀ Deleted Records File                   ▶
  \fp\dclerk vidrecd -sl                 ▶
9 -----
◀                                         ▶
10 -----
◀                                         ▶
Menu: VIDEO
F5 - Create batch file, ESC - Save, BREAK - Cancel, F10 - Help

```

So that there will be a screen in "vidrecd" that resembles the receipts screen in "vidwork" or "vidrec", copy "vidrec/screen.1" to "vidrecd/screen.1". Use the **(C)opy** function in Design Screens to do this.

Deleted Records File

Account Code: *1 Type: !4 Address: *aa
First Name: !2 : *ab
Last Name: *3 : *ac
 Phone: *ad

Catalog#	Description	Charge
*17	!21	!25
*18	!22	!26
*19	!23	!27
*20	!24	!28
	Late Charge:	*10
	Subtotal:	!11
	Tax:	!12
	Total charge:	!13
Receipt Date: !6	Payment:	*14
Video Due Date: !7	Balance Due:	!15
Receipt Status: !8		

File: VIDRECD-----Color Screen: 1

Enter Selection >

F10 -Help, **U** -Update, **S** -Select Screen, **F** -Select File,
C -Copy Screen, **H** -Hardcopy, **D** -Delete Screens, **X** -Exit

This application is obviously just a shell. There are many places where it can be tricked, broken, fooled, etc. However, it is still a good starting point for designing a real application. Remember, though, it was only designed to demonstrate various filePro functions and processing.

Arrays

An array is a name that is given to a set of elements. In filePro, we can match an array to a list of real fields, dummy fields or lookup fields. We say that an array can be "overlaid" or "aliased" or "mapped to" the group of fields.

Arrays are used for repetitive operations to increase speed and to decrease the size of your processing table.

Arrays are created with the DIM command. They are given a specified number of elements (fields) and attributes (lengths and edit types) if required.

Arrays are mapped to a specific set of fields. Those fields must be contiguous.

The element number of an array is called a subscript, and fields within the array are referred to as array sub1, written array["1"], or array sub2, written array["2"].

The subscripts are a pointer to that fields place in the array. They are numeric and can be manipulated with math formulas. By adding or subtracting to the subscript you can move around inside the array. By using an integer type dummy field as the subscript for arrays you can manipulate the subscript very easily with simple math.

Remember not to use array["1"] and array[1] interchangeably. The first is correct, the second means array sub(contents of field 1), which may or may not make sense. If the contents of field 1 is within the limits of the number of elements in the array it will work, otherwise it wont.

Dimensioning Arrays (defining them)

Version 6.0.00 dimarray(); ' 0 size arrays are now allowed by filePro

Example 1

```
dim array[20] no alias
```

The elements in the array will take on attributes of whatever is put into them according to normal filePro rules.

Example 2

```
dim array[20]:13 alias begins at field 13
array[" 1 "] is equal to field 13
array[" 2 "] is equal to field 14
array[" 20 "] is equal to field 32
```

Example 3

```
dim array[30]:aa alias begins at dummy field AA
array["1"] is equal to field AA
array["2"] is equal to field AB
array["26"] is equal to field AZ
array["27"] is equal to field BA
array["30"] is equal to field BD
```

Example 4

```
dim array[20]:filename(13) alias begins at lookup field 13
array["1"] is equal to looked-up filename(13)
array["2"] is equal to looked-up filename(14)
array["20"] is equal to looked-up file(32)
```

Example 5

```
dim array[20](8,2):13
array["1"] is equal to field 13
```

You can define the length and edit type for all members of the array

Example 6

```
dim array[5](8,2)(10,*)(2,state)(1,yesno)(5,allup):13
array[" 1 "] is equal to field 13
```

You can define the length and type for each specific member

Example 7 - Performing Repetitive Calculations

If you have 12 sets of data to perform an identical operation upon, instead of handling each set individually, you can overlay an array and perform the calculation repeatedly until the program reaches the end of the array.

```
Then: n[2,0]="1"
Then: dim price[12]:2; dim quant[12]:14; dim total[12]:26
Loop If: quant[n] ne ""
Then: total[n]=price[n]*quant[n]
If: n lt "12"
Then: n=n+ " 1 " ; goto loop
Then: end
```

Example 8 - Moving Data

Arrays can be used to move large amounts of data from one record to another. This is done by overlaying an array on the lookup file and overlaying an array on the record you are standing on, then running a loop which copies one array to the other field by field.

```

Then: dim here[20]:12
Then: lookup cust=mscust k=1 i=a -ex
Then: dim there[20];cust(12)
Then: l(2,0)="1"
Again
Then: here[i]=there[i]
f: i eq " 20 "
Then: end
Then: i=i+"1";goto again

```

Example 9 - Arrays and Columns

By overlaying an ARRAY over columns of fields, you can operate on related line items. Imagine an invoice with 10 quantity fields, 10 catalog items fields, 10 price fields and 10 extension fields. Lets say you wanted to find out the total amount of money collected for one particular catalog item even though it may have been sold on any one or all of the line items. You could overlay a 10 element array over the catalog item fields, and a 10 element array over the 10 extensions fields. Then you could test the catalog array to see if the catalog item you are looking for is on the invoice, if it is found, take the corresponding extension (from its array) and add it to your count.

Example:

```

If: s = ""
Then: input s(5,0,g)"What stock number are you searching for?"
If:
Then: i(2,0)="1" ; dim cat[10]:5 ; dim ext[10]:45
Again If: i gt " 10"
Then: end
If: cat[i]=s
Then: ct(10,.2,g) = ct + ext[i];i=i+"1";goto again
If:
Then: end

```

SCREEN

```

Catalog #  Extended Price
*5  *45
*6  *46
*14 *54

```

Arrays and Associated Fields

If you overlay an ARRAY over a group of associated fields you can do better magic. This is because of a system maintained field called @AF. It holds a number corresponding to the occurrence of the associated field within its group.

If you have 32 associated fields and you do a search of that group and find a match, the @AF field will contain the occurrence number. If it was the fifth one in the associated group, then @AF will be "5".

The applications are endless. Take the above example of finding total number of a particular catalog item sold. If your columnar fields are associated fields, then the arrays overlaid on top of them can be scanned with one command. Rather than testing each element of the catalog array you can just say if the associated field group holds the catalog number, then the extension you want is ext[@AF].

Example

```

Assume field 5 starts associated field A2
if: s=""
then: input s(6,0,g) "Enter Stock Number: "
if:
then: dim cat[10]:5;dim ext[10]:45
if: A2 co s
then: ct(10,.2,g) = ct + ext[@AF]

```

Catalog # Extended Price

```

*5 A2) *45
*6 A2) *46
*14 A2) *54

```

This code is not very robust, what if you have more than one occurrence? However, it demonstrates the idea. @when leaving field is a good place to make use of arrays and @af combinations. Take a look at the following for concise easy code? It handles all ten lines of an invoice.

```

@wlf0 if: unit[@af] ne ""
then: chg[@af]=qty[@af] * unit[@af]
if:
then: gosub scr1tot;display;end

```

Note: (Parenthesis are usually optional when referring to associated fields. e0 = e0)

IMPORTANT: Use brackets [] on arrays instead of parenthesis (), always! It is easier to tell them apart from lookups.

Totaling Months with Arrays

Often people have devised accounting systems in filePro that do the job at hand, but dont lend themselves to future expansion and needs. One such situation is where a record

holds a series of payment fields and another series of fields to hold the dates of the individual payments. People usually pick a number of payments that they don't think will be exceeded. Let's talk about a system that has had 10 payment fields (and 10 dates) allocated per record. A typical screen might look like the following:

Account Code: *1

Company Name: !2

PAYMENT HISTORY

Date Amount

Payment 1: *78 *35

Payment 2: *79 *36

Payment 3: *80 *37

Payment 4: *81 *38

Payment 5: *82 *39

Payment 6: *83 *40

Payment 7: *84 *41

Payment 8: *85 *42

Payment 9: *86 *43

Payment 10: *87 *44

=====
Total Receipts: 160

Payments and their receipt date are put into the next available pair of fields. Everything is fine and you can easily get a total of payments made by adding all the payment fields together. But what if YOU want to know how much money you received during the month of May, or how much was received on a particular day? It is possible that the specified range of payments fall into any of the 10 allotted fields...

You will need to build a processing table for a report that tests every single date field to see if it contains a date within the desired range, and if it does add its partner amount into the total. To do this conventionally with field numbers would require many lines of programming! This kind of hard coded programming is also inelegant. (As if you care! Its the massive amount of typing we all hate.)

The report can be built quickly (and elegantly) with arrays. We can overlay arrays over the fields and refer to the array name and an element number instead of the actual field numbers. It requires that the 10 (or however many payment fields you need) are contiguous in the Define Files Map, and the 10 date fields for these payments must also be contiguous.

You build an output report that says something in the Total Lines Area like:

THE TOTAL RECEIPTS FOR

*bd to *ed were *aa

That's all. Then you build processing attached to this report which does the following. First it asks the operator for the date range required, (for May the beginning and ending dates of May are entered, for any particular day both the beginning date and ending date can be the same. The processing dimensions an array called date with 10 elements and overlays it on top of the 10 date fields. It does the same with an array called amount and overlays it on top of the 10 payment fields. In the example these fields start at 78 and 35 respectively. The syntax shown in line 6 makes the first element of these arrays start on the specified field and every subsequent element of the array is equal to every subsequent field, net date["2"] is equal to field 70 date["3"] is equal to field 80 amount["1"] is equal to field 35, cat["5"] is equal to field 39 and so forth.

The processing starts at the first element in the date array (because "i", the array index, starts out being set equal to "1") and if the first date falls within the requested date range, the associated payment, (also using an index of "i" or "1"> is added into the total, field "aa". Then "i" is incremented tested to see if it exceeds 10 (the maximum number of elements in our arrays) and if it isn't, everything happens again, if it is greater than 10, the process is done on this record and the report moves on to the next record and its data. Because the totaling field "aa" is global it will hold its value from record to record. At the end the report will print the total of all payments received within the specified date range.

To use this table on your own records that are structured like the above example, be sure to set the array sizes to the number of fields you have set aside for payments and dates. Overlay the arrays on top of the starting field in each group (line 67. Then be sure to set the boundary check on line 8 to the number of elements you have defined for the arrays.

You will find that if you understand the nature of how arrays are working in this example, you will be able to use their power and efficiency in many other applications. (HINT: Make me 10 date fields an associated Held group and the amount field another associated field group. Thus will here in many instances. As an example, if the dates are associated fields, you could scan for all payments in a particular date range. This would be a very complicated group of selection sets to write otherwise and selecting on a particular date would be extremely difficult to do.).

```
If: ad ne ""
```

```
Then: goto doit
```

```
sdate If:
```

```
Then: input sd(8,mdy/,g) "Enter beginning date?"
```

```
If: sd=""
```

```
Then: goto sdate
```

```
edate If:
```

```
Then: input ed(8,mdy/,g) "Enter ending date?"
```

```
If: ed = ""
```

```
Then: goto edate
```

```
doit If:
```

```
Then: dim date[10]:78 ; dim amount[10]:35
```

```
If:
```

```
Then: I(2,0)="1"
```

```
Loop If: I gt "10"
```

```
Then: end
```

```
If: date[i] ge sd and date[i] le ed
```

```
Then: aa(7,2,g)=aa+amount[i]
```

```
if:
```

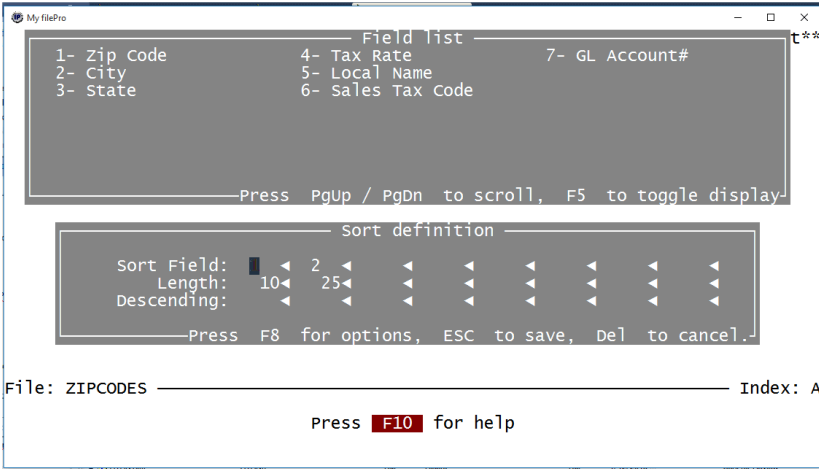
```
Then: i=i+1:goto loop
```


AutoIndexSelection - Version 5.8.01

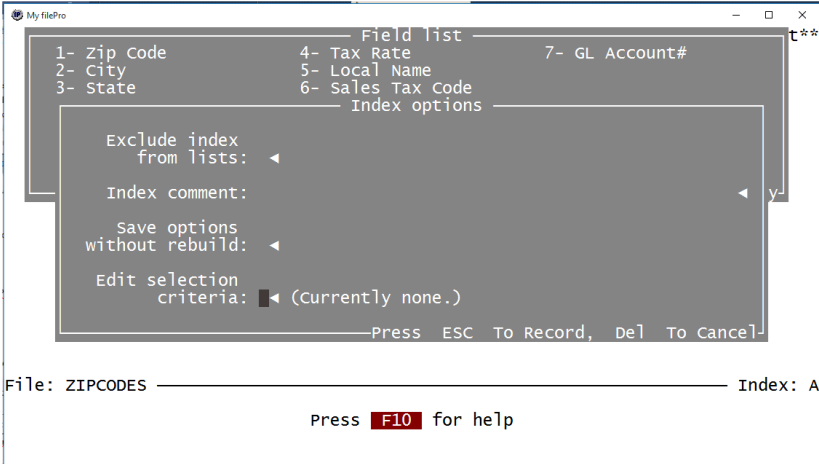
See [LXSE](#) and [-SX](#) flag - (Version 5.8.03)

Automatic indexes now have the advanced feature of being able to attach a selection set to the index. This enables you to have only certain records that meet the selection criteria to be included in the index.

In index maintenance, when you press F8 for Options...



... you will now see an option for adding a selection set.



Enter Y to set the selection set for this index. The selection screen looks the same as the Extended Selection of IUA (dclerk) however it cannot be saved since it is part of the index. It will remain part of the index and it will be used to determine rather other filePro programs need to delete or add records to this index.

IMPORTANT NOTE If you update the key field and that action removes the record from the selection index, then pressing B for browse or using a getnext will continue from that record's location which means it has fallen OUT of the index. It is best when this happens to re-execute the index scan or lookup to get back into the desired selection set index.

Clone Files

Big Hint: Making a clone file can save your act. Keep an audit trail of everything that happens to a particular data set. If you have ever had a user tell you "I didnt do that" and you can see that they were the last person to update the file, it makes for a difficult session. You "think" they might have been the one, but you cant say for sure. Here is the solution. On the files where you have this kind of trouble, or on any file for which you want to keep accurate track of every change, do the following: store exactly what happens to each field of the record from the time it is created; every single change along with date/time/user stamp. You will never have to guess again.

The idea is simple and it can be carried out in a variety of ways. Here is how I do it.

1. Build a "logging file" to capture the log information. This file should have fields for containing the filename, the process name, the field #, the unique ref # for the record, the original data, the changed data and the date/time/who for any file you wish to track. The original data field and changed data field should be 79 characters long. Call this file something like "myprefixlog".
2. First, I copy the file I wish to track with "fpcopy". Only the file structure is necessary, nothing else. Give the new file the name "oldfilename.l" for "log".
3. Next, build an array in the "input" table of the file to be tracked that contains every record in the file.

```
Dim clone[#]:1
```

Where # is the number of fields in the file.

4. Whenever there is a possibility that fields may change, do a lookup free to filename.l and build a similar array over that file and immediately copy the original file/record to the clone file/record.
5. When the record is finally stored, compare the two arrays element by element. If any field has changed, write the original value, the changed value and the date/time/who into your actual log file (myprefixlog).
6. By indexing "myprefixlog" correctly, you can instantly see every change in chronological order for any file.

Here is an example of this code. Your values would have to change to suit one of your own files.

This snippet has to be "atomically" close to the real END statement of your processing table.

If: clone

```
Then: gosub changed
```

```
If:
```

```
Then: end
```

As soon as there is a chance that some fields may change, you must get the free record in the clone file. This is a simple gosub.

```
@keyU if:
```

```
then: gosub clonit
```

The actual cloning code looks like this:

```
If:
```

```
Then: lookup clone=npio r=free -e
```

```
If:
```

```
Then: copy clone
```

```
If:
```

```
Then: dim before[566]:clone(1) ; write clone
```

```
If:
```

```
Then: c(8,0)=clone(@m) ; return
```

Debugger

This Section Contains

Description

Turning the Debugger On

Turning the Debugger Off

Using the Debugger

Debugger Options

Running Operations

Finding Values

Isolating Errors

Testing New Statements

5.0 Enhancements

Description

Once engaged, the debugger allows you to step you through your processing tables one step or one-half step at a time.

A step for debugger purposes is considered to be one if-then element of a processing table. However, it is very important that you realize where the debugger is pausing to allow you to examine the process. It stops after testing the "if" condition of each element and allows you to enter debugger commands until you press RETURN and cause it to execute the "Then" line.

Turning the Debugger On - Three ways:

While on line in dclerk by pressing "b" at the Enter Selection prompt. Nothing will appear to happen, but the next time you press U for Update the debugger will be engaged.

From within processing, by putting the DEBUG command on the appropriate line in the processing table.

Syntax:

Then: DEBUG ON

Example:

```
if:
Then: input q(1,yesno) "Turn debugger on? (Y/N):"
if: q="Y"
Then: debug on
```

3. From the operating system command line (or menu action line) by using the [flag](#) -db.

Example:

```
/appl/fp/programfilename -db
```

Turning the Debugger Off - Two ways:

1. While on line in dclerk or dreport by pressing a "q" at the Enter command prompt.

2. From within processing, by putting the DEBUG command on the appropriate line in the processing table.

Using the Debugger

Steps

1. From the Inquire, Update, Add option line, press "B".
2. Press "U" to update.
3. Debugger appears at bottom of screen.

Layout

```
True-----Prc. Name: automatic
```

```
if:
```

```
Then:
```

```
Enter Command >
```

Prompts

```
Prc.Name: the current table
```

True/False shows the result of testing the current "If" line

```
if: the current element
```

```
Then:
```

```
Enter Command > your debugger command
```

Debugger Options

Looking

L = Look Lists the processing one line at a time without executing any of the actions.

ARROWS Moves you up or down in the table

F = Find label Enter a label name. Program goes to that label and returns to the "look" options. Use arrows or press RETURN to go back to start.

N = Line Number Enter an element number.

P = Print Prints the selected lines.

RETURN Returns you to the line from which you started.

DELETE Returns you to the line from which you started.

Running Operations

RETURN Executes the element shown and shows the next element and the value of its condition. Redraws the screen and displays changed data.

S = Single Step Executes the element shown and shows the next element and the value of its condition.

D = Display Displays results, on the screen, of the processing up to that point.

H = Half Step Like "S" except that the next condition is not executed.

Finding Values

- F = Field** Enter a field number, a dummy field letter, or a system maintained field and the value will be returned.
- E = Expression** Evaluate expressions and show the result. It can be used to see the contents of fields just like the "E" command.
- It can also show you the value of any expression containing fields. (3 * 4) Any valid expression can be tested.
- Another useful expression is a lookup field. Type filename(n) on the command line and the value is returned.
- Literals can also be part of a valid expression so you could test values such as this: (3 + 4/"2.2").
- Date math can also be done using the e command.

Isolating Errors

- B = Break** Stops the processing at the point indicated. You may stop and start the debugger, testing along the way. There are three ways to set break points:
- N = Number** Stop at the designated line number. Maximum of eight.
- C = Condition** Stop when the designated condition is met.
- F = Field** Stop when the designated field changes value.
- C = Continue** Runs the processing table through until it reaches a break point. Changing from one table to another (automatic to input) constitutes a logical break point.
- I = Ignore** Ignores any break points for the remainder of the current record.
- R = Remove** Removes line number break points.

Testing New Statements

- A = Action** Write and test new actions. The "A" command is very useful for testing situations that don't exist on the record you are examining. The best use is for "what-if" testing. You can for instance set a date to a certain day to test particular date processes and so on.
- Q = Quit** Turns off the debugger and returns you to normal program control.
- DELETE (Break)** Use to break out of specific operations.
- F10 Help** Displays the debugger help screen.
- 5.0 Enhancements** F8 (Extended functions) provides additional debugging options as follows:
- 5.8 Enhancements** Break points will now accept long variables (The scope of a longvar is different from a normal dummy field. Technically, longvar is not at a true global scope, and isn't available in the automatic processing table. Declaring it 'g' only will work across records, but not tables, declaring it GLOBAL will fix that, but it has to be matched with an EXTERN in the other prc table.
- L** Displays all lookups within the processing table, along with their status e.g. closed, failed, or open.
- F** Displays all open files.
- **** An asterisk marks which is the current lookup for a given alias.

DROP ALL

DROP ALL is the enhancement added in 4.5 that makes browse lookups function as flexibly as regular lookups. It is now possible to stop the browse lookup from searching the entire file in either direction (forward or backward) based on any criteria. Without DROP ALL, each individual record in an entire file had to be DROPPED individually even after all the desired records were in the browse window.

Example:

```
58 -----
@keyV If: @cd=""
Then: end
59 -----
If:
Then: td="What beginning date? (yy/mm/dd)\n(RETURN=earliest on file)==>"
60 -----
If:
Then: input popup d(8,ymd/) td
61 -----
If: d=""
Then: d="01/01/01"
62 -----
If:
Then: ba="(brw=18 show=pkeep pop=2 prc=prcV fill=asc,top)"
63 -----
If:
Then: bb="[ChkDate Check# Gross FIT FICA SIT Ded
s Net]"
64 -----
If:
Then: bc="*2 *3 *15 *17 *18 *20 *75
*16"
65 -----
If:
Then: ky=1&d
66 -----
brw1 If:
Then: lookup ref= nexprhst k=ky i=a -ng b=(ba&bb&bc)
67 -----
If: @sk="SAVE"
Then: clearb;end
68 -----
If: @sk="BRKY"
Then: clearb;end
69 -----
If: not ref
Then: clearb;end
70 -----
If: Enter will close down browse
Then: clearb;end
71 -----
prcV If: *prcV
Then:
72 -----
If: ref(1) ne 1
Then: drop all ; end
73 -----
If: ref(2) lt d
Then: drop all ; end
74 -----
If: ref(2) gt d in some cases DROP ALL does it all!
Then: drop all after ; end
75 -----
If: ref(2) lt d and BEFORE and AFTER become superfluous
Then: drop all before ; end
76 -----
If:
```

Then: end

DROP ALL

Allows all remaining records in a file to be quickly dropped from a browse window.

Syntax

DROP ALL BEFORE

DROP ALL AFTER

DROP ALL

Notes

When using BEFORE and AFTER, the record you are on is also dropped.

Examples

You want to do a lookup browse to see all the invoices for a particular month. Field one of the file is the customer number, field two is the date.

Then: Input dt(2,0) "what month do you want to see enter 01-12" <dt

If: dt gt " "

Then: lookup inv k=1 i=a -exm b=("brw=8 prc=date file=asc, top)*1 *2

Then: end

date if:

Then: zz=mid(inv(2),"1","2")

If: zz lt dt

Then: drop all before

If: zz gt dt

Then: drop all after

Using this logic, only the records that are in the requested month show and the data processing did not have to be done on every record for that customer.

HELP Screens

The Help utility

filePro allows for the creation of extensive Help screens, specifically designed for each application you build.

More importantly, Help can be provided in several different situations.

Types of Help

1. Press the Help key (termcap designated, ansi=F10) while within a field and receive help information for that particular field.
2. Press the Help key while at the option line in Inquire, Update, and Add and receive help for the current screen being displayed.
3. Specific Help can also be displayed from the processing table based upon varying conditions or triggers. By using @whp### processing or by using the HELP command or any combinations, you can provide very elaborate and/or useful Help for your applications.

Steps

1. Use a text editor or word processor to create an ASCII file named "help" in the directory of the specific "filepro file".

Unix: vi /app/filePro/filename/help

Windows: edit C:\FILEPRO\FILENAME\HELP

2. The contents of the help file determine what help is given to the user and can also specify at what points.

Help for a field

##Fn

Helpful text ...

where "n" can be a real or dummy field

Help for a screen

##Sn

Helpful text ...

where "n" is a screen number or name

Help called from within processing

##helpname

Helpful text ...

where "helpname" can be any name of up to 16 characters. Named help screens are accessed from a processing table.

HELP for any type of entry is ended by the occurrence of another entry (##xn), or by the end of the HELP file itself.

Example - Help on field and screen triggers

##F28

Please fill this field with the name of the purchasing agent for the customer being added. If you do not know the name, please enter N/A.

##S3

You only need to enter information on this screen if the account has a bad credit rating. The credit rating is displayed on the bottom left of this screen in RED.

##AA

Please enter the amount of the overdue charge to be added to the customers invoice.

Example - Help Displayed From Within Processing

##helpname

Before you print this invoice, check to be sure the proper paper is in the printer. Align the print head to the exact top of the paper.

Example: From the Input processing table

@whp27 if:

then: HELP "helpname";screen ,27

or

@wlf27 if: 27 = "?"

then: HELP "helpname";screen ,27

HELP can be multiple pages.

If the HELP for any of these entries were several pages long, filePro would allow the operator to scroll backwards and forwards through the HELP until he/she presses RETURN to re-enter the application at the point they left.

HELP can be indexed.

If you want to include indexed help, add a line to the section of the help using prefix "@@@" keyword1, keyword2, etc.

Figure AC-01 is a portion of the "Define Processing" help screen for commands ABS and ACCESS which depicts the use of @@@ ABS, ACCESS to create indexed entries for these commands.

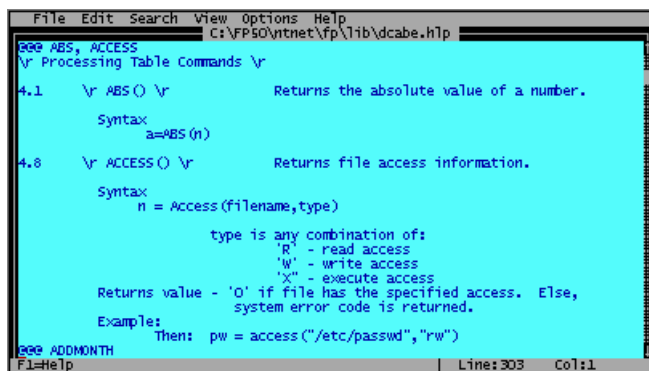


Figure AC-01 - Indexed Help

The end-result of these indexed help entries is depicted when pressing [F9] after invoking help in " Define Processing ". See figure AC-02.

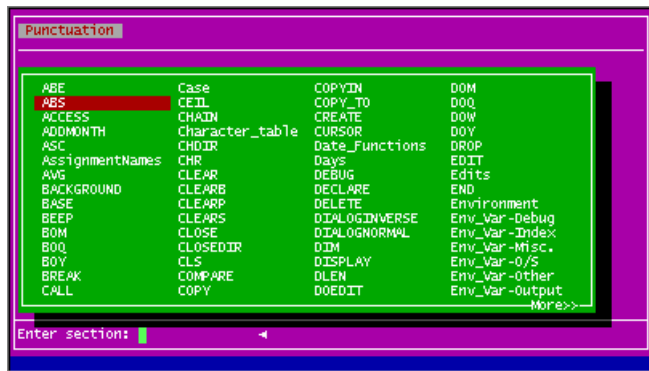


Figure AC-02 - IndexHelp Displayed

The HELP keys can be changed in the termcap entry with an entry for the sequence and the label. See the [Terminal Guide](#) section of this manual.

Finally, this is the actual testing and logging code.

```

If:
Then: show popup "Please wait while files are being updated."
If:
Then: dim after[566]:1
If:
Then: i(3,0)="1" ; yy(8,ynd/.g)=@td ; ti(8,time)=@tm
chgloop If: i gt "566"
Then: delete clone;clears;return
If: before[i] eq after[i]
Then: i=i+"1";goto chgloop
If:
Then: lookup log=npilog r=free -ep
If:
Then: log(1)="npio"; log(2)=412 ; log(3)=yy ; log(4)=ti ; log(5)=@id
If:
Then: log(6)=i ; log(7)=before[i];log(8)=after[i]
If:
Then: log(10)=1 ; log(9)="npio/input"
If:
Then: write log ; i=i+"1" ; goto chgloop

```

When someone has a question about this file, you can look in the logfile based on the file you are tracking and its unique reference #. Each change will be cataloged. You can make a screen for your logfile that tells the whole story. No one can dispute the documented truth.

Transaction Log Screen

```

In file: !1 Ref#: !10
Unique record#: !2
Field#: !6 was changed by !5 on !3 at !4

From
!7
to
!8

```

Note: If you have a slow machine, you could build two other arrays of only 1 element each for the total number of characters in the file you want to track. Then, before doing the changed loop, test each array against each other. If there is no change between the one element arrays, there obviously wont be any change to any individual field and you have nothing to write to the logfile. This first big test is not necessary on a fast machine. Checking the arrays on a 566 field record with almost 5K of data per record takes up minimal time.

Integrating Concepts

Thanks to John Esak for providing this general overview.

In [rd]clerk: When filePro retrieves a record, whether you get it by pressing the down arrow in dclerk, or going through scan or an index, ... whether you are coming off the browse screen into the full screen mode or already in full screen mode ... before you see the record on the screen, the automatic processing is run against the data in that record. Then, after the first encountered "end" on the automatic table, the @entsel processing is run. In other words just before the cursor is put at the "Enter Selection" prompt, the @entsel processing is run.

Now, let's say you are sitting at the Enter Selection prompt and looking at this record which has just been run through those two processes and everything looks fine. You press "U" to update the record and filePro will immediately retrieve the record again and run and run the automatic processing again before it puts your cursor into the first field on the cursor path. It does this because the record may have been altered during the time you've been sitting there perusing it at the Enter Selection prompt. Good idea... now you have the latest copy of the data, properly adjusted and bashed about by your automatic processing, your cursor should go into the first field on the cursor path... but no, first it has to run through any @wefxxx that exists for that field! So, just retrieving the record and pressing "U" to update it, you've run it through automatic twice and @entsel twice... and now @wefxxx. No problem, both AUTOMATIC and @ENTSEL processing should have NOTHING on them that alters real fields. This will be "noted" and "ignored" by any @entsel processing that tries to alter a real field... but you must be smart enough not to do it on the AUTOMATIC yourself... filePro will actually let you alter a real field on AUTOMATIC, but don't do it! It is an EXTREMELY, HORRIBLY bad thing to do.

Why? Simple... the record is not "locked" by AUTOMATIC processing, so you can't be sure someone else doesn't also have a copy of that record, changing it at the same time you are... Meanwhile, back to the flow of processing in [rd]clerk. You will note, that after retrieving a record, pressing "U" to update it, at no time have we yet run any INPUT processing. The INPUT table processing pointer is still firmly clamped at line 1 or loosely called the TOP of the INPUT processing table. It has moved some other processing pointers around, the one for the AUTOMATIC table, the one for the @entsel processing, even the one for the @wefxxx processing have all been moved through their little processing snips, but the main INPUT processing pointer has not moved at all. It is at "the top" or line 1. The main INPUT processing gets run only when the program or the user say it's time to "store" the record. This situation happens 99% of the time when the user presses the key designated for this purpose, in Unix usually ESCAPE/ESCAPE, in Windows just ESCAPE (This key can be changed...) The other 1% of the time, the running of the INPUT table can be forced by actually issuing the "SAVE" command. (I do actually use this command, but it is only in very, very, rare and usually special circumstances.) In general, the INPUT table is left waiting, never to run, until the user presses the key himself to do it.

Why explain all this? Because, in that 99% of the time, filePro is going to store the record and run the INPUT processing table for you "automatically", you don't have to engineer it with the "SAVE" command. Even if your cursor is in various fields when the ESCAPE/ESCAPE is pressed... filePro will still run the @wefxxx snip as usual, then just after the ending command of that @wefxxx is encountered, the INPUT table picks up from wherever its pointer is positioned... usually the "top" of the table. The times you may NOT want to run the @wefxxx code again before running the INPUT table are some of the times you might actually want to put the "SAVE" command on the very first line of the @wefxxx and act on it before the other stuff. Like in your example.

```
@wefxxx if: @sn eq "something"
then: SAVE
if: 'other stuff that happens if they are
then: 'not on this screen.
if:
then: end
```

Now, you mentioned setting them back to the beginning of the INPUT table with the "RESTART" command. Of course it will do this, but there is a not so subtle difference. The RESTART command would put there cursor back on the screen at the beginning of the cursor path and move the INPUT processing pointer to the "top", i.e., line 1. BUT, the ball would be back in the user's court. The SAVE command would not give the ball back, but simply start running the INPUT table.

Why is all this important? Because, there is one more thing which must be added to the mix to get to the reason for the behavior you saw in the debugger. AFTER the user has pressed the SAVE key (escape/escape), the INPUT table is run, yes, but immediately after it encounters its first "end" statement the AUTOMATIC table is run one more time... why? Because you may have changed some data on the record and you want to massage it again in the same way you did when you first retrieved it but now with the newly updated data. Hey! pretty good, so now we finally get to see everything... nope... there's that pesky Enter Selection prompt... and what runs just before the cursor is placed there? Yup! @entsel processing. Now, you've got the whole picture, or a lot of it. The flow of processing is very important, because it is one thing you can count on working the same way, every time, all the time.

Now, how is it you saw the AUTOMATIC table running first. All I can guess is you had possibly forgotten that first "end" on the INPUT table. Then when the user pressed ESCAPE/ESCAPE you saw the @wefxxx processing label come up and thought it was running because they were leaving that field... well, yes, it was, but it was running because filePro first ran your @wefxxx snippet as it should, which you saw... then it ran the INPUT table from the top because you had put an explicit "SAVE" command as the first thing to do... in which case it would now be looking just the same in the debugger... my guess is you had to press ENTER one extra time before you saw the AUTOMATIC table running in the debugger and you just ignored this. I do this all the time while in the debugger. You press the ENTER and think you somehow didn't really press it because you are seeing the same thing twice... in this really strange case you've bumped into, that is exactly what it would do. You're right very confusing. Putting an "end" as the first > line in the INPUT table would prevent this. Unless, of course, there actually IS some INPUT processing that needs to be run... then be sure there is an "end" somewhere along the way, just so the processing doesn't "fall" into the @wefxxx processing snippets by accident.

Okay, all clear as mud? good. :-) The reason's I've gone all through this stuff (which I realize you and most people may know) is this... To reiterate, just once more, I promise... :-) The @entsel processing and any other trigger processing (@wefxxx, @wlfxxx, etc.) in clerk have to be put on a processing table somewhere. filePro mandates that this trigger processing has to be on the INPUT table. So, where does it all go??? It must come after an "end" statement. The way filePro differentiates "what" is the INPUT processing part of the INPUT table and what are the other trigger parts of it like @wlfxxx, @wefxxx, @wukxxx, @wlpxxx, etc.) is to set aside all the lines from line 1 until the first encountered "end" statement for it. (This is really a fantastic thing and sets up a very useful symbiotic relationship between the two types of processing... even though they are NOT connected at all in any specified way, unless you arrange it. More about this later.) You can badly obviate and break all this wonderful schema by not remembering to put an "end" statement to delineate the END of INPUT processing. :-) A troubling "statement" in itself. Yes. But the simplest demonstration is this. If you had the following as the entire INPUT table:

```
@keyZ
1 then: show "@ am @keyZ processing."
if:
2 then: end
```

It would only work badly, in that it would work twice... which is sort of a screw-up all around. It would work if they press Z, but it would also work when they press "U" and then ESCAPE/ESCAPE... which you probably do NOT want.

Whereas, doing the way we all know it should be done:

```
1 then: end
@keyZ if:
2 then: show "@ am @keyZ processing."
if:
then: end
```

The above is perfectly fine INPUT table (with NO input processing at all, but just some trigger processing.) It doesn't mean there is actually no INPUT processing, because INPUT processing still gets run, it is just that the only thing INPUT processing is telling filePro to do is "end". That step actually does get executed, it just obviously does nothing. If it weren't there, filePro would run the stuff you only wanted to be run when the keyZ was pressed. It would run because INPUT processing starts from line 1 regardless of what the label says. So pressing key Z would make it show " I am at the @keyZ processing." and also pressing "U" and ESCAPE/ESCAPE would do the same thing... probably not what you meant by the code. filePro is often knocked for the fact that the first line of the INPUT processing table is often END. :-) Everyone knows that this simply means there is NOTHING to be done when the record is saved and written out.

Now, let me write some ABSOLUTELY terrible, really BAD code to demonstrate that symbiotic thing between the INPUT table processing and trigger snippets. You might actually do something like this... but don't. It's just an example of why having all these things on the same table can be VERY confusing as you said.

```

then: show "@hello, I'm in INPUT processing now."
totals if:
then: 14=7+8+9+10+11
if:
then: display
if:
then: end
@wlf12 if: 12 eq "Y"
then: goto totals
if:
then: end

```

The programmer here (who shall remain nameless) is doing something that filePro will allow him/her to do, but it only works because filePro does what it does without flinching... and you are just lucky if you write code like this and can figure out later when things get more complicated what's going on. Sure, you are ending the @wlfxxx processing with an "end" statement, one of the 5 commands that will properly end the @wlfxxx snippet, but you've done it sort of sneakily by co-opting the end of the INPUT processing table. This would all be so much better written by doing something like this:

```

if:
then: show "@hello, I'm in INPUT processing now."
if:
then: gosub totals
if:
then: display
if:
then: end
totals if:
then: 14=7+8+9+10+11
if:
then: return
@wlf12 if: 12 eq "Y"
then: gosub totals
if:
then: end

```

Now, you still get to use stuff that may be shared with the INPUT processing, but you are doing it much more clearly and explicitly... i.e., in a way that will be much easier to read as the processing gets more and more complex.

Which brings us to the next level.

```

top if: @sn eq "4"
then: form "something"; screen 5; goto top
if:
then: screen 9
if:
then: end
@wlf12 if: 12 eq "Y"
then: screen 4
if:
then: end

```

Let's say the user starts out on Screen 1 and field 12 is on screen 1. Different things will happen based on whether he puts an X in this field or not. The relationship between the INPUT processing and the trigger processing can be non-existent, or it can be very tightly integrated and interdependent as shown above. It's all in what you arrange... knowing the flow of which processing runs when... what *ends* an @wlfxxx snippet.

There are 5 things which do this, END, SKIP, SCREEN, RESTART and, EXIT. Remember the not-so-subtle between RESTART and "goto top" being that RESTART gives control back to the user, goto top doesn't. There are zillions of variations to all this, but here is one important one. If you are using @key processing, the flow is this... first the AUTOMATIC table is run, and then the @key processing is run until an "end" is encountered. INPUT processing is not considered at all. You could, of course, invoke INPUT processing from @key processing by using the RESTART command.

```

@keyT if:
then: do some stuff
if:
then: restart

```

This would END the @key processing snippet, and deposit the user into the first field on the cursor path. The next time they press ESCAPE/ESCAPE, the INPUT table would run from the top. Interesting, huh? (Remember also, that after this, the AUTOMATIC table and the @entsel snip would run as well... :-)

Okay, hope all of that helped and you aren't more hopelessly confused. If I've messed something up in my description, forgive me, it is late after a long day. Oh yeah, just one last thing... what I was going for at first to help describe another reason why the AUTOMATIC processing appeared to be running first. If you have already run _part_ of the INPUT processing... that is where the INPUT processing pointer will be, and where processing will begin again when ESCAPE/ESCAPE is pressed (or you hit that "SAVE" you had put in the @wlfxxx snip.) Consider you've done something like this:

```

1 then: do a lot of stuff; do more stuff
  if: something equals something
2 then: some more stuff
  if:
3 then: screen 7
  if:
4 then: end
@wlfxxx if: @sn eq "7"
5 then: form "blah"; SAVE

```

If the user has pressed ESCAPE/ESCAPE once already before ever entering field "xxx", then the INPUT processing table has "already" started running and has stopped at line 3, the screen 7 command. Assuming field "xxx" is on Screen 7 and the cursor goes there, when the user presses ESCAPE/ESCAPE this time, the INPUT processing will start up where it left off, which would be the next command immediately after the screen 7 command in this case just ending the INPUT processing. You would only see the end command in the debugger and then the AUTOMATIC processing would appear.

Obviously, we can't tell exactly why you saw AUTOMATIC processing running first, but I can assure you it was because you were somehow telling filePro to do this.... In other words, it actually did run its normal flow of things and it just appeared to not be running the INPUT table... (again, I think there might have been a doubling of the @wlfxxx because of no "end" statement above it... and things just appeared messed up.

Good luck. Regardless of what it was. It seems you've fixed it with a RESTART, I hope this helped a little anyway.

LISTBOX Command

The LISTBOX command allows you to put up a series of choices and capture the element number of the one chosen by the user. You can use this number to GOTO or GOSUB through routines associated with each individual choice.

Here is something useful you can do with LISTBOX and printing.

After a new invoice record is placed on file and the user presses ESC ESC to store the record, the question is asked, "Do you want to print an invoice?" If the answer to this question is yes, the form is printed. In the past, all invoices printed on the same main printer in an outer office. This is inconvenient. If each user has access to his own printer, and this personal printer is named after him, personalized printing can occur very easily. You may find that an adaptation of the following will work for you too.

LISTBOX puts a window up on your screen. The size and location of the window are preset by default, but can be altered to suit your needs. The contents of this box are really a menu of choices, with the standard filePro highlighted bar cursor and selection mechanism. (You can press the first letter of any of the choices and the cursor jumps right to it.) The choices are elements of an array, which you fill on the processing table. They can be variable and change each time you call up the LISTBOX. In this example, they are variable based on the login name of whatever user is operating the program. (In this model, provisions are made for only four users and "hard-coded" into the table. The implementation at Guru Headquarters is actually more elaborate, and dynamically allows for new users/printers, but this example is meant only to illustrate a good use for LISTBOX. This function is more clearly presented in this manner, and not obscured by the other code.)

Besides the ability to print an invoice (or any form, of course) on any of three printers, the added sophistication of defaulting to the users own printer is very friendly. When Robin is logged in at her terminal, she sees her printer as the first (default) choice and can just press RETURN under normal printing conditions. If, for some reason, she is logged in at Karens terminal and wants to print, as always, at her own printer, she still just presses RETURN as usual. She would need to make a conscious decision to move the cursor bar or pick a different selection by letter if she doesnt want *her* default. (You would be surprised how many times it is necessary to print on anothers printer... out of paper, letterhead isnt loaded, printer is broken, someone forgot to buy new toner, printer isnt warmed up yet... etc., etc. If you cant think of any other reasons, call Robin or Karen; theyll give you a bunch more. -ed)

Processing: input

```
If:
Then: your input processing goes here
If: Build an array that holds 3 elements (printer names)
Then: dim printers[3](10)
If: @id="karen" fill array based on person.
Then: printers["1"]="karen";printers["2"]="robin";printers["3"]="john"
If: @id="robin" making sure the running user is the first element
Then: printers["1"]="robin";printers["2"]="karen";printers["3"]="john"
If: @id="root" or @id="john"
Then: printers["1"]="john";printers["2"]="karen";printers["3"]="robin"
If:
Then: gosub doprint
If:
Then: end
```

doprint If:

```
Then: cls("21","4")
If:
Then: input q(1,yesno) "Do you want to print this invoice? (y/n) "
If: q ne "Y"
Then: goto label
If:
Then: show "\r What printer should be used? \r"
If:
Then: pr(1)=listbox(printers,"",",", "20", "56")
If:
Then: printer type "dumb"
If:
Then: printer "lp -s -d {printers[pr]} ; form "invoice" ; printer reset
If:
Then: cls("21","4")
```

label If:

```
Then: input q "Do you want to print a label? (y/n)"
If: q ne "Y"
Then: return
If:
Then: printer type "dmp2100";printer "lp -dlabel -s"
If:
Then: form label;printer reset;return
```

Logtext

There is a simple filePro-based message saving technique that can be used with the above audit trail technique, or for a myriad of other reasons.

This is a filePro command that makes it easy to keep various audit trails. Perhaps it was designed to aid in doing something like "clone" files, but it has many other uses.

The LOGTEXT command allows you to direct messages into a logfile from within clerk or report. Your output will go into a file defined by the environment variable LOGFILE. If you set the environment variable LOGAPPEND, the data you send to the file will be appended at each instance of the LOGTEXT command, otherwise the file is cleared and new data overwrites the old data.

Syntax:

```
LOGTEXT "message"
```

Where "message" is any valid expression. Be careful about quotation marks.

Example:

```
If: 14 gt CL
```

```
Then: logtext "Credit limit was exceeded by"<@id<"on  
      record#"<@m<"on"<@td<"at"<@tm; end
```

The LOGFILE variable must be set before using LOGTEXT.

Lookup Dash

Usually when you do a "lookup" it is because you want to retrieve data from, or place data into that "looked-up" record. Usually you look up records from other files. Occasionally you must look up records in the same file you are "standing in", but still, you will want to retrieve some data or write some data to the record you've looked up. Not so with "lookup -". This special lookup doesn't give you a chance to retrieve data or place data into the lookup record, it just MOVES you there immediately. What you do at that point is up to you, but you are no longer "standing" where you were. "Lookup -" can not move you to a record in another file only to another record in your current file.

After putting the following four lines at the end of any input table (choose any file YOU are comfortable using), then pressing the "G" key will show you exactly what "lookup -" does. It physically (from your point of view) moves you directly to the looked-up record. Do NOT pick a record number higher than the highest record in your file. This is just an example of programming to demonstrate a feature, and it is not completely foolproof. Try this out and then read on.

```
If:
Then: your processing is above here
If:
Then: end
```

```
@keyG If:
Then: input re(6,0) "Goto what record?"
If:
Then: lookup - r=re -e
If:
Then: end
```

You must modify the above piece of processing to look like the following if you want to have something you can use. The GOTO record number idea is not very useful, but it helps demonstrate graphically how "lookup -" works.

To make this piece of code very useful, modify it to read as follows:

```
If:
Then: your processing is above here
If:
Then: end
```

```
@keyG If:
Then: input re(6,allup) "Goto what Account Code?"
If:
Then: lookup - k=re I=a -nx
If: not -
Then: show "@sorry, not on file";end
If:
```

(You must have index.A built on account code). To use this processing, you could either press "G" while at the Enter Selection prompt of any record, (which would do the same thing as backing out to the index selection for account code and entering it from there) OR, you could access this @key from within processing. Whenever you want the operator to be able to move to another account code, just use:

```
If:
Then: goto @keyG
```

Since "lookup -" saves out the record you are standing on, using this idea would be like pressing SAVE for the operator and then backing them out to the account code index. Also, it can be done from a @when leaving field routine which would save many keystrokes for someone who is doing a repetitive updating task amongst varying account codes.

NOTE: "Lookup -" can be used (with caution) on the automatic table. Once I used it on this table to keep people from seeing various records based on their user ID. Its beyond the scope of this discussion, but a description of the process, which you may be able to use, is as follows: An army base had a filePro file that had varying levels of access allowed according to the operators rank. Captains could see captain stuff, corporals could see corporal stuff, privates saw very little and so forth. We gave each user a login ID (on the *nix system) which had a number appended to his name, john200, bill300, mary400 etc. This number represented their rank. I could then use their login ID to govern access privilege to this filePro file. I used the @id system maintained field to match against a list of privileges in the automatic table. If the record contained a 300 (in the access privilege field) and the operator had a 300 or higher in his login ID, we let him stop on the record. If his login ID contained less than a 300, we "lookup dashed" him to the next record in the file. By capturing UP and DOWN arrows and testing which way they wanted to go, along with keeping track of the way we put them into the file in the first place (which index and no browse mode allowed), we were able to offer a slightly more secure filePro than the average bill of fare. Anytime they came to a record they weren't supposed to see, the operator was gracefully moved to the next available record. They were not even aware there were records they weren't seeing. The managers loved it and have made use of the idea in many other things they've created since then. -Ed

Ultra Fast Selection With "LOOKUP -"

There is no question that the most spectacular use for "lookup -" is in conjunction with scan/selection sets. A picture is worth a thousand words, so picture this...

You have an invoice file with 48,000 records. Each day you generate about 100 new invoices. At the end of the day you want to print just the newly added invoices. Without "lookup -", your invoice printing report takes 6 minutes just to select the correct records for printing. With "lookup -" the same report selects the proper records in 4 seconds! Imagine this kind of savings throughout the day in many different reporting situations. How much time do you spend watching the Records Read/Selected at the bottom right of the screen count up to the top of your file, before you see the Generating Output message in the middle of the screen start counting down? Eliminating (or drastically reducing to near zero) this selection time WILL change your life.

Making this magic happen involves using a scan/selection table (a -v table as we users have taken to calling them) and requires building an index on the field we are going to select by. For future reference we will call this the "scanning index".

Example 1: (Select by one criteria, i.e. status)

First make sure an invoice (or test) file contains a one character "status" field (or flag) that tells us whether the invoice has been printed yet. (Add a 1 character, allup field in Define files to a test Invoice file to serve as this flag.) When you create the invoice, You must be sure (on the input processing table) to fill this field with a "W" to represent that it is waiting to be printed. On the processing for the output report that actually prints these invoices, You must change this field to a "P" to represent that it has been printed. By selecting all records that have a "W" in this field, we will be getting all the newly created invoices (or those waiting to be printed, whether they are new or not). Obviously once you print these invoices the status field will Set filled with a P" and wont be selected next time... which Is just what we want to happen.

Put the following into an output processing table called "flagsel". (You can name these lines whatever makes more sense for you, we pick names only for their descriptive value to the lessons.)

(This is not the most elegant way to write this table, but it is the clearest way to show what happens when "lookup -" is used on a sort/selection table.)

```
Processing Name: flagsel
If: aa ne ""
```

```

Then: goto getit
askit If:
Then: input aa(l,allup,g) "Enter Status Code"
If: aa eq ""
Then: goto askit
getit If: l lt aa
Then: lookup -k=aa i=h -ng
If: l gt aa
Then: hv(8,,g)="Z" set HV to the highest possible value index can hold
If: l gt aa
Then: lookup -k=hv i=h -nl Get the last indexed record on file
If: l ne aa
Then: end
If:
Then: select;end

```

To implement this processing, along with your output report that actually prints the invoices, you would use the following command line:

```
dreport file -f invoice -v flagsel -a -ih
```

An English description of what happens when you execute this type of processing is as follows. The dreport program gets the first record in the "A" index. Line 1 is NOT executed because at this point "act" is not equal to anything. The 2nd line then asks the operator what status flag to select records by. Line 3 forces them to answer the question and not leave it blank. Line 4 now tests the status field on the record the program is currently standing on. If it is less than the status required, it does a "lookup -" to the next greater record in the Index (which is the index we built on the status field). "Lookup -" causes the processing to save out the current record and move to the requested record. Now the processing table runs to the top and starts processing all over again on this record. Now, since "act" is a global variable, it holds its value across record changes. This being true, line 1 is NOW executed and the program bypasses asking the question again. The Process tests the status field again. This "lookup -" on line 4 will be executed over and over again until the first record in the index that is not less than the status field is obtained. At this point, line 4s "if" line fails and line 5 tests if the retrieved records status flag is now greater than the desired status. If it isn't, the process falls to line 6 and fails there also. At line 7, the test will be true, assuming you have some records with a "W" status. If line 7 tests false, the records status must match the one desired and line 8 "selects" the record for further processing by the output report which will actually Print the invoice once all this scan/selection is finished. The next record is retrieved and it starts all over again.

The magic trick is hidden in lines 5 and 6. As soon as a record is retrieved from the index that is greater than the desired status, the lookup key is changed to the "highest possible value" for this type of index, and another lookup is done. But this time with the "-l" option, which tells filePro to get either an exact match for the lookup key or the "next lowest value" in the index. Since the highest possible value for an allup field is "Z", we try for this. If it is there, it will fail the next "if" test and the process ends having selected all the required records. If there is no "Z" record the next lowest one is retrieved and the next "if" line fails anyway. Both cases cause us to be moved directly to the LAST RECORD IN THE INDEX. There are no records left to process so the selection process is over. All the "selected" records are then handed to the output table for its processing, one at a time as usual.

As you can see that "lookup -" is jumping from the first record in the index to the first one that matches your criteria, then every record that matches from that point is "selected" for output processing. As soon as the criteria no longer matches, "lookup-" jumps you past every other remaining record in the index to the last record on file, and scan selection just ends normally.

Example 2: (Select by a range of values, i.e. invoice printed date)

Since this idea is SO powerful and time saving, it is worth exploring the myriads of ways it can be used. The next table is very similar to the preceding one but will select a range of dates.

It should be self-explanatory now, and hopefully you can adapt this to other search criteria besides dates.

To make the below table function correctly, use the date the invoice was printed as your scanning index. In this example as your output report prints each invoice, this field must be filled with @td (today's date). By selecting all records that have nothing in this field we will be retrieving all the invoices newly created today. (Obviously once we print the new invoices, this field will get filled and they won't be new anymore... which is just what we want to happen.) By using a date instead of a flag to represent the printed date, we can later reprint all invoices printed on a particular date, or do a report totaling all invoices printed that day, etc. Moreover, we can select a range of dates and do the same things.

Put the following into an output processing table called "datesel", and be sure there is a field to hold printed date in the file. We will again use field 37 to represent this key held. Build index H on this field and use the following line to execute the selection:

```
dreport file -f invoice -v datesel -a -ih -h "Select By Printed Date"
```

Processing Name: datesel

```

If: da ne ""
Then: goto getit
stdate If:
Then: input da(8,mdy/,g) "Enter starting date?"
If: da = ""
Then: goto stdate
endate If:
Then: input db(8,mdy/,g) "Enter ending date? (RETURN=starting date)"
If: db = ""
Then: db=da;goto getit
If: db lt da
Then: goto endate
getit If: 37 lt da 37 is printed date field
Then: lookup -k=da i=h -ng
If: 37 gt db
Then: ky(8,mdy/,g)="12/31/99" set TV to highest value index can hold
If: 37 gt db
Then: lookup -k=ky i=h -nl get last indexed record on file
If: 37 lt da or 37 gt db

```

Then: end

If:

Then: select;end

Getting More Out Of It

Once you've mastered this technique and have a few basic tables to play with, try doing more than one selection on the same table. For instance, ask the operator which group of zip codes he wants labels printed for... his response could be 33000 to 39999 and 60000 to 69999 and others. Once you have all of his request, start a loop that puts the first range into variables and selects them. Instead of "ending" when the first range is fully selected, change the variables to the next range pair and "goto" another iteration of the select loop. Do this as many times as the operator gave you parts for the request. You must, of course, specify a limit to how many requests you allow the operator on one table. Store his choices in as many variables as you need or want to allow. Arrays help in managing routines like this.

MENU Command

Another factory pre-built array construction is the MENU command. It is similar to LISTBOX in a way, but much more familiar because it mimics the filePro menus that you use all the time. It does this as a processing command. In other words, you can put up a "menu" of choices and act on the response. There is a major caveat regarding the MENU command. If you are going to use a processing menu, you must be aware that the user does not see much difference between one of these menus and a real filePro menu. While a processing MENU is up on the screen, the record is "locked" because the user is still actually in INPUT mode. The user is really sitting at the point in the table where you are waiting for his response. If he decides to get up and go to lunch while this type of menu is on the screen, no one else will be able to use that record until he comes back and finishes his processing. This is very important to understand and program around. Either that or use processing menus very sparingly.

If:

Then: gosub domenu

If:

Then: end

domenu If: * domenu

Then: BREAK OFF

If:

Then: cls

If:

Then: dim mnu[12]

If:

Then: mnu["1"]="Company Setup and Maintenance -"<1

If:

Then: mnu["2"]="A: Company Data"

If:

Then: mnu["3"]="B: Fiscal Year"

If:

Then: mnu["4"]="C: G/L Account Types"

If:

Then: mnu["5"]="D: G/L Chart Of Accounts"

If:

Then: mnu["6"]="E: Checkbooks"

If:

Then: mnu["7"]="F: Accounts Receivable"

If:

Then: mnu["8"]="G: Accounts Payable"

If:

Then: mnu["9"]="H: Payroll"

If:

Then: mnu["10"]="I: System Generated Numbers"

If:

Then: mnu["11"]="J: Miscellaneous Defaults"

If: Here is where the menu gets put on the screen

Then: menu mnu do 1,do2,do3,dosys,do4,do5,do6,do7,donums,do8

If:

Then: cls

If:

Then: input q(1,yesno) "Are you sure you are done?"

If: qq ne "Y"

Then: BREAK OFF;goto domenu

If:

Then: BREAK ON;return

do1 If:

Then: screen 1;goto domenu

do2 If:

Then: x=24;screen 2;gosub suretst

If: 24 ne "" and 168 = ""

Then: 51="0";52="0";53="0";54="0";161=24;162=47;163=24;164=47

If: 24 ne "" and 168 = ""

Then: 165=24;166=47;167=24;168=47

If:

Then: goto domenu

do3 If:

Then: fl="";dim new1[1](80);11;dim old1[1](80)

If:

Then: dim new2[1](16);95 ;dim old2[1](16)


```
If:
Then: old["1"]=new["1"]; old2["1"]=new2["1"]
If:
Then: screen 3
do3a If:
Then: write;gosub suretst
If: fl=""
Then: goto domenu
If:
Then: dimtyp[1](80):11
If: typ["1"]=""
Then: goto domenu
do4 If:
Then: gosub cash;goto domenu
do5 If:
Then: ...
do6 If:
Then: ...
do7 If:
Then: ...
do8 If:
Then: gosub get8;screen 8;goto domenu
```

NOTE: As of 5.7.04, you can now run a menu command from outside of filePro

Example: p menuname -Xn

This will run menu option n, and then exit.

Negative number in "Number of Forms" field

Picture a check stub / check. The check stub is 22 lines and the check is 22 lines, for a report length of 44. You have tractor checks. (This is also adaptable to laser by just making the page length 60) where the check stub is the detail of the report, listing the items to be paid, and the check is the subtotals section of the output. You always want the check part to start on a certain line for it to print on the form correctly.

You analyze the check stub and determine how many detail lines will fit on one page (there is usually some headings and stuff printed that might reduce the lines that fit). You must also know if you are printing one line or two lines per detail line (record in filePro). Now you measure the space you have for detail lines, divide by number of lines / record.

So, you may have room for 10 one-line detail lines, or 5 two-line detail lines. So you would set it to -10 in the first case, or -5 in the second case.

Another example would be for a pre-printed invoice form, where it has at the bottom a pre-printed place to put the total of the invoice. If the invoice is designed as a report in filePro which prints more than one charge (record) on one invoice (page), and you want the total to print at the bottom no matter if the invoice was for 1 item or 10 items, then you would use this feature.

The "Number of forms down" will allow you to indicate how many detail lines fill a form to reach the subtotal location on the paper form.

Grand totals do not apply in this situation. Grand totals are always treated as an additional page if you do not tell it to print on the last page, or added after the subtotals when you do tell it to print on the last page. A check or invoice would never make it appropriate to print the grand totals with the subtotals. That would be a summary page that might be printed to tell how much was invoiced or printed. (In the case of the checks, it would waste a check.)

TRY IT

- Make a report output, page length 60/66 (depends on what is your standard paper size.)
- Add headings, detail line (one line) , and a subtotal section. On the output setup, put -10 for forms down.
- Select 5 records that would be in the subtotal section and print.
- Try selecting and amount in that subtotal group from 1 to 10 and see what prints.

Now filePro is certainly not having any trouble printing the subtotals it is just where it is to be printed that is affected. Other processing methods might now allow similar results with different methods.

Thanks to Nancy Palmquist for providing this great trick and example.

O/S FILE I/O FUNCTIONS()

filePro has a full set of I/O functions that allow access and manipulation of operating system files. These functions can be used on any processing table.

Note: I/O functions cannot be used to access any file with a name formatted as a filePro file. For example, you cannot access a file named screen.1, index.A or key.

Refer to the following I/O functions links.

CLOSE	Close an open file
CREATE	Create a file with a name and optional permissions
FILESIZE	Get the size (in bytes) of a file
OPEN	Open a file for reading/writing
READ	Read bytes from a file
READLINE	Read bytes from a file up to a new line (or end of file)
REMOVE	Remove an existing file
SEEK	Move to a position in an open file
TELL	Get the current position of an open file
WRITE	Write bytes to a file
WRITELINE	Write a line to a file and append a newline character

filePro PDF printing

filePro version 5.7.03 introduces the ability to print directly to a PDF document. There are two parts to this feature – specifying the destination, and the new FPML "filePro markup language".

Specifying the destination

At its most basic, PDF printing is accomplished by specifying "PDF:filename" as the output destination. This will cause the output to be sent to the specified filename in PDF format. Note, however, that any print codes must be in FPML format, as filePro will not translate other print code languages into PDF.

There are several options for the "PDF:destination" specification:

- PDF:filename

Sends the output to the specified filename.

- PDF:>filename

Sends the output to the specified filename.

- PDF:|command

Sends the output to the standard input stream of the specified command.

On Windows platforms, there are several additional options available:

- PDF:[open]

Creates a PDF document in the user's "temp" directory, and starts the default PDF application to display it.

- PDF:[print]

Creates a PDF document in the user's "temp" directory, and sends it to the default printer.

- PDF:[printto]printer

Creates a PDF document in the user's "temp" directory, and sends it to the specified printer.

- PDF:[edit]

Creates a PDF document in the user's "temp" directory, and starts the application defined as the PDF editor.

Note that the "[print]" and "[printto]" options will work with "Windows-only" or "host-based" printers, if the printer's device driver is installed.

FPML – The filePro Markup Language

In order to generate anything beyond "plain text" output, you need to use an XML-like "markup language" called FPML – the "filePro Markup Language" – to control things such as text formatting and image processing.

For many simple reports and forms, a PDF file can be generated by simply changing the print code table to "fpml", and specifying a PDF destination as described above. For more complex print jobs, you may need to include FPML in your output.

There is a new option on dmoedef's F8/Options screen – allow embedded FPML on form – which, if set, allows you to put FPML print codes directly on the form and in data included on the form. Note, however, that this means that data cannot contain the "<" character, as this will be interpreted as the start of an FPML tag. With this option off, it is still possible to embed FPML on the form and in data, by prepending an ESCape character immediately before the "<". For example:

```
xx = chr("27") & "<font color='red'">
```

Unless otherwise noted, everything is case-insensitive. For example, these are equivalent:

```
X="CENTER"
```

```
X="Center"
```

```
x="center"
```

Values can be enclosed in single- or double-quotes:

```
FILE="foo.jpg"
```

```
file='foo.jpg'
```

Unknown tags are silently ignored.

The "fpml" print code table

filePro includes a new "fpml" print code table. This is based on the PCL print code tables already included with filePro. Any output format which uses a filePro-supplied PCL print code table will likely work as-is by simply specifying "fpml" as the printer type, and giving a PDF destination. (Note that PCL codes generated in processing will not work as-is.)

Format

The basic format for FPML tags is:

```
<TAGNAME ATTRIBUTE="VALUE" ATTRIBUTE="VALUE" ... >
```

Note that, unlike XML, there are no "close" tags, nor do the tags end with a slash.

For example, to use the FONT tag to set the font to 18-point bold, you would use:

```
<FONT SIZE="18" BOLD="ON">
```

Remember, case is not significant, and you can use either single or double-quotes, so the following is equivalent:

```
<font size='18' bold='on'>
```

Note that numeric attributes can include decimals. For example:

```
<font size="14.4">
```

Page position

When specifying a page position, the coordinates start at (0,0) as the upper-left corner, within the margins. X increases to the right, and Y increases down the page. Coordinates are specified in "points", which is the typography measure for 1/72 of an inch.

<NUL>

A no-op. This does nothing, and any attribute/values that are given are ignored. This is useful in places where you build the tag at runtime, and may need to have a

placeholder when nothing needs to be done.

<PAGE>

Sets the page size and orientation. Note that, if data has already been sent to the current page, this will implicitly start a new page.

All attributes are optional.

SIZE Specifies the page size

- LETTER
- LEGAL
- A3
- A4
- A5
- B4
- B5
- EXECUTIVE
- 4x6
- 4x8
- 5x7

ORIENT or **ORIENTATION** Specifies the page orientation

- LANDSCAPE
- PORTRAIT
- Alternatively, an explicit height and width can be specified using **HEIGHT** and **WIDTH**.

The default is LETTER, PORTRAIT. This setting will be retained until explicitly changed.

Margins are currently fixed at 1/4 inch. **NOTE:** You must specify BOTH height and width, otherwise, the PDF library interprets a single dimension as an invalid size and sets it to standard before we can access it.

Example:

```
<page size="letter" orientation="landscape">
<page height="396" width="612">
```

Version 5.8.02

There are several **MARGIN** attributes you can set on the <PAGE> tag:

Margins are set with the **MARGIN*** attributes

```
MARGIN|MARGINB|MARGINL|MARGINR="margin"
```

This will set the top/bottom/left/right margins independently.

```
MARGIN="margin"
MARGIN="marginTB,marginLR"
MARGIN="marginT,marginRL,marginB"
MARGIN="marginT,marginR,marginB,marginL"
```

(The order of the margin settings corresponds to the CSS3 order. See http://www.w3schools.com/css/css_margin.asp.)

This will set the margin in groups. Given one number, all margins are set the same. Given two numbers, the vertical (top and bottom) and horizontal (left and right) will be set, each pair equal. Given all four numbers sets all four margins independently.

The default is 1/4 inch (18 points) on each side.

Example: Set the page to letter size, with 1/2 inch margins, except for a 1-inch margin on top. There are several ways to do this. All of these should result in the above margins:

```
<page size="letter" margin="72,36,36,36">
<page size="letter" margin="72,36,36">
```

```
<page size="letter" margin="36" margint="72">
```

Set margins to 1-inch top, 1/2-inch bottom, and 1/4 inch for left/right:

<page size="letter" margin="72,18,36,18">
<page size="letter" margin="72,18,36">

<IMAGE>

Places an image on the document.

All attributes, except for "FILE", are optional.

FILE Specifies the filename. (No default. Filename must be specified.)

ROTATE Rotate the image. (Default: 0.0 – no rotation.)

HEIGHT Specifies the height. (Default: image's actual height.)

WIDTH Specifies the width. (Default: image's actual width.)

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

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

There is no way to explicitly set the Z-order of images. The image will be "above" any text that comes earlier on the page, and "under" any text that comes later on the page.

If no explicit path is given to the filename, filePro will search the following, in this order:

- The main filePro file's default directory.
- PFIMAGEDIR
- PFDLDIR

The filename is case-sensitive if the underlying O/S uses case-sensitive filenames. (ie: Unix/Linux are case-sensitive, while Windows is not.)

Currently, only PNG (*.png) and JPEG (*.jpg or *.jpeg) image files are supported.

The rotation is specified in degrees, counter-clockwise.

Height and width are specified in points.

If you specify "scale", then that direction will scale proportionately, based on the size given the other direction. For example, if the image has a height of 75, then specifying:

```
HEIGHT="150" WIDTH="scale"
```

will double the width as well. Specifying one direction and not the other will leave the other unchanged.

The position on the page (default: the current position) is done with the XY attributes. These can be a number, specified in points, or one of "LEFT"/"CENTER"/"RIGHT" for "X", and

```
"TOP"/"CENTER"/"BOTTOM" for "Y".
```

Note that there is currently no resampling of the image. The image is printed at full resolution, just scaled to the specified size.

Example:

```
<image file="letterhead.png" x="top" y="center">  
<image file="watermark.jpg" x="center" y="center" rotate="45">
```


Sets the font as specified.

All attributes are optional.

Note that font names are case-significant.

NAME The name of the font.

PDF supports 14 "base" fonts:

- "Courier" (plain, bold, italic, bold-italic)
- "Helvetica" (plain, bold, italic, bold-italic)
- "Times" (plain, bold, italic, bold-italic)
- "Symbol" (plain only)
- "ZapfDingbats" (plain only)

Other fonts can be loaded via the PFFONT_* environment variables. (See below.)

SIZE Size of the font, in points.

BOLD Turns bold ON or OFF.

ITALIC Turns italics ON or OFF.

UNDERLINE Turns underlining ON or OFF.

ENCODE Specifies the character set encoding for characters 128-255.

Possible values are:

- StandardEncoding It is the default encoding of PDF
- MacRomanEncoding The standard encoding of Mac OS
- WinAnsiEncoding The standard encoding of Windows
- FontSpecific Use the built-in encoding of a font.
- ISO8859-2 Latin Alphabet No.2
- ISO8859-3 Latin Alphabet No.3
- ISO8859-4 Latin Alphabet No.4
- ISO8859-5 Latin Cyrillic Alphabet

ISO8859-6 Latin Arabic Alphabet
ISO8859-7 Latin Greek Alphabet
ISO8859-8 Latin Hebrew Alphabet
ISO8859-9 Latin Alphabet No. 5
ISO8859-10 Latin Alphabet No. 6
ISO8859-11 Thai, TIS 620-2569 character set
ISO8859-13 Latin Alphabet No. 7
ISO8859-14 Latin Alphabet No. 8
ISO8859-15 Latin Alphabet No. 9
ISO8859-16 Latin Alphabet No. 10
CP1250 Microsoft Windows Codepage 1250 (EE)
CP1251 Microsoft Windows Codepage 1251 (Cyril)
CP1252 Microsoft Windows Codepage 1252 (ANSI)
CP1253 Microsoft Windows Codepage 1253 (Greek)
CP1254 Microsoft Windows Codepage 1254 (Turk)
CP1255 Microsoft Windows Codepage 1255 (Hebr)
CP1256 Microsoft Windows Codepage 1256 (Arab)
CP1257 Microsoft Windows Codepage 1257 (BaltRim)
CP1258 Microsoft Windows Codepage 1258 (Viet)
KOI8-R Russian Net Character Set

COLOR Color

Currently, the only way to specify colors is "#rrggbb", where "rr", "gg", and "bb" are the two-digit hex value for red, green, and blue, respectively. The default is Courier 10, black, with "standard" encoding. If the name, size, color, or encoding is not specified, that attribute remains unchanged.

Note that specifying a font name will clear bold/italic/underline, unless explicitly set to "on" in the same code.

Note that PDF does not support underlined text. Instead, this is accomplished by drawing a line underneath the text.

Example:

```
<font name="Helvetica" size="14" bold="on">
```

<MOVETO>

Move the current text position as specified.

All attributes are optional.

X Specifies the X position.

Y Specifies the Y position.

If a coordinate is not specified, the position on that axis remains unchanged.

Units can be absolute or relative. (If the value starts with "+" or "-", it is relative to the current position.)

Example:

```
<moveto y="-6">raised text<moveto y="+6">
```

<RECT>

Draw a rectangle.

All attributes are optional.

X Specifies the X position of one corner.

Y Specifies the Y position of one corner.

WIDTH The width of the rectangle.

HEIGHT The height of the rectangle.

STROKE The width of the line.

COLOR Color of the line.

Draw a rectangle. If (X,Y) is not specified, the current position is used.

Note that a height of 0 draws a horizontal line, and a width of 0 draws a vertical line.

```
<rect height="72" width="360" stroke="3.5">
```

PFFONT_*

Although PDF only includes the built-in "base14" fonts listed above, additional fonts can be used by defining them with the "PFFONT_*" environment variable. The format is:

```
PFFONT_name=embed:base:bold:italic:bold-italic:
```

(Note: On Windows platforms, the semicolon is used rather than the colon as the separator.)

Where:

"name" is the name by which you will refer to the font. For example, setting the environment variable "PFFONT_Fancy" would create a font named "Fancy", which could be referenced in any tag. (Remember that font names are case-sensitive.) "embed" specifies whether the font is to be embedded in the PDF document. If a font is not embedded, the document may not be displayed properly if the system on which it is viewed does not have that font installed. However, embedding the font makes the PDF document larger, and copyright restrictions may prevent a font from being embedded. The allowed values are "0" (do not embed) and "1" (embed).

"base" specifies the filename which contains the base font description, and is required.

"bold", "italic", and "bold-italic" specify the filenames containing the font description for bold, italic, and bold-italic, respectively. These are all optional, and if left off will cause any reference to the font to revert to the "base" font.

If a path is not specified in the filename, filePro searches for the file in the following directories:

- PFFONDIR
- PFDLDIR
- (Windows only) The Windows system font directory.

Currently, only TrueType (*.ttf) fonts are supported. The filename extension must be included.

Note that the built-in base14 fonts can be overridden using the PFFONT_* variable. For example, setting PFFONT_Courier=1:cour.ttf:courbd.ttf:couri.ttf:courbi.ttf would replace the built-in "Courier" font with the specified TrueType font.

FPML output

If you set the new "allow embedded FPML on form" option in dmoedef, you can include FPML codes directly on the output format.

You can also generate "on-the-fly" FPML by putting the FPML in a field on the form, prefaced with ESC. For example:

```
if:
Then: color = "#000000" ' black
if: value lt "0"
Then: color = "#800000" ' red
if:
Then: xx = chr("27") & "<FONT COLOR=" { color { ""> { value
      { chr("27") & "<FONT COLOR=#000000">"
```

Finally, note that you can put FPML directly on the form, and still fill in values at runtime, by putting something like this on the form:

```
<FONT COLOR="*aa ">*bb <FONT COLOR=#000000>
```

PFPCFCOMPRESSMODE (Ver. 5.8.02)

Sets PDF compression mode

Line & Box drawing (Version 5.8.03)

We now include **fpml3.prt** in the lib directory. This is so we don't overwrite any changes you may have made to fpml.prt

This .prt now has different codes for 5.8.03 and higher that will properly draw lines when sending output to PDF formats.

To use the new fpml3.prt, change the printer type in printer maintenance to fpml3

Set the following in your config file on **Windows**.

```
PFFONDIR=/appl/fp/fonts

PFPDFFONTSIZE=12
PFFONT_COURIER=0;cour.ttf;courbd.ttf;couri.ttf;courbi.ttf;
```

Set the following in your config file on **NIX**.

```
PFFONDIR=/appl/fp/fonts

PFPDFFONTSIZE=12

PFFONT_COURIER=1:UbuntuMono-R.ttf:UbuntuMono-B.ttf:UbuntuMono-RI.ttf:UbuntuMono-BI.ttf.
```

You need to add one of the above to the config file (? and then F6 to edit):

(Fonts Directory based on Default Installation - Your path may be different)

PDF:[Pop] Version 6.0.00

```
popPDF,fpml,PDF:[Pop]/appl/fpoffice/fpPDF;http://172.16.2.25/fpoffice/fpPDF,PDF remote
```

```
printername,fpml,PDF:[Pop]output destination:[webserver output destination],description
```

This new printer type will print a report in PDF format to /appl/fpoffice/fpPDF and then properly format a message to either GI or WEBfilePro to display in the appropriate viewer. WEBfilePro is assumed as the client if it is set.

Version 6.0.01

FPML commands to control the appearance of underlines.

```
<FONT OFFSETU="..."> and <FONT SIZEU="...">
```

Setting the values to zero will restore the default behavior
(default value is 0)

OFFSETU allows the user to change the Y coordinate location of
the underline, this is an offset around the font baseline. A positive
value moves it down, negative moves it up.

SIZEU changes the stroke size of the underline.

Version 6.1.01

Added QRcodeFPML print code.

```
<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 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.

SORT/SELECT Processing

(Sort/Selection processing is also called Scan/Selection. Sometimes they are referred to as -v tables.)

There are many folks out there using filePro for massive chores, doing amazing processing that would boggle the mind of a 20 year data processing veteran and forever subdue the heart of a mainframe aficionado. But one thing I rarely see used well in the filePro world is the power of "scan/selection" processing (also known as "-v tables" to us less erudite). These tables replace the selection set usually used to grab only the desired records for output. With a "-v" table you can customize your output selection and present a more user friendly interface for your operators. Dumping someone on to a pre-built or blank selection screen and telling them to fill in the field number for "Invoice Date" and "eq" or "gt" for relationship, than the desired date... is a little cumbersome and filePro just simply has a better way to do it.

Using a "-v table" to select only the records you want processed is easy if you understand the basics. Here is an explanation/tutorial.

When you make a "-v" table you will use Define Processing and design a new output processing table. Use naming conventions like "labelsel", "vinvoice" or "seldates". In other words let the name of the processing table indicate to you somehow that this is a "selection" or "-v" table. This way you wont confuse these special tables with other actual "output" processing tables that are connected to some kind of output format. "-v" tables stand alone and are not connected to any output format.

On a "-v" table you can ask questions and based on the answers received, select the records the operator wants. The following table demonstrates the idea:

File Name: Invoices

```
Processing: vtable
  If: aa ne ""
  Then: goto sel
ques  If:
  Then: input aa(8,mdy/,g) "What invoice date do you want? "
  If: aa = ""
  Then: goto ques
sel  If: 1 eq aa
  Then: select;end
  Then: end
```

The "-v" table is activated as follows:

```
dreport Invoices -f invoice -v vtable -a -u
```

Lets assume "invoice" in this case is your invoice form. By executing this line you can print invoices for records containing a requested date and skip over other records in the file.

The way a "-v" table works is interesting. Before your output format and processing are run, the "-v" processing table is run against every record in the file. Dreport (the Request Output program in filePro) uses a "-v" table by standing on the first record in the file and running it (the -v table) against one record at a time until the entire file is read. Then, the actual output format and processing are loaded and only the records that were "selected" by the "-v" table are processed. During this "first pass" of the file the "-v" table is testing each record to see if it fits your "selection" criteria. In the case of the example table "...does field number 1 match the date entered by the operator..."

The Big Trick on "-v" Tables

Here is the trick to this table (and virtually ALL "-v" tables). The first record of the file is gotten from the disk, the "-v" table starts on its first line by testing the variable "aa". At this time it is equal to nothing so the "if" line tests false and the processing falls through to the next line. The operator is asked "What Date?" and when he answers with a valid date, the processing falls through to the line labeled "sel". Here, Field 1 (well assume it holds the records invoice date) is tested against the operators chosen date as held by "aa". If field 1 equals "aa" the record is "selected" and the process "ends". This, (hitting the "end" statement) causes the dreport program to get the next record from the disk. This time and for EVERY OTHER record the first "if" line will always test true since "aa" is a global variable. (This means it holds its value from record to record). Because the "if" line tests true the processing will immediately "goto" the line labeled "sel" where the all-important "selection" test is done again. If this records field 1 matches the date held in "aa" it will be "selected", if field 1 is not equal to "aa" the process falls through to the "end" statement standing alone on line 5 and does not perform the "select;end" on line 4. Dreport moves on to the next record and the same thing happens over and over again until every record has been tested. When this first pass is completed only those records "selected" are formatted by the "invoice" format/processing.

You can do more complicated selections like the following with a "-v" table. It can get as sophisticated as your imagination. The processing shown here should be self-explanatory. See if you can figure out what it will do.

File Name: Invoices

```
Processing: vtable
  If: aa ne ""
  Then: goto sel
ques  If:
  Then: input aa(8,mdy/,g) "What invoice date do you want? "
  If: aa = ""
  Then: goto ques
sel  If:
  Then: input bb(6,0,g) "Enter customer code? (RETURN=ALL) "
  If: 1 eq aa
  Then: goto chkcust
  If:
  Then: end
chkcust  If: bb = ""
  Then: select;end
  If: 2 = bb
  Then: select;end
  If:
  Then: end
```

This table will work just like the first "-v" table asking for the desired date and then it will ask for the desired customer code. If no customer code has been entered then only the date match matters to the selection, but if "bb" has been filled with something, both criteria have to be true for a record to be "selected". (Assume field 2 is the customer code stored on each record) You could choose to print invoices for client "USA2" on "02/05/89" and only those would be printed. This example is used to demonstrate the principle of asking lots of questions on a "-v" table and then selecting records based on the response, it isnt meant to be very useful as it exits for your system designs.

These tables will work with ranges of dates, zip codes, client codes etc. In fact any criteria that can be tested on a selection table can be tested this way, with the added advantage that other processing can be done during the "-v" pass. So that you have the ability to do things like get the percentage of one records value to an entire file, or many more wonderful things that are difficult or impossible without this powerful filePro feature.

How It Works

Output makes two passes of the file when generating output. On the first pass, it selects records and sorts the selected records. (Messages appear: "Reading Keys", "Writing Keys", and "Sorting Keys".) Sort/selection processing is done during the first pass for each record

On the second pass, processing works the selected records in sorted order. (Message appears: "Generating Output".) Automatic and output processing are done during the second pass.

Advantages

Sort/selection processing make the sort and selection operations more friendly. You can put in personalized questions rather than standard screens.

The user can specify additional details to the selection criteria at runtime.

The user can define the sort criteria at runtime and override or add to the standard sort.

The user can define the sort or selection based on criteria not in the current record (as in selection sets). Records can be chosen based on lookups, dummy variables, or expressions.

Percentages and other mathematical functions that require two passes of the file can also be performed.

However, instead of selecting Request Output you must generate the output from a user menu or from the command line. The sort/selection table is specified using the -v flag.

The syntax is as follows:

```
dreport filename -f formatname -v sortselectname -a
```

Two special commands are used on Sort/Selection processes, SORT and SELECT. These two commands only function on a -v processing table.

SORTn

Where n is the expression that designates the sort field desired.

Overrides the sort as defined in the output format (but, not subtotal or total breaks).

You can define up to eight sort fields.

SORT is used on sort/selection tables only.

Example

```
Then: sortn=m
```

where "n" is the level (a number from 1 to 8), and "m" is the field to sort on (real, dummy, or lookup)

SELECT

SELECT command, a record is selected only if it encounters the **SELECT** command. If the **SELECT** statement does not have a true if condition or if it is bypassed (jumped around with other redirection) that record will not be included with those sent through to the output pass.

SELECT is used only on sort/selection tables.

Example

```
If: aa eq ""
Then: input aa(2,.0,g) "Enter Doctor Code To Print:"
If: l3 = aa
Then: select
If:
Then: end
```

Note that a global dummy field is used since you want the question to only be asked at the very beginning of the run and not on every successive record.

Passing Data From a Sort/Selection Table to the Output Table

It is possible to pass dummy fields from the sort/selection table to the output table via the automatic table.

Example:

Suppose that on your sort/selection table you prompt the user for beginning and ending dates for invoices to be printed on a report. You use input questions with dummy fields "bd" and "ed". Be sure that on your automatic table (or -y table) that you bring the variables into existence as "global" variables. This will ensure that the output format (and its processing table) will see the value of these dummies. Only variables that have been defined as global on the automatic table can pass values from the sort/selection table through to output.

Example (somewhere on the automatic table)

```
Then: bd(8,mdy/,g); ed(8,mdy/,g)
```

Calculating Percentages

1. Add global dummy field "rt" and local dummy field "ip" to your automatic processing table.

```
Then: rt(12,.2,g); ip(10,.2)
```

2. Add a totaling function to your sort/selection table.

```
Then: rt=tot(14)
```

3. Add a percentage formula to your output table.

```
Then: ip=(14/rt)*"100"; print
```

4. Execute the process from a user menu or command-line statement that includes the "-v" flag and the sort/selection processing table and you will see the "individual percentage" each record is out of the entire "report total". If you were running a report without sort/selection processing first, you would be stopping on each record to do your calculation without knowing what the total value of the whole file is. The sort/selection pass of the file is all important for such things.

Lockfile

The lockfile format has been changed with release 5.6. Although this does not prevent files from being accessed by earlier releases, you should avoid accessing the same files at the same time with release 5.6 and earlier releases since this will present the following error.

*** A filePro Error Has Occurred ***

On File: W:\ccp/filepro/l_patient_market/lockfile

Old Version of filePro Function Running On This File

File not available. Somebody else is modifying the file; try again later.

You should not get this error unless someone is still in the file with 5.0 or earlier or a program crashed thus leaving the lockfile marked "in use".

Deleting the lockfile will allow you to run release 5.6 or earlier releases without the above message as long as you do not attempt to use 5.6 and earlier releases at the same time.

Caution: Don't delete the lockfile unless you know that no one is using the file.

See [FilePro Directory - Option ?](#) for how to unlock a file.

Alternate Automatic Processing

Defining processing tables now allows you to tag a specific auto processing to use with the process for tokenizing and syntax checking. This alternate auto process is displayed at the bottom of the window. *clerk and *report will use this alternate process if there is no -y flag on command line.

*cabe, *clerk, & *report
uses -y process if set
uses alternate auto process if set
uses automatic process if no -y or alternate auto process set

Another Trigger - @key

There is a relatively simple command called @key, that lets you do things with processing while you are sitting at the "Enter Selection" prompt on a filePro screen. This feature lets you execute processing when the user presses a specified key. These keys are similar to the keys you see at the bottom of the screen like H=Hardcopy, F=Form, U=Update and so forth.

For demonstration purposes, Let's say you have a screen that is so full of information there is no room to put the creation date and created by system maintained fields anywhere on it. You could use the following @key to display them. Or maybe you don't like seeing this information ALL the time. We will use @keyC for this example but just about any character key on the keyboard would do:

```
      If:
      Then: end
@keyC  If:
      Then: show "@Created On"<@cd<"Created By"<@cb
      If:
      Then: end
```

When the operator presses the C key, this @key processing will display the creation date and created by fields filePro stores on each record.

Note 1) Any processing above the @key should not fall through accidentally to the @key processing. This example shows the "end" command just above this @key processing. For now, put all your @key processing at the very end of your main input table, and before each @key you should have an "end" statement.

Note 2) @key processing must have a definite ending line, a place where the processing stops and the @key is over. For now, until you learn more about them, be sure there is an "end" statement to stop the @key processing when it should stop.

You don't have a watch, you are deeply involved with your filePro records but are too lazy to go and look at a clock. How about showing the time when you hit @keyT, enter the following:

```
      If:
      Then: end
@keyT  If:
      Then: show "@The time is"<@tm
      If:
      Then: end
```

You can use an @key to do fast custom updating of flags in filePro mailing lists. Let's say you have a file that you want someone to use as a "prospect calling list". You know all the people in the file and want to select a group of them for the person to call. The selection criteria is totally subjective so you want to look at each record and make up your mind whether to mark it for calling or not. Do the following:

Go into Define Files on the prospect file, add a 1 character field to the file, give it an edit type of "allup". Let's use field 22 as an example.

Go into Define Screens and put this field on your screen somewhere.

Go into Define Processing tables and add this little table to the end of your input table:

```
      If:
      Then: end
@keyY  If: 22 eq ""
      Then: 22 = "Y";display;end
      If: 22 eq "Y"
      Then: 22="";display;end
```

This now becomes a toggling switch for you to use in marking the records. If you want to mark the record, press Y. (Remember your cursor must be blinking at the Enter Selection prompt) If you want to deselect the record, press the Y again and the field is cleared. You don't have to go into Update mode cursor down to the right field enter the right thing and then press ESC ESC. It's a real time saver, you can arrow down through the records marking and unmarking as you go.

You could then build another @key like @keyC which would do the same thing but with letter C. The person doing the calling could mark everyone he called with a C so they wouldn't be called again by accident. When the file is empty of Y's and full of C's start all over again with other letters or different classifications.

You can have an @key zoom you into another file temporarily with the following:

```
      If:
      Then: end
@keyZ  If:
      Then: system "/appl/fp/rcclerk otherfile -s1"
      If:
      Then: end
```

This would put you into "otherfile" on screen 1. You can use this file as long as you want, when you exit this file, you are back on the record in the first file where you pressed the @keyZ.

NOTE: The @keyX letter like this X do not have to be upper case, I do that just as a matter of style. It makes them easier to read than something like @keyy, which would work just as well as @keyY.

You can also use the punctuation symbols on the keyboard, as in @key+ and @key- or @key?

Trigger Processing (INPUT)

There are several @trigger types of processing available that are coded on the INPUT processing table. The major types are @when leaving/entering, @Key, @entsel, @menu, @whp, @wuk. There are others.

@When Leaving/Entering Field Processing

When-field processing is done when the cursor enters or exits a field. It is used when the processing should be done on a field-by-field basis.

When-field processing is performed immediately as the user is entering data and moving from field to field, rather than when the user has finished entering data and finally presses ESC ESC.

Common Uses

Displaying messages when the user enters or exits a field.

Preventing the user from leaving a field blank or with unwanted data in it.

Filling in looked up data immediately.

Automatic Processing - Compiling

When compiling processing tables that rely on dummy fields defined in an automatic processing table having a name other than "automatic" (UNIX) or "auto" (DOS, Windows), you need to specify the automatic processing table name with the "-y" flag when defining the processing table.

Example:

If you are using dummy field "aa" to keep track of a subtotal in a report1 and you are going to run rreport with a different automatic processing table named "autotot" (which defines dummy variable "aa"), then you must compile the report processing table as follows.

```
rcabe filename report1 -y autotot
```

Or, put another way:

If you are using "aa" to subtotal, and are going to run rreport with an automatic processing that does not define "aa", then you must also compile with an automatic table that doesn't define "aa". (Or vice versa.)

Of course, your best bet is to simply compile with the same automatic table.

See PFZEROLENWARN to turn off assigning to a zero length field warning

Browse Lookup Example

Browse lookups are the saving grace of filePro. They take much of the hard coding out of viewing and manipulating your data. Moreover, they allow relatively new programmers the ability to write some very sophisticated programs.

NOTE: Keep in mind that files with creation passwords prevents fields from being displayed in a browse lookup unless the field is included on the screen. This provides a way of protecting data from unauthorized viewing.

Here is a fairly generic application written solely based on a browse lookup. It stands users in one file and allows them to Add, Modify or Delete records in another file, while keeping some simple totals in the current file.

File Name: npwarestand

Number	Field Heading	Len	Type
1	Spec#	5	.0
2	Seq#	3	.0
3	cartons	5	,0
4	R	0	
5	pounds	8	,0
6	descr1	28	
7	CustCode	6	
8	roll/case weight	5	.1

Key segment record length: 60

There is no data segment.

File Name: npwarestand

Screen: 1

```

+- Totals for SPEC# !1 -----+
|                               |
|           Cartons   Pounds   |
|           !ta      !tc      |
| !6                               |
| !7                               |
+-----+
    
```

File Name: npwarestand

Screen: add

```

+-----+
|                               |
|           SEC#: !oa   Shift: !ob   |
+-----+
|                               |
| Date      In/Out  P#      Cartons  Skids  Pounds  Comment  |
| *ob       *oh     *od     *oe     *of     !og     *oi     |
|                               |
+-----+
    
```

Cursor Path:

```

12
TAB
2
1
3
TAB
10
4
5
8
    
```

File Name: npwarestand

Screen: goto

```

+-----+
| Goto which S# or P# |
|                               |
| SPEC#: *s           |
|                               |
| P#: *p              |
|                               |
+-----+
    
```

```

|
|
| Press ESC ESC to Go |
|
| Press DEL to cancel |
+-----+

```

File Name: npwarestand Printed: May 6 1997 14:04 Page 1
Output Format: list-noval (fP 4), Dated: Mar 4 12:04:11 1997

```

      10      20      30      40      50      60      70      80
      : | : | : | : | : | : | : | : |
      H E A D I N G / T I T L E L I N E S
Nexus Plastics, Inc. Date: *@td
Warehouse #2 Inventory Page:<@pn

```

SPEC#	Cust	Description	Cartons	Pounds
D A T A L I N E S				
*1	*7	*6	*3	*5
T O T A L L I N E S				
G R A N D T O T A L				

```

                          Cartons  Pounds
                          Totals:   =3    =5
E N D O F F O R M

```

File Name: npwarestand Printed: May 6 1997 14:04 Page 2
Output Format: list-noval (fP 4), Dated: Mar 4 12:04:11 1997

```

Sort Field: 1
Length: 5
Descending:
Subtotal Field:
Align form: N
Remove blank lines: N
Grand Total new page: Y
Printer:
Initial print code:
Final print code:

```

File Name: npwarestand
Processing Table: list-noval

```

If: 3 eq "" and 5 eq ""
Then: end
Then: print
Then: end

```

File Name: npwarestand
Browse Format: default

Spec#	Cust	Description	Cartons	Pounds
*1	*7	*6	*3	*5

Number	Field Heading
1	Spec#
7	CustCode
6	descr1
3	cartons
5	pounds

File Name: npwaredet

Number	Field Heading	Len	Type
1	spec#	5	.0
2	date	8	mdy/
3	seq#	3	.0
4	P#	5	.0
5	cartons	5	.0
6	r	0	

7	pounds	7	.0
8	in/out	3	
9	comment	20	
10	initials	3	

File Name: npwarestand
Processing Table: automatic

```
Then: ff(14,,g)="npwaredet"
  If: fl eq ""
Then: display "blank";end
Then: display 1 ; end
```

File Name: npwarestand
Processing Table: input

```
Then: 'run with "rreport npwarestand -sblank -lx -d"
Then: gosub realltot
Then: end
@menu  If:
Then:
  If: bk eq "set"
Then: bk="";end
  If: in ne ""
Then: exit
Then: input popup in(3,,g) "What are your initials? "
  If: @sk eq "BRKY"
Then: exit
  If: in eq ""
Then: goto @menu
Then: pushkey "11[ENTR]G";end
@keyG  If:
Then:
Then: cls("20")
Then: s(5,allup,g)="" ; p(5,allup,g)=""
Then: popup update -, "goto"
  If: @sk eq "SAVE"
Then: clearp; goto dogo
  If: @sk eq "BRKY"
Then: clearp; end
  If: 'stopgap not needed
Then: end
dogo  If:
Then:
Then: fl="1"
  If: s eq "" and p eq ""
Then: end
  If: s ne ""
Then: goto dos
dop  If:
Then: lookup npio k=p i=a -nx
  If: not npio
Then: lookup npio = npioarch k=p i=a -nx
  If: not npio
Then: show "@P# not on file, try again.";end
Then: s=npio(3)
dos  If:
Then: lookup npis k=s i=a -nx
  If: not npis
Then: show "@Sorry, that SPEC# is not on file!"; show "";end
Then: rc(5,.1,g)=npis(44)
Then: da(28,,g)=npis(22) ; db(6,,g)=npis(2)
Then: kg=s
Then: lookup tst=npwarestand k=kg i=a -nx
  If: not tst
Then: goto addnew
Then: lookup - k=kg i=a -nx
Then: end
addnew If:
Then: lookup - r=free -ep
Then: l=s ; 8=rc ; 6=da ; 7=db ; write ; display ; pushkey "[SAVE]";end
@keyD  If:
Then: input popup q " \r Are you SURE you want to DELETE EVERY ENTRY on t
  his Date/Shift? (y/n) \r "
  If: q ne "Y"
Then: end
Then: gosub killall
Then: gosub clrfls
Then: end
killall If:
Then: kd=1
Then: lookup eat=(ff) k=kd i=a -nx
```

```

moreat  If: not eat
        Then: return
        If: eat(1) ne 1
        Then: return
        Then: delete eat;getnext eat;goto moreat
clrfls  If:
        Then:
        Then: ta(6,,0)=""; tc(8,,0)=" "
        Then: return
@keyU   If:
        Then:
        If: @rn eq "1"
        Then: show "@Goto a Spec# first... then press U again.";show "";end
        getdet  If: '*getdet
        Then:
        Then: gosub tstnew
        If: n eq "1"
        Then: goto contold
        Then: input popup q "No entries yet, would you like to add some? (y/n) "
        If: q ne "Y"
        Then: end
        Then: cls("20");gosub clrvars;gosub adddet;goto @keyU
        Then: '
contold If:
        Then: gosub gettots
        Then: cls("20")
        Then: display
        Then: gosub prompts
        Then: ky=1
brwl    If: '*brwl
        Then:
        Then: za="(brw=13,7,1 xkey=AMDx show=pkeep prc=procl mlen=5 fill=asc)"
        Then: zb="[Date P# In/Out Cartons Skids Pounds Comment
        Ini]"
        Then: zc= "*2 *4 *8 *5 *6 *7 *9
        *10"
        Then: lookup bom=(ff) k=ky i=a -ngm b=(za&zb&zc)
        If: @sk="SAVE"
        Then: cls("20");clearb;goto cleanup
        If: @sk="BRKY"
        Then: cls("20");clearb;goto cleanup
        If: @bk="X"
        Then: cls("20");clearb;gosub realtot;clearb;end
        If: @bk="D"
        Then: cls("20");gosub deldet;gosub gettots
        If: @bk="D" and n eq ""
        Then: cls("20");clearb;df="";goto getdet
        If: @bk="D" and n eq "1"
        Then: n="";gosub gettots;cls("20");gosub prompts;goto brwl
        If: @bk="A"
        Then: cls("20");gosub clrvars;gosub adddet;gosub gettots;cls("20");gosub
        prompts;goto brwl
        If: @bk="M"
        Then: cls("20");gosub clrvars;gosub ldvars;gosub moddet;gosub gettots;cls
        ("20");gosub prompts;goto brwl
        Then: goto brwl
procl   If:
        Then:
        Then: end
deldet  If: '*deldet
        Then:
        Then: cls("20")
        Then: beep
        Then: input popup q(1,yesno) " \r Are you SURE you want to delete this
        entry? (y/n) \r "
        If: q ne "Y"
        Then: return
        Then: delete bom
        Then: gosub realtot
        Then: gosub tstnew
        Then: return
adddet  If: '*adddet
        Then:
        Then: lookup bom=(ff) r=free -ep
        Then: bom(1)=1
moddet  If:
        Then: oa=1
        Then: cls("20")
        Then: gosub prompt2
        Then: BREAK OFF
        Then: popup update -, "add",ob
        If: ob&od&oe&of eq ""
        Then: write bom;delete bom;clearp;return
        Then: BREAK ON
        Then: bom(2)=ob
        Then: bom(4)=od ; bom(5)=oe ; bom(6)=of ; bom(7)=og
        If: oh eq "o"
        Then: bom(8)="out"
        If: oh eq "i"
        Then: bom(8)="in"
        Then: bom(9)=oi ; bom(10)=in
        If: bom(3) eq ""

```

```

Then: 2=2+"1" ; bom(3)=2
Then: write
Then: ky=l&bom(3)
Then: clearp ; return
@entsel If:
Then:
If: @sk eq "BRKY"
Then: exit
Then: gosub gettots
Then: gosub legend
Then: end
legend If: '*legend
Then:
Then: cls("20")
Then: show("21","1") "-----"
-----"
Then: show("22","1") "\r U \rupdate/View entries"
Then: show("24","1") "\r G \rto another SPEC#"
Then: show("22","53") "\r D \relete this entire SPEC#"
Then: show("24","67") "\r X \r=Exit"
Then: show("24","37") "\r L \r=List records (browse)"
Then: show("22","37") "Select >"
Then: return
gettots If:
Then:
Then: gosub clrfls
Then: kt=1
Then: lookup det=(ff) k=kt i=a -nx
mordet If: not det
Then: goto fintots
If: det(1) ne 1
Then: goto fintots
If: det(8) eq "in"
Then: goto doin
doout If:
Then: ta=ta-det(5)
Then: tc=tc-det(7)
Then: getnext det ; goto mordet
doin If:
Then: ta=ta+det(5)
Then: tc=tc+det(7)
Then: getnext det;goto mordet
fintots If:
Then:
If: ta eq "0"
Then: ta=""
If: tc eq "0"
Then: tc=""
Then: display;return
realtot If:
Then: 3=ta;5=tc;write
Then: display;return
prompts If: '*prompts
Then:
Then: cls("20")
Then: cb="Use UP and DOWN arrows or NxtPg and PrvPg to scroll entries."
Then: show ("22","40"-dlen(cb)/"2") cb
Then: cc="\r A \r to Add an entry, \r M \r to Modify an entry, \r D \r to
Delete an entry. \r X \r=Exit"
Then: show ("24","40"-dlen(cc)/"2") cc
Then: return
prompt2 If:
Then: cd="Press \r ESC ESC \r to Save, \r DEL \r to cancel."
Then: show ("24","40"-dlen(cc)/"2") cd
Then: return
clrvars If: '*clrvars
Then:
Then: oa(5,.0)="" ; ob(8,mdy/)=""
Then: oc(3)="" ; od(5,.0)="" ; oe(4,.0)="" ; of(2,.0)="" ; og(7,.1)=""
Then: return
ldvars If: '*ldvars
Then:
Then: ob=bom(2) ; od=bom(4) ; oe=bom(5) ; of=bom(6) ; og=bom(7)
Then: oh=bom(8) ; oi=bom(9)
Then: return
tstnew If:
Then: kn=1
Then: lookup new=(ff) k=kn i=a -nx
If: not new
Then: n="" ; return
Then: n="1" ; return
@whelp If:
Then: help "main";end
@wlfs If: s ne ""
Then: p = "" ; display ; end
Then: display;end
@wlfp If: p ne ""
Then: s = "" ; display ; end
Then: display;end
cleanup If:
Then:
Then: end
@wlfob If: ob eq ""

```

```
Then: screen ,ob
     If: ob lt "01/01/97"
Then: show "Date out of range";screen ,d
Then: show "";end
@wlfoe If:
Then:
Then: og=oe*8
Then: display;end
@wlfoh If: oh ne "i" and oh ne "o"
Then: screen ,oh
Then: end
@keyL  If:
Then: video off
Then: bk(3,,g)="set"
     If: l eq "0"
Then: pushkey "x4a"{1{"[ENTR]b" ; end
Then: pushkey "x4a"{1{"b" ; end
```

COMMAND GROUPS

Commands can be grouped into sub-categories as follows;

User Interaction

BEEP
BREAK
CLS
HELP
INPUT
SHOW
DISPLAY
MSGBOX
ERRORBOX

Input Control (Inquire, Update, Add only)

ESCAPE
RESTART
SCREEN
SKIP

General Processing Control

END
EXIT
GOTO
GOSUB
MENU
RETURN

File Control and Interaction

CLOSE
COPY
COPYIN
DELETE
LOOKUP
GETNEXT
GETPREV
SYSTEM
WRITE
PUSHKEY

Dummy Fields in Processing Tables

Dummy fields hold data temporarily (in memory) for processing operations, display, and printing. Dummy fields (variables) can be generally categorized as either "Short" or "Long" dummy fields. The "Short" dummy field names are limited to 1 or 2 letters from A to Z and AA to ZZ. "Long" dummy field names can have a length up to 64 characters thus offering an advantage in documenting the variable usage within the name. Either category of dummy field can be used in filePro processing tables to hold any data type, the results of lookups, calculations, or data you want to temporarily store from record to record. Only "Short" dummy field names can be used for presenting/updating data on data-entry screens and printing data on reports.

Dummy fields can be used for almost every operation for which real fields can be used. Remember that these fields are "memory variables" and are not saved to disk unless the results are written to real fields.

Dummy fields are always cleared (set to null) at the beginning of processing for each record (just before the automatic table is run) unless a GLOBAL attribute is assigned.

A variable can be set up to 32127 in character length.

Defining the Short Dummy Fields

1. You bring a "Short" dummy field into existence by typing the field, with its attributes (length, edit type, global attribute) on the "Then" line of a processing table. The exact syntax is described below.

Aa(L,T,G)	Define the dummy field
Aa	name of the dummy field, 1 or 2 letters and case is not significant. Codes aa - zz can be used. Do not use a letter with a number since these are reserved for associated fields.
L	length of the dummy field
T	edit type of the dummy field (any available edit)
G	Optional Global attribute. Default is non-global.

If edit type is left out, the dummy field assumes the default edit type (*), therefore:

`aa(1)` and `aa(1,*)` are equivalent.

If the global attribute is used, the same data is retained in the dummy field between records until the value is cleared or overwritten by processing.

`aa(1,g)` defines aa as a global dummy field without an edit type.

`aa(1,YESNO,g)` defines a global dummy field with the edit type YESNO (only allows a "Y" or "N" to be entered).

2. A dummy field can be defined on a line by itself or when first used. It doesn't matter where you define the dummy field on the processing table.

```
Then: aa(10,.0); bb(12,UFLOW)
```

```
Then: x(12,.2)="298.33"
```

3. Dummy fields only have to be defined once on a table. For example, if you do `aa(8,allup)` anywhere on the table, you can refer to that field anywhere else as `aa` (do not need the definition of length and edit).

4. If you define a dummy field on the automatic processing table, it can be used on output, input, and CALLED processing tables without being redefined.

5. Don't redefine dummy fields. Don't redefine fields that you've defined on the automatic table.

6. Undefined dummy fields will take on the attributes (length and edit) of fields they are assigned to, for example:

`aa=12` will set aa equal to the contents of field 12 and give it that same length and edit type.

7. Undefined dummy fields set equal to a numeric calculation will be given a default length of 255, as in.

8. Undefined dummy fields set equal to a string will be assigned a length of 128.

How A Processing Table Works

It is important to understand the "operation" and flow of the various processing tables.

Simply put, the operation of a processing table is a matter of "stop and flow". When you update (or create) a record, the cursor is put on the first field available on the screen (or the first one specified by the cursor path) and control is handed to the user. When the operator is finished entering data on this screen an interactive program called the input processing table is run. The most basic way this program is started is by the action of the operator pressing the SAVE command (Unix=ESCAPE ESCAPE, DOS=ESCAPE). At this point the "program pointer" starts from line number 1 of the input table and progresses as directed from line to line, executing each "then" line for which the "if" line is true. If the "if" line is false the program "falls through" the "then" line WITHOUT executing it, to the next "if" line and tests it, etc. etc. A "then" line can direct the program to "GOTO" a different "if" line instead of the one immediately following. The program continues executing each line as directed until certain commands which stop the flow are encountered. Commands that stop the flow are "then" lines which hand control back to the operator. The two most basic commands that do this are the SCREEN command and the INPUT command.

The SCREEN command switches the user to another screen so he can see and enter data in other fields. When a SCREEN command is encountered, the user is placed on that screen and control is once again handed back to him. The IMPORTANT thing to learn is that the program counter is sitting at whatever line the SCREEN command was on in the processing table. The next time the user presses SAVE (ESCAPE/ESCAPE), the program (the input table) will start running with the command immediately after the SCREEN command. It DOES NOT start from line number 1 as it did before.

There are several commands that can hand control back to the user. It is this interaction of user and program that eventually brings you to an END statement or the literal end of the input table. If you are in Add Records mode, you will be brought to the next available record and the cursor is deposited on original screen you saw when you first entered Update mode. The input table program pointer is sitting at line 1 again waiting for the user to press ESCAPE/ESCAPE on this new record. If you are not in Add Records mode, the cursor is brought to the Enter Selection prompt.

To demonstrate this "stop and flow", consider the following input table. It demonstrates some aspects of program flow control. (We will assume there is no automatic processing on this file.)

File: test

Processing: Input

```
Then: 15=12+13+14
Then: screen 2,9
If:
Then: 3=15*2
If: 3 gt "100"
Then: screen "credit";goto quest
Then: 4="Y";end
quest If:
Then: input q(1,yesno) "Exceeds limit! Okay?"
If: q eq ""
Then: goto quest
If: q eq "Y"
Then: 4="Y";end
Then: 4="N";end
```

An English translation of this table is as follows. When the operator presses ESCAPE ESCAPE the first time, the program pointer starts at line 1. Because there is no "if" condition the process fills field 15 with the value of 12 plus 13 plus 14 (presumably just entered by the operator on the screen he was on).

Because there is no "if" condition on line 2, the process switches screens to screen 2 and hands control back to the operator by putting his cursor in field 9 on that screen. The program pointer stops here and must wait for the operator to enter or view data on screen 2. The program is stopped and waiting. When the operator is finished on this screen and presses ESCAPE ESCAPE, the program picks up at the point where it left off, and executes line 3 (because once again there is no "if" condition). Field 3 is filled with the value of field 15 times the value of field 2.

The process then moves to line 4. Here it tests the "if" condition. If the value of field 3 is greater than "100", the user is sent to screen "credit" and the program pointer stops right there. If it proves false (3 is equal to or less than "100"), the program falls through the "then" to line 5. Because there is no "if" condition on line 5 sets field 4 equal to a "Y", and then ends. If the test of line 4's "if" condition did prove true, the program will be waiting for the operator to press ESCAPE ESCAPE to record the "credit" screen. At this point the process picks up with the command immediately following the SCREEN command which means it goes to the line labeled "quest".

Here the INPUT command both alerts the operator to something and waits for a response. The program pointer is waiting once again at line 6. This time, however, the operator must supply an answer to the INPUT statement. He cannot just press ESCAPE ESCAPE it will have no effect. His answer is placed in the dummy field (also called a variable) "q". The dummy field "q" has an edit type of "yesno", which means only a "Y", an "N", or a RETURN is accepted by filePro as legitimate answers. Anything else will generate an Edit Failed error. The process tests "q" for each of these possibilities and does something different for each answer.

As soon as the operator answers the question, the program pointer moves to line 7 and tests to see if RETURN was pressed. (If this is true "q" will be filled with nothing, represented as "". This is referred to by programmers as "null".) If "q" is null, then the process sends the operator right back to the line labeled "quest", because we need either a Y or N answer. We are in control, not him! The program displays the INPUT question again and waits for a better answer this time. If he keeps answering with RETURN, he keeps getting the INPUT question. When he finally gives in and answers with either a "Y" or an "N", the "if" condition on line 7 tests false, because "q" is now not null and the process falls to line 8. If the operator answered "Y", field 4 is filled with a "Y" and the program ends. If the operator answered "N", field 4 is filled with an "N" and the process ends.

NOTICE that there is no "if" condition necessary for "q" being equal to "N". This is so, because if "q" is not equal to "" and it is not equal to "Y" the only other value the "yesno" edit would have allowed the operator to enter was "N". Therefore, if the process hits that line, we know "q" must be equal to "N". We have provided for the other two possibilities before this line is reached. (It would do no harm to have line 9's "if" condition read q eq "N", but you might as well get used to the usual programmer obsession of not writing any code that isn't absolutely necessary.)

By directing users through your screens with logic that forces them to do only what you allow; and by using edits which force them to give only answers that you have completely covered in your processing, you can start to write simple filePro tables that will do what you want done and not what the computer wants done.

Keyboard Input

The **INPUT** command is how you obtain input from a user. It allows you to ask the user a question and put his response into a dummy variable so you can test that answer (variable) against various criteria and thereby take some specified action.

Syntax:

```
input dv "message/question."
```

The dummy variable used does not matter. It will be cleared of whatever value it had and filled with whatever the user types to answer the question/message.

Often, it is valuable to limit the response from the user to only certain values. You can do this most easily by giving the dummy variable a specific edit type. You can do this right on the INPUT line, as in:

```
input qq(1, yesno) "Does everything look okay?"
```

This assigns a length of 1 to the variable qq and it assigns the edit type of yesno. This is a factory provided edit type that only allows a Y or an N as an answer (or the user may press ENTER). If the user presses a "B" or a "J" or any other key other than a Y or an N (or ENTER) the program will just sit there and wait for him to get it right. That is why it is valuable to put the suggested answers inside your INPUT message, like this:

```
input qq(1, yesno) "Does everything look okay? (y/n) "
```

In this way, you are asking the question and providing the allowable answers all in one message.

The INPUT command has just about the same set of variations as the SHOW command. You can use:

```
INPUT(r,c) dv message
```

Where r and c are the row and column at which to show the input message, and dv is the dummy variable into which to which the answer goes. There is also:

```
INPUT POPUP(r,c) dv "message"
```

And this is just like the show popup. The input message is neatly boxed (and centered if you leave row and column blank).

There is one more INPUT command, INPUTPW (and INPUTPW POPUP). These are the very same as INPUT except that they hide the users response by printing #s on the screen instead of their actual response. This is so you can ask for passwords and secret things and no one standing behind the user will see the secret. However, the correct response is still deposited in the specified input dummy variable.

Making Use of the Response

Once you have queried the user with any of the INPUT variations, you will end up with his response as the contents of the dummy variable you provided. If you issued the statement as:

```
input qq(1, yesno) "Do you want a hardcopy? (y/n) "
```

The program will now have the users response store in qq. You may do something like:

```
if qq ne "Y"  
  Then: end  
if  
  Then: hardcopy ; end
```

This says in English: If the users answer was not equal to a "Y" then he must not want a hardcopy, just end the processing here. Otherwise, if the user had answered a "Y", the first test would be false and the then line would not be done. Instead, the program falls through to the next "if" line and tests it. Since there is nothing to test, the program assumes that it should execute the "then" line. This is true anytime there is no "if" test, the "then" line will be executed.

You can use the same INPUT variable over and over again. In other words, you do not need to assign a different INPUT variable each time you ask a "yesno" question. If you have used qq once before, you can use it again. It gets cleared just before it is put on the screen so you will be testing the newly offered input from the user each time. If you need a different type of input, say a date, then you would need to establish a date type INPUT variable (which you could use for every date question from then on in that processing table.)

The LOOKUP Command

Definition of a Lookup

To set up a lookup operation, there are always two, usually three activities:

Writing the LOOKUP statement, you specify:

1. The file to be connected and the record(s) from which data will be taken
2. Handle failure, tell the program what to do if the lookup fails.
3. Using the looked-up data, usually field assignments you specify the fields from which data will be read and the fields to which the data will be written.

Terms:

Current file , the file in which you're writing the lookup

Lookup file , the file you are looking into.

MSGBOX/ERRORBOX

The **MSGBOX** and **ERRORBOX** commands are essentially the same thing. You would only use one over the other for the sake of giving users a consistent look and feel to your program. The **MSGBOX** and **ERRORBOX** can be assigned different color schemes. By this color scheme, the user knows immediately that he is seeing an **ERROR** or not. Each command can take input from the user and allow only a special group of keys as acceptable responses. The syntax for these boxes is:

```
MSGBOX (r,c) message,prompt,keylist
```

This means you can not only give a message to the user, but prompt him with special instructions (like what keys he may press in response, and you can disallow any other keypresses. Something like this:

```
errorbox("5","5") "You forgot the payment. Enter it now? ","Y=yes,L=later,C=Cancel ",YLC
```

The message appears inside the box. The prompt on the bottom edge of the box, and the allowable responses go into a space provided on the bottom line by filePro.

You can make **MSGBOX** and **ERRORBOX** very elaborate with `\r` codes for reverse video (just as in **SHOW** and **INPUT** statements). You may also use `\n` to generate a "newline" within the boxed message. All the **SHOW** codes work inside **MSGBOX** and **ERRORBOX**.

Output Processing Tables

Output processing is performed during Request Output on a selected group of records.

When you Request Output the program performs 2 passes through the file. First, it selects and sorts the records. Then it performs the output processing on each record.

It may or may not include the printing of output. If there is no printout, it is called a "Processing-only" operation.

Common Uses

Formatting data for special display on the output device.

Accumulating special totals on reports.

Posting data to summary files

Uses of Processing-Only Operations

Mass recalculation, performing a math operation on a field throughout a file.

Global update, performing a textual change throughout a file

Moving outdated records into archive files

Trigger Processing (Output)

@wbrk called "when break" processing is triggered at subtotal and grand total breakpoints on a report.

It is done at subtotal or grand total breaks in Request Output.

Common Uses

Performing calculations or lookups when Request Output changes subtotal levels or breaks for the grand total.

Sort/Selection Processing

Sort/Selection processing can be performed when you Request Output during the first pass, before the records are sorted and selected.

Common Uses

Prompting users for sort or selection categories using customized prompts.

Calculating percentages.

Version 6.0.01

CABEBACKUP ON/OFF (on by default)

CABEBACKUPMINS n (minutes between backups)

CABEBACKUPCT n (backup files per process)

While editing a process it will automatically be * backed up depending on the settings of these variables: * CABEBACKUP (ON/OFF) * CABEBACKUPMINS (MINUTES) * CABEBACKUPCT (NUMBER OF BACKUPS BEFORE ROLLOVER) * Backups can be restored through menu item 5.

Processing Table Ingredients

Processing tables test the conditions to see if the action should be carried out. When conditions are tested, they are either true or false.

If the condition is true, the action is carried out. If the condition is false, the program falls through that "then" line without executing it to the next "if" line.

If the condition line ("if") is blank, the action line ("then") will be carried out.

Ingredients

Fields

- Real fields
- Dummy fields
- System-maintained fields
- Associated fields

Literals

Text or numbers must be enclosed in quotes.

- Two quotation marks together ("") means null (nothing or blank)
- /SConnectives AND, both conditions must be true
- OR, either conditions must be true

Negate (NOT) Statements

NOT "not this", negates an expression

Relationship codes

- EQ or equals
- NE not equal
- GT is greater than
- GE is greater than or equal to
- LT is less than
- LE is less than or equal to
- CO contains
- XxF compare two fields

Punctuation

"n"	Literals
()	Groups parts of a condition
'	Remark ~
;	Command separator
:	to assign aliases (and overlay arrays)

To write more than one action on a line, separate each action with semicolons.

Math Operators

+	Add
-	Subtract
*	Multiply
3/	!

Text Operators

&	Join
<	Push left, and add one space
{	Squeeze left, with no space

Lookup file names - test if lookup is successful

If:

Then: lookup filename

If: not filename

Then: show "Not on File"

Labels - if conditions on that line are met

Example 1

past If: "complicated selection"

Then: "complicated action, too long to fit on one line)

If: past

Then: "rest of action"

Example 2

pastIf: "complicated selection, too long to fit on one line"

If: past connective "rest of selection"

Then: "action"

Selection set names - if specified conditions are met

If: current

Then: "action"

Where "current" is a saved selection set

Actions ("then lines")

What the program does if the "if" condition is found to be true or blank.

Actions contain:

expressions or assignments

commands

Screen Messages With SHOW

The SHOW statement gives you great flexibility in presenting information to the user at virtually any point during a filePro program. The basic syntax is very simple:

Syntax :

```
show "message"
```

This will place the message on Line 23 of the screen and center it for you. The message can not exceed 80 characters.

The "message" can be any expression allowed in filePro. For example, the "message" could be made up of the contents of a field.

```
show 14
```

This would simply put the contents of field 14 on the screen. It might be nicer to do it like this, though:

```
show "Field 14 contains"<14
```

This will print the text "Field 14 contains" and then leave one space and print the actual contents of field 14. The < operator is just that a "push left" operator. It tells filePro to push the next value one space away from the preceding text or value. There is another operator like <. It is {. This is called a "squeeze left" operator because it does not leave the one space the < does. It would squish the contents of field 14 right next to the letter "s" of the word contains.

If you wanted to print the content of several fields, you could do something like:

```
show 14<15<16
```

Assuming that these were City State and Zip fields, you might see:

```
Oakland NJ 07436
```

Again, the show command will mix and match text and expressions (a field is nothing but an expression, by the way, albeit a very simple one.) You could get elaborate, like this:

```
show 14{" "<15<16
```

This would print the following on the screen:

```
Oakland, NJ 07436
```

There is one more important string operator (that is what the < and { are called). The third and final string operator is the &. It simply puts the values it is connecting next to each other without leaving a space, squishing out space or doing anything at all. The same show statement above done with the & instead of the < or { would do this:

```
show 14&","&15&16
```

```
Oakland ,NJ07436
```

Why? Because the contents of field 14 do not entirely fill up field 14. Usually a city field has about 15 characters in it and Oakland only takes up 7. The & operator tells filePro to just place the next expression "next" to the previous one, do not add or take away any spaces so it does. FilePro knows how long field 14 really is and that is what it uses when you use the & operator. The same is true for the literal "," which is only 1 character long, so the state field NJ butts right up against it. State fields are only two characters long so the zip field butts right up against that and we are left with the funny mish-mosh above. Using the < and { and the & correctly allows you to format output to look just about any way you want it to look. For example, you could do something like this:

```
show "The contents of field 14 15 and 16 are:" < 14 { "," < 16 & " and I live there!"
```

The formatting will follow your instructions to the letter. The "<" operator will also work on output reports that you design.

When you join two or more fields with the & operator, you are "concatenating the fields". This means you are joining the fields right next to each other without taking away or putting any extra spaces. If you join a 14 character field and a 25 character field in this way, you need a 29 character place to put them either screen, paper, or dummy variable.

Reverse Video inside SHOW .

Any part of the message can be shown in "reverse video" (highlighted) by using the reverse video codes inside the message area, as in:

```
show "Are you \rSure\r you want to delete this record? (y/n)"
```

Anything between the two \rs will show up highlighted.

SHOW(r,c) message

There is a valuable variation to the show command. You can assign the row and column on which the message will appear. When you do this, the message is NOT automatically centered anymore. You would use:

```
show("10","4") "Please enter a bigger number here."
```

And it would appear at that point on the screen.

SHOWCTR(r) "message" command.

If you want to center a message on a line other than line 23, use the showctr command.

```
showctr("10") "I am in the middle of the screen."
```

NOTE: SHOWCTR must have a line number specified.

IMPORTANT: You can use SHOW to stop the user dead in their tracks. They must acknowledge that they have read your show message by pressing ENTER. To do this, put an @ as the first character after the opening quote of your message text.

```
show "@The balance due is TOO HIGH!"
```

This will print on the screen as follows:

```
The balance due is TOO HIGH!  
Press ENTER to Continue
```

The word ENTER will be surrounded by a highlight. The user can not do anything else but press Enter if he wants to continue working.

If you were to have no message text, the @ still works, just put it inside quotes by itself:

```
show "@"
```

Produces

```
Press Enter to Continue
```

The use of the @ does not preclude mix and matching expressions to form your message text.

```
show "@The balance due is ${22<"This is TOO HIGH!"
```

You can obtain more emphasis by issuing a BEEP command along with various messages. This keeps the user on his toes. Otherwise, they are likely to start ignoring things. Just put the command BEEP immediately before your SHOW statement.

```
If:  
Then: beep ; show "@Call this guy up right now."
```

Clearing a SHOW statement

If you do not use the @ which will stop the user at the SHOW line and then automatically clear the show message, the text of your message will remain on the screen until you take it down (or until a variety of other things clear it.) For the most part, it is up to you to clear things that you put on the screen. If you are clearing the default show line (#23), you can just issue the command:

```
show ""
```

This, of course, means SHOW nothing, so the line is essentially rewritten with nothing anything that was there will be cleared.

If, on the other hand, you have shown something on a different part of the screen, you must clear it yourself with the same amount of spaces that your message used. In other words if you issue:

```
show("10", "3") "Error, bad code"
```

Later, when you want to remove the show message from that location, you would have to do something like:

```
show("10", "3") "          " <=there are 15 spaces here
```

This would "cover up" the message with blanks.

There is a short hand for this using a dummy variable. You could assign a dummy variable the length of 15 and set this variable equal to nothing and then show that variable on the screen. This would accomplish exactly the same thing.

```
aa(15)=""; show("10", "3") aa
```

IMPORTANT

Please note that you can put more than one command on a "then" line as long as you separate them by semi-colons. Also note that SPACES are not important between commands. That is, you can use:

```
show "@Did you see this";show "@Are you sure?"
```

Or

```
show "@Did you see this?" ; show "@Are you sure?"
```

SHOW POPUP

This is the same as the SHOW command except that it puts your message inside a box that is centered on the screen. Everything works the same as the simple SHOW command,

including the @ for stopping the user and making him press ENTER.

```
show popup "I m boxed in the middle."
```

The SHOW POPUP command can also place the box anywhere on the screen within the limitations of the row and column designations that will fit the whole box. The row and column designate the top left corner of the popup box.

Syntax

```
show popup (r,c) message
```

There is one more variation to SHOW POPUP. It allows you to number the boxes you put on the screen.

```
Show popup (r,c,popupnum) message
```

This means you can put several messages anywhere on the screen simultaneously. The number.

Clearing Show Popups

You must clear show popups with the CLEAR # command. You must issue the clears command with the number of the popup you are taking down (off the screen). If you overlap the show popups, they must be taken down in reverse order of the way they were put up.

Simple Operations on Processing Tables

The idea of a processing table is simple. It is a place where you write code that controls your program. The code does not have to be complex. When you are starting off, it can be as simple as 4th grade math and it looks just about like that too.

Getting Totals and Balance Due

Probably the most basic thing you would ever want to do is add up the charges on a multi-line invoice and put the total on the screen somewhere. Next, you might want to take a payment against this charge and show the balance due after the payment. Here is the INPUT processing table that will do this.

We will assume that fields 5 through 14 are the ten charge amount fields on an invoice record. We will let field 15 be the subtotal of those charges, 16 be the tax amount and 17 be the total charges (the addition of the subtotal and the tax). Field 18 will be the total payment received, and field 19 will be the balance due field.

Filename: classinv

Number	Field Heading	Len	Type
1	Acct. Code	6	allup
2	Date	8	mdy/
3	Invoice#	6	.0
4	Status	1	allup
5	charge_1	7	.2
6	charge_2	7	.2
7	charge_3	7	.2
8	charge_4	7	.2
9	charge_5	7	.2
10	charge_6	7	.2
11	charge_7	7	.2
12	charge_8	7	.2
13	charge_9	7	.2
14	charge_10	7	.2
15	subtotal	8	.2
16	tax	7	.2
17	total_charge	8	.2
18	payments	8	.2
19	balance_due	8	.2

Key segment record length: 130

Screen .1

CLASSINV

Acct. Code: *1

Date: *2

Invoice#: *3

Status: *4

charge_1: *5

charge_2: *6

charge_3: *7

charge_4: *8

charge_5: *9

charge_6: *10

charge_7: *11

charge_8: *12

charge_9: *13

subtotal: *15

tax: *16

=====
total_charge: !17

charge_10: *14

payments: *18
balance_due: !19

The processing to do simple calculations begins with math operators. These are the +, -, * and / operators you learned in grade school. FilePro can add, subtract, multiply and divide just about anything. The difference is, you are usually adding, subtracting and dividing the "contents" of a field, you do not need to know the value in the field, just specify the fields and where you want to place the "answer". In other words if you want to total up 5 fields and put the "answer" into a 6th field just write it out the way you would any math statement. 8=1+3+92 just remember that you are adding up the contents of fields 1, 3 and 92 not adding "1", "3" and "92". Although filePro can certainly do math on real numbers, you just have to put real numbers inside quotes, otherwise it will think you are talking about a real field.

File Name: classinv
Processing Table: input

If:
Then: end

Totals If:
Then:
If:
Then: 15=5+6+7+8+9+10+11+12+13+14
Then: total_charge = subtotal+tax
Then: balance_due = total_charge - payments
Then: display
Then: return

@wlf4 If:
Then: gosub totals ; end

@wlf5 If:
Then: gosub totals ; end

@wlf16 If:
Then: gosub totals ; end

@wlf18 If:
Then: gosub totals ; end

Date Math

FilePro can also do math on dates. If you take one date and subtract another from it, filePro will tell you how many days there are between those two dates. Or, if you add a real number of days to a date, filePro will tell you the resulting date. For example:

If:
Then: aa(8,mdy/) = @td + "30"

This will put the resulting date into the dummy variable aa. We know we are going to end up with a date so we give aa a date type edit at the same time that we put it on the left side of the equals sign. If you have code on a "then" line that has an equals sign in it, filePro will ALWAYS take the value it determines is on the right side of the equals sign and assigns it to become the value of whatever field is on the left side of the equals sign. This is ALWAYS true. Even when it may not look right to you. If you are used to algebra, the following would look very wrong, but it works just fine in filePro.

If:
Then: aa = aa + "3"

How can aa be equal to itself plus "3". That is not what this says in filePro processing terms. To filePro, this means "take whatever value is currently in aa, add "3" to it and stuff this new value into aa. The next time filePro looks at aa, it will be 3 higher than it was before this "then" line was encountered. If you understand this, you understand just about all there is to making filePro do all its tricks.

Steps to Define a Key-field Lookup

Lookup Wizard	Position cursor on an empty "Then" line and press the Define Lookup key (F5, Ctl-R). (You can write the statement on the processing line yourself, by NOT pressing the Lookup wizard key.)
Select Lookup File	"Lookup From File:" Name of the other file, the lookup file.
"Name Of Lookup Is?"	You can give a shorter "alias" to this lookup filename.
"How Is The Record To Be Found?"	K Key Field, R Record Number, F Free Record, Z Fuzzy Search You are cross-referencing the files based on a key field. The key field in the current file has a corresponding field in the lookup file on which an index is built.
"What Index Is To Be Used?"	A-P or 0-9, must be built on the corresponding field in the lookup file.
"What Field In 'current filename' Contains The key:"	The field that contains the data that can be used to reference the correct record in the lookup file.
Lookup Match Criteria	X Key Must Match Exactly G Use Next Greater Match L Use Next Lower Match When you pick the "less than" or "greater than" modes, the program looks for an exact match first. If it finds no exact match, it selects a value just less or greater than the one requested.
Lookup Fail Action	B Blank The Field N Do Nothing E Report An Error B Allows the user to fill in and save despite incomplete entries in the lookup file. When data cannot be looked up from the lookup file, the program puts blanks in the affected fields. N The lookup filename holds a true/false value for the success of the lookup and this can be used for testing. This allows you to maintain control of the failure yourself. (i.e., put up a message like "Product not on file, do you want to add it?") E Displays the message "Lookup Failed - Try Again". Then it returns you to the update mode in the field where the lookup failed (the key field) In automatic processing with -E flag, the error is displayed before the record appears, but you can't get into the record to change the problem. Do not use -E on automatic processing tables. In when-field processing, you cannot use the -E flag. In output processing, the -E flag halts the processing and kicks you out. Do not use -E on output processing tables.
Create Browse Lookup?	This allows you to build elaborate row-oriented display lookups. You are returned to the processing table. The lookup statement is now written on the "Then" line.

Handling Lookup Failure

Definition

The lookup fails if no match is found in the "lookup" file. The lookup filename will "hold" the success or failure of the lookup. You must tell the program what to do.

Sample

```
If: not filename  
Then: show "@Not on file, try again.":goto again
```

Making Field Assignments

Once the lookup file and record(s) are specified, you must tell the program how the data is to be used. Usually, this means "assigning" the lookup fields to fields in the current file. This just means setting the fields in this file equal to the ones you found in the lookup file (or vice versa). Result fields ALWAYS go to the LEFT of the equal sign. Make sure the edit types of the two fields match.

You can make assignments in either direction once you have done a lookup. You can change things in the lookup file or change things in the current file.

Receive - Data is received into the current file from the lookup file

x=filename(n)

filename is the name of the lookup file

x is the field in the current file which receives the data

n is the field in the lookup file that contains the looked-up data

Send - data is posted from the current file into the lookup file.

filename(n)=x

filename is the name of the lookup file

n is the field in the lookup file to which data is sent

x is the field in the current file that contains the data to be sent

Triggers

There are several kinds of processing "triggers". This section will cover @when leaving and @when entering triggers. They are used only on the INPUT processing table.

When you first work with FilePro processing tables, you are happy to find that math can be performed on the data: payments subtract from charges to leave balances due, columns of figures add up into totals, and due dates pop into place. You can be shown help, moved to other screens, or be presented with menus, all matter of magic can happen to your screen after you press the ESCAPE ESCAPE sequence. In many instances, though, it would be wonderful to do some of this magic before storing the record. This can be accomplished by using the @when processing feature of FilePro. Pronounced "at when" processing, this is only one type of trigger that filePro can intercept.

There are several kinds of @when processing; we will discuss two here, @wlf (at when leaving field) and @wef (at when entering field). These are triggers that can initiate a processing routine. For instance, when you leave field 22, the date field of an invoice, you want to immediately calculate the due date (say 30 days from this date) and display it in field 23. Your input processing table (@when processing NEVER goes on the automatic table) might look like this:

```
      If:
      Then: end
@wlf22  If: 22 eq ""
      Then: 23 = "" ; display ; end
      If:
      Then: 23 = 22 + "30" ; display ; end
```

In English this processing says: when the cursor leaves field 22, if field 22 is empty (programmers say, if 22 is equal to null or if 22 is null) then set field 23 equal to null. Then display all the fields on the screen with their most current values, then end. (In other words, this little snippet of processing is done, so end it and put the cursor into the next field on the cursor path.)

Note 1: There is an end statement above this processing. You must not allow other processing to fall into an @when leaving or @when entering routines. The line above them must either stop the processing or direct it somewhere else, such as "goto label", "return", "exit". For now, and until you get far more advanced, it is a safe idea to put all of your @when processing snippets at the end of your regular input processing (the stuff that happens when you press ESCAPE ESCAPE) and be sure there is an END statement immediately preceding each @when label. (or something that will not let the processing fall through by mistake.) (Advanced users can use @when processing as a label, but this is because they thoroughly understand the difference between using it as a trigger and as just another piece of regular code.)

Note 2: All branches of each @when processing must be "stopped" correctly. There are five commands that will correctly stop an @when process (END, SCREEN, SKIP, ESCAPE, EXIT), and they do JUST THAT. Study the following example. Let's say you want the operator to be brought to SCREEN 2 when his cursor leaves field 17. On SCREEN 2 are some fields (13, 14 and 15) which you want to add up and put into field 16 when the operator is done on SCREEN 2. You have written the following code:

```
      If:
      Then: end
@wlf17  If:
      Then: screen 2
      If:
      Then: 16 = 13 + 14 + 15 ; display ; end
```

Everything looks beautiful, right? NO! The SCREEN command is an "ending" command for the @when processing. True, leaving field 17 will cause that command to happen, but that is it! The processing on line 3 will never get executed. When the operator presses ESCAPE ESCAPE on screen 3, the processing table will pick up wherever the program counter was last sitting (probably at the top of the INPUT table). Why? Because that is where input processing starts and INPUT processing is the table that is run when ESCAPE ESCAPE is pressed. Just because the @wlf and @wef trigger processes are on this table, does not mean they get executed when ESCAPE ESCAPE is pressed. They ONLY get executed when their trigger is pulled when the user actually moves the cursor into or out of the field they specify. Line 3 of the above code will never be reached.

These are the two keys to understanding @when processing. First, every possible branch of actions must eventually be ended with an "ending" command (one of the 5 listed above). The real INPUT table program counter/pointer has never moved, and will never move because of a trigger process being executed. It stays right where it is. The next line that will be executed when next INPUT processing is called (ESCAPE/ESCAPE) is the same one that would have been executed with or without any @when processing..

The same rules apply to @when entering processing. An example might be the following. You have a new employee, Fred, who is allowed to change anything on a record but the price field 14. All of your other operators CAN change this field. Your code might look like this:

```
      If:
      Then: end
@wef14  If: @id eq "fred"
      Then: skip
      If:
      Then: end
```

It may look strange, but it is correct. As the cursor "enters" field 14, this processing routine is triggered. FilePro checks the user ID of the operator and if it is equal to "fred", the process skips this field and deposits the cursor (and Fred) into the next field on the cursor path. Any other user ID and the process just ENDS. Which, in this case, leaves the cursor right where it was headed, field 14. In other words, there was NO @wef14 processing, just do nothing.

Using @when Leaving As Powerful Edits.

One thing you often want to do is disallow certain data in a field. You might use something like the following to accomplish this :

```
      If:
      Then: end
@wlf3   If: 3 eq "whatever-you-don't-want"
      Then: beep; show "@Sorry, wrong data"; 3=""; screen 3
      If:
      Then: end
```


Types of Processing Tables

Automatic Processing

Automatic processing is done every time a record is accessed from disk. Records are accessed from disk at two times for user interaction, during Request Output and during Inquire, Update, Add. Other times records are retrieved, for example, during index maintenance, there is no need to run any automatic processing.

More specifically, automatic processing is performed once if a record is accessed for output, but several times when a record is displayed on the screen in Inquire, Update, Add.

Automatic processing affects data as it comes from the disk to be displayed on the screen, but it does not, and should not, affect data on its way back to the disk. The automatic table should never change anything on the record.

You can use automatic processing for standard processing that is the same every time you access the record, whether in Inquire, Update, Add or Request Output.

Common Uses

Calculating a math formula, without saving it, whenever a record is accessed in Inquire, Update, Add or in Request Output.

Looking up an address from another file using a customer code, so that the address appears on the screen but is not saved in the records of the current file.

There are thousands of good uses for the automatic table. One of its most important features is that it ties together all other tables so that values can be passed through it to them.

Input Processing Tables

Input processing is done on a single record when it is added or updated. The processing is performed after the user presses the SAVE key. (Unix=ESCAPE ESCAPE, DOS=ESCAPE <= this can be customized).

Remember the user may press SAVE in 2 cases, either saving data for a brand new record, or saving updates (changes) to an existing record. Sometimes the programmer will want different things to happen in each of these cases.

Use input processing when you want processing to occur on a record-by-record basis after you add or change data in the records.

Common Uses

Filling in an empty shipping address with the billing address.

Attaching screens to one another so that the user is taken from one screen to the next.

Posting data to another file on a record-by-record basis after the user presses ESC/ESC.

Output Processing Tables

Output processing is performed during Request Output on a selected group of records.

When you Request Output the program performs 2 passes through the file. First, it selects and sorts the records. Then it performs the output processing on each record.

It may or may not include the printing of output. If there is no printout, it is called a "Processing - Only" operation.

Common Uses

Formatting data for special display on the output device.

Accumulating special totals on reports.

Posting data to summary files

Uses of Processing - Only Operations

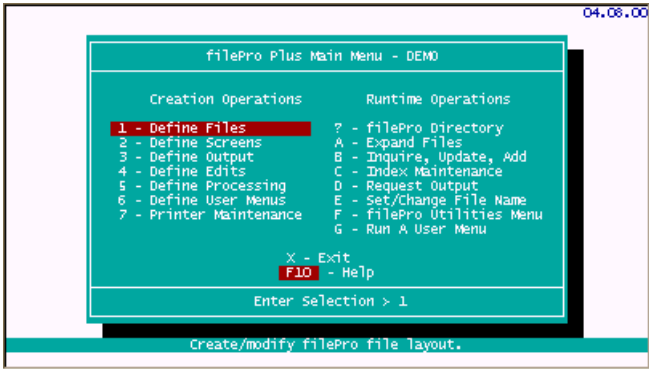
Mass recalculation - performing a math operation on a field throughout a file.

Global update - performing a textual change throughout a file.

Moving outdated records into archive files.

Creation Operations

The left side of the filePro Main Menu includes the Creation Operations, while the right side of the menu lists Runtime Operations. This primer will walk you through each of the Creation Operations and then the Runtime Operations in sequence.



The filePro Plus Main Menu is a developer menu, so you will not normally make this menu available to your end-users. You will typically create [user menus](#) to control what your end-users are able to do.

The above menu separates the filePro options into logical groupings. The "Creation Operations" are used for creating and modifying filePro elements, while the right side of the menu "Runtime Operations", is used to test and perform maintenance on what you have created.

Prior to getting into a detail review of each option, suggest that you browse through the topics to get an overview of what each of the options entails.

Define Files - Option 1

Contents of this section

Define Files Description
Beginner Design
filePro and Non-filePro files
Data Types [edits]
Create Default Formats
Qualifiers (basic)

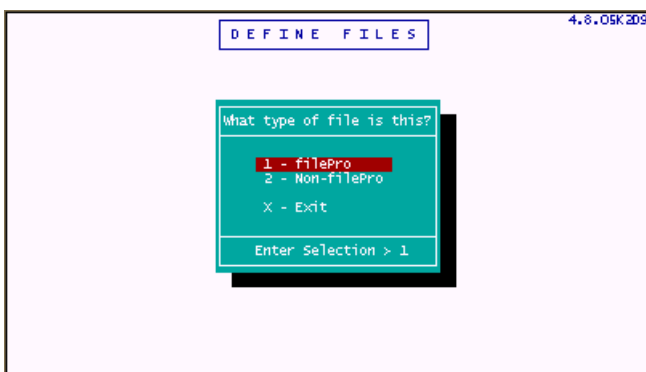
Description:

When defining a new filePro file, filePro will create a subdirectory in the "filePro" directory with your chosen filename. All of the screens, formats, user data, etc., for this filePro file are stored as regular system files within this named directory. Therefore, a filePro filename actually refers to a directory name.

Under Unix, the filePro filename can not exceed 14 characters in length. Under WINDOWS, the filename size can be up to 8 characters long. The WIN95/98/NT version allows up to 32 characters for filename. It is good practice to use filenames that have more than two characters.

Beginner Design

When you define a filePro file, after you pick <NEW> and enter a filename of your choice, you will see the following screen:



"filePro & Non-filePro files

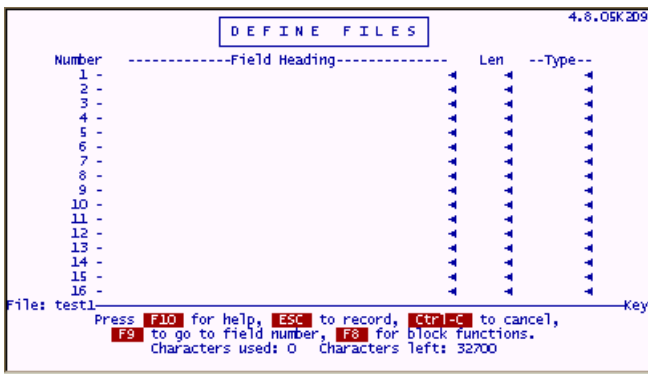
filePro can work with filePro format files and flat ASCII files. The flat ASCII files or "non-filePro" files are also called "alien" files because they are not structured in the filePro format. It is normally better to use the filePro file type since it keeps track of things like "Creation Date", "Date Last Updated", and other system information that is useful in selecting and maintaining the records.

When selecting either file type, you are prompted for a creation password.



With this password, you can prevent others from changing your file structure and formats. If you are a beginner, do not use passwords. Creation passwords can always be applied later.

After the password prompt, you are presented with a "Define Files" screen as follows;



Field Heading

You simply enter a description that is meaningful for each field under "Field Heading". Although the description can be anything you want (up to 32 characters), use descriptions that are meaningful to the user.

Headings longer than 21 characters may be truncated on some screens.

Len

The Len is the maximum number of characters that the field will hold. Remember to leave space for punctuation in dates and non-integer numbers i.e. "04/21/99" is 8 characters, "99999.99" is 8 characters. Many edits (discussed later) such as "Dates", "Time", etc. have specific field length requirements. The Len is limited to a number between "0" and "999". Many programmers use a zero "0" in the "Len" column to reserve lines for spare fields but normally a number "1" through "999" would be entered as the length of a field. It is best to limit the length of the field to what you think you really need since longer fields take longer to sort and will use up unnecessary space when storing the data.

Type (Edit)

The "Type" is short for "Edit Type". This column determines the kind of information that will be allowed in the field. If the "Type" column is left blank or contains an " * ", the user will be able to enter just about anything. filePro provides pre-defined types (edits) such as; "allup" to capitalize any data input into the field; "phone" to insert parentheses and dash in a typical phone number format. You can also create "user edits" which will be discussed later.

Data Types

Data can be classified into four major groups or "Data Types" as follows;

Character:	Any combination of ASCII characters (almost anything)
Numeric:	Integer numbers; numbers with decimal values or commas
Date:	Dates in various formats mdy; mdy/; ymd; ymd/; etc.
Time:	In hours & minutes - hm; or hours, minutes & seconds - hms

IMPORTANT: If you will need to perform any math operations on a field, be sure that the field is defined as a "Numeric" data type. If you need to perform any date calculations on a field, use the "Date" data type.

Example: To make sure that a Social Security Number is always entered correctly, we can enter "SSNUM" in the "Type" column to ensure that Social Security Numbers are always entered; with the correct number of digits; as numbers; with the "-" dashes which will be automatically insert dashes, if not provided.

User Input	Edit Results	Description
123121234	123-12-1234	Valid Data & dashes inserted
123-12-1234	123-12-1234	Valid Data
a12121234	SSN Invalid	Data rejected "a" is invalid

The "SSNUM" example is just one of many "edits" pre-defined for you in filePro. Edits are contained in tables called "Edit Dictionaries". The "GLOBAL Edits Dictionary" contains the pre-defined edits furnished by filePro to cover commonly used fields like phone, zip, state, allup, cheque, SSN etc. Click on [edits](#) to view the "GLOBAL Edits Dictionary".

User Edits

The data allowed in a field can be further constrained by creating your own filePro "User Edits". "User Edits" are maintained and associated with a specific filePro directory. Therefore, "User Edits" are unique to each filePro file unless you include the edit in the "GLOBAL Edits Dictionary".

A typical filePro file

The following is a snapshot of a filePro called "FPCUST".

```

4.8.05K209
  DEFINE FILES
Number -----Field Heading----- Len --Type--
 1 - customer number                4  uplow
 2 - last name                      15  uplow
 3 - first name                     15  uplow
 4 - address                        15  allup
 5 - city                           15  allup
 6 - state                           2  state
 7 - zip                             10  zip
 8 - year-to-date sales             9  .2
 9 -
10 -
11 -
12 -
13 -
14 -
15 -
16 -
File: FPCUST
Press K for key, D for data, U to update, H to hardcopy,
F10 for help, PgDn, PgUp to scroll, X to exit.
Previous record length: 85 Current record length: 85

```

This file is a typical customer file that you might create. Notice that the field headings are self-documenting, so when the end-users access the data, they will know exactly what it is. The "Len" length of the fields are typical of what you will need to accommodate "last name" first name" etc.

The "edit types" column contains "uplow" for the name fields which will always capitalize the first letter in the name and set all other characters in the name fields to lower case unless you have purposely typed a character in caps. The "allup" edit will set the address and city fields to "all uppercase". This is a normal U.S. Postal Service requirement. The "state" edit type will ensure that only valid state abbreviations can be entered. The "zip" edit type will handle both 5 character and 10 character zip codes. The ".2" edit for "year-to-date sales" will ensure that dollar amounts will always be formatted with 2 decimal places.

Selection Options

- K for Key** Press K for the key segment. filePro data can be split between two Segments called "key" and "data". When entering "Define Files" The "key" segment is used by default.
- D for data** Press D for data segment. Most developers no longer use the "data" segment. It is still maintained for compatibility to older versions of filePro.
- U to update** Press U to define or change fields.
- O for Options** Press O to select Advanced Options
- H to Hardcopy** Press H to print a list of the fields and definition.
- PgUp PgDn** Use to see fields below and above the edges of the screen.
- X to Exit** Press X to record the changes.

The "Previous record length" and "Current record length" are provided for informational purposes.

Press "U" to update

```

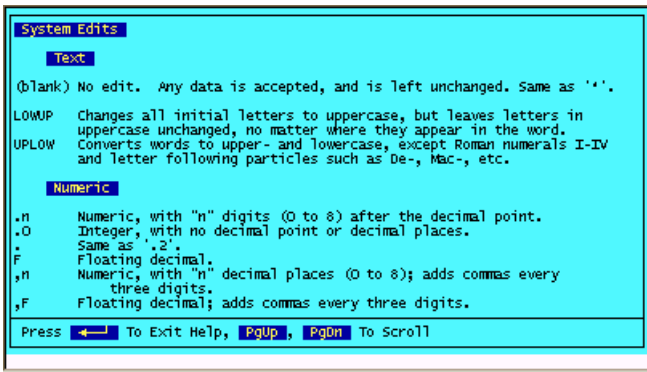
4.8.05K209
  DEFINE FILES
Number -----Field Heading----- Len --Type--
 1 - customer number                4  uplow
 2 - last name                      15  uplow
 3 - first name                     15  uplow
 4 - address                        15  allup
 5 - city                           15  allup
 6 - state                           2  state
 7 - zip                             10  zip
 8 - year-to-date sales             9  .2
 9 -
10 -
11 -
12 -
13 -
14 -
15 -
16 -
File: FPCUST
Press F10 for help, ESC to record, Ctrl-C to cancel,
F9 to go to field number, F8 for block functions.
Previous record length: 85 Current record length: 85

```

When entering Update, the selection options change at the bottom of the screen.

- F10 for Help** Retrieves system help for define files.
- ESC to record** Saves the changes made to your file definition. (this is ESC on UNIX and other systems. Refer to Terminal Guides)
- Ctrl - C** Cancels changes made.
- F9 to go to.** Used with definitions with many pages to quickly go to a Specific field number.
- F8 for Block...** Allows you to copy rows quickly.

By pressing the "F10 for Help" key, you can retrieve the system help for each section. The following is an example of what you see for the "Type" column.

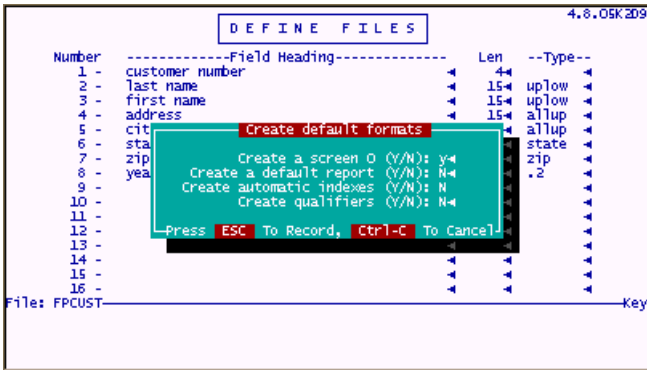


Select Options

Create Default formats

After creating / updating in "Define Files" and saving the file definition , you are presented with various options for creating default formats. When typing a "Y" for these options, defaults will be established or you will be presented with additional screens to specify the values for the defaults.

Creating a screen 0 (Y/N)



"filePro" tries to make a screen from the fields provided. Sometimes there are too many fields or the field lengths are too long. In these cases, filePro truncates the screen to whatever it can do.

Create a default report (Y/N)

The default report will be labeled "default.out" in WINDOWS or "out.default" in UNIX/LINUX systems.

Create automatic indexes (Y/N)

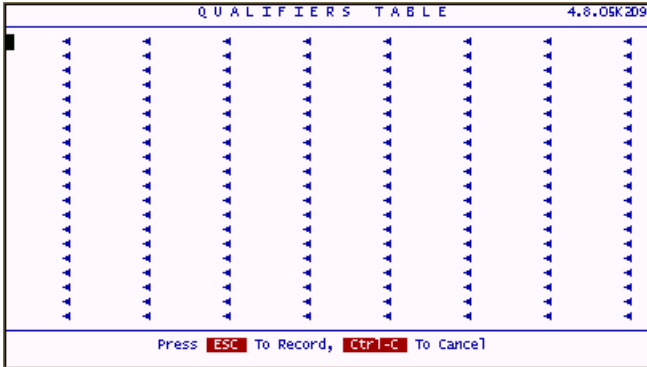
You can elect to create indexes at this time or create at a later point using the "Index Maintenance" option. Automatic indexes can be established for any field in the key segment. For filePro versions prior to 4.5, automatic indexes could only be built on multiple fields if the fields were concatenated (if maintained in the map next to each other). For example, in the "FPCUST" file, an automatic index could be built on "last name" and "first name" by extending the sort length to 30 characters, 15 characters for "last name" and 15 characters for "first name". With versions 4.5 and later, you can sort on multiple fields no matter where they are defined in the map.

NOTE: ver. 5.8.03 - ddefine will now use the version 4.5+ dxmaint interface when making indexes for new files

Set PFDDEFXMAINT to OFF to revert to the old 4.1 index routine in ddefine

Create qualifiers (Y/N)

Qualifiers allow you maintain multiple sets of data for a single filePro program definition. This is valuable for maintaining a separate set of data for testing your design i.e. use a qualifier name "TST" for test. Other uses include maintaining sets of data for multiple customers i.e. payroll systems for company A, company B and company C, or inventory data for more than one convenience store. As a developer, the qualifier feature is valuable for testing or troubleshooting a data problem for customers. Qualifiers allow you to maintain multiple copies of data for a single set of filePro program files. Data from a customer site could be copied with a qualified name to the same filePro directory. So when you make a change to a process, report or screen, the changes can be easily maintained since you will only have a single version or a master of your programs. The following screen will be presented by define files when you set the "Create qualifiers" option to "Y".



Enter the qualifier names you wish to create. After pressing ESC, the files associated with filePro data will be created in the respective filePro directory.

Example:

If you entered qualifier "TST" for test, the filePro directory would contain "keyTST", "dataTST", "indexTST.a", etc. The qualified sets of data can be accessed through processing , by using the appropriate menu flags or from the command line.

Other uses of qualified sets of data may include developing divisional accounting systems. The data for each division may be maintained in qualified data sets while the corporate-wide data may be maintained in the non-qualified data set. filePro processing allows you to easily post and retrieve data between qualified sets with the "LOOKUP" command.

Advanced Concepts - Define Files

Contents of this section

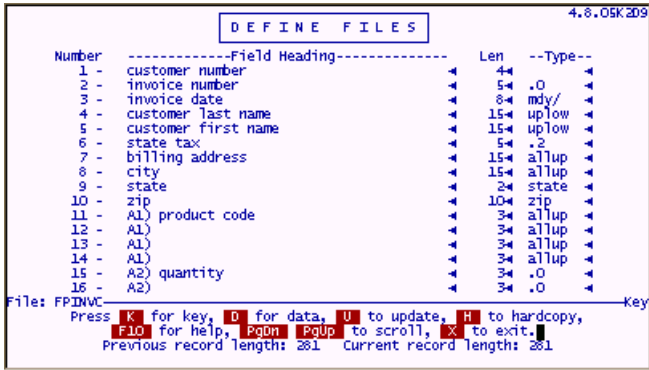
- Design Considerations
- Associated Fields
- Non-filePro Files
- Qualifiers

Design Considerations

You can very quickly design applications with filePro ranging from a simple "single" file application (rolodex) to complex relational database applications containing many filePro files (accounting system). Although it is relatively easy to go back and make changes with the "Define Files" option, it is best to give some thought to organization and logical grouping of fields in this option on your initial design. If you forget something or put something in the wrong place, it is not difficult to add or change the fields at a later time since filePro will take care of restructuring your data with ease. You dont want to be forced to re-visit a design any more than necessary, so this preliminary organization will save you time.

Associated fields

These fields are sorted and selected as a group. Whenever one member of the group is specified, all members of the group are considered. These fields can only be defined as real fields with the "Define Files" option. The following provides a sample of associated fields for "product code" and "quantity".



Associated fields are designated with a "letter" , a "number" followed by a close parenthesis, and then whatever description you want to give the group of fields. You can also add subfields (the quantity fields) by starting with the same letter and advancing the number.

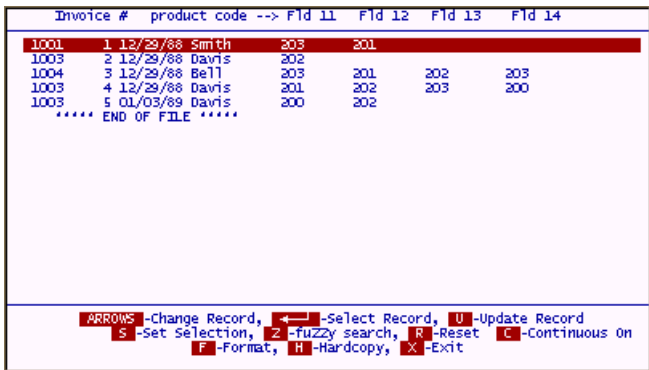
Example:

- A1) product code A2) quantity
- A1) A2)
- A1) A2)
- A1) A2)

In the above example, we now have two associated field groups containing four fields each.

Selecting

To demonstrate the advantage of selecting data using associated fields, refer to the following;



In the above screen, you can see that product code "201" applies to Invoice # "1", "3" and "4". When selecting field "A1" eq "201", you would only see invoices where fields 11, 12, 13 or 14 contains product code "201".

```

INQUIRE, UPDATE, ADD 4.8.05K409 DEMO
1- customer number      7- billing address      13- A1)
2- invoice number       8- city                 14- A1)
3- invoice date         9- state                15- A2) quantity
4- customer last name   10- zip                 16- A2)
5- customer first name  11- A1) product code    17- A2)
6- state tax            12- A1)                 18- A2)

Enter Relationship (E0,ME,GE,GT,LE,LT,RG,C0): eq
Enter A1) product code: 201
Enter Connective (and/or): |

Press PgDn For Next, PgUp For Previous Fields
Press F5 To Toggle Between Field Headings And Lengths/Edits

```

Results of "A1" eq "201"

```

Invoice #  product code --> Fld 11  Fld 12  Fld 13
-----
1001      1 12/29/88 Smith      203      201
1004      3 12/29/88 Bell       203      201      202
1003      4 12/29/88 Davis      201      202      203
***** END OF FILE *****

Select
ARROWS -Change Record, ← -Select Record, U -Update Record
S -Set Selection, Z -fuzzy search, R -Reset C -Continuous On
F -Format, H -Hardcopy, X -Exit

```

To get the same results without associated fields would require a more advanced search as follows;

```

Group  Field  Heading      Rel  Value
-----
OR     11  A1) product code  eq   201
OR     12  A1)                eq   201
OR     13  A1)                eq   201
OR     14  A1)                eq   201
      15  A2)
      16  A2)
      17  A2)
      18  A2)
      19  A3)
      20  A3)
      21  A3)
      22  A3)
      23  A4)
      24  A4)
      25  A4)
      26  A4)
      27  A5)
      28  A5)
      29  A5)
      30  A5)

Selector Sentence:
File: FPDVNC
Enter Selection >
U To Update, C To Clear, S To Save, L To Load, H To Hardcopy
← To Select Records, X To Exit, F10 For Help

```

The above selection is not extensive for a group of four fields, but if it were 10 fields, you can see how time consuming creating the selection could be.

Sorting

Another big advantage of associated fields is sorting. We can now sort on the entire group of fields as a single group. The following example shows a typical sort.

```

Field list
1- customer number  11- A1) product code  21- A3)
2- invoice number   12- A1)                22- A3)
3- invoice date     13- A1)                23- A4) unit price
4- customer last name 14- A1)                24- A4)
5- customer first name 15- A2) quantity      25- A4)
6- state tax        16- A2)                26- A4)
7- billing address  17- A2)                27- A5) total cost
8- city             18- A2)                28- A5)
9- state            19- A3) description    29- A5)
10- zip             20- A3)                30- A5)

Sort definition
Sort Field: A1)
Length: 3)
Descending?
Subtotal Field? X)

Press ESC To Record, Ctrl-C To Cancel
Press F10 for help

```

The results of sorting the file by "Customer number" and then "product code".

Product/Invoice Cross Reference

Date: Aug 24, 1999 Page: <fpr

cust#	customer name	product	description	quant	invoice date
1003	Sam Davis	200	Clock	1	4 12/29/88
1003	Sam Davis	200	Clock	2	5 01/03/89
SUBTOTAL FOR PRODUCT: 200		RECORDS: 2	QUANTITY: 3		
1001	Bob Smith	201	Vase	1	1 12/29/88
1004	Brian Bell	201	Vase	1	3 12/29/88
1003	Sam Davis	201	Vase	1	4 12/29/88
SUBTOTAL FOR PRODUCT: 201		RECORDS: 3	QUANTITY: 3		
1003	Sam Davis	202	Ash Tray	10	2 12/29/88
1004	Brian Bell	202	Ash Tray	3	3 12/29/88
1003	Sam Davis	202	Ash Tray	1	4 12/29/88
1003	Sam Davis	202	Ash Tray	3	5 01/03/89
SUBTOTAL FOR PRODUCT: 202		RECORDS: 4	QUANTITY: 17		

Press any key to continue

Without the associated field "A1" to sort on, it would very difficult to even get these results since the product code fields 11, 12, 13, 14 could only be sorted using four sort levels i.e. 1st, 2nd, 3rd, 4th sort respectively.

Field list

1- customer number	11- A1) product code	21- A3)
2- invoice number	12- A1)	22- A3)
3- invoice date	13- A1)	23- A4)
4- customer last name	14- A1)	24- A4)
5- customer first name	15- A2) quantity	25- A4)
6- state tax	16- A2)	26- A4)
7- billing address	17- A2)	27- A5) total cost
8- city	18- A2)	28- A5)
9- state	19- A3) description	29- A5)
10- zip	20- A3)	30- A5)

Sort definition

Sort Field:	11	12	13	14				
Length:	3	3	3	3				
Descending?								
Subtotal Field?								

Press ESC To Record, Ctrl-C To Cancel

Press F10 for help

Product code "200" would only be sorted to the top of the list for all records only if it is entered into field 11 and never entered in fields 12, 13 or 14. If product code "200" was entered into any other field than field 11, it would show up further down the list after all product codes were sorted for field 11 as depicted in the following example.

Results of sorting by fields 11, 12, 13, 14

Product/Invoice Cross Reference

Date: Aug 24, 1999 Page: 1

cust#	customer	11	12	13	14	Desc.	Qty	invoice date
1003	Sam Davis	200	202			Clock	2	5 01/03/89
1003	Sam Davis	201	202	203	200	Vase	1	4 12/29/88
1003	Sam Davis	202				Ash Tray	10	2 12/29/88
1001	Bob Smith	203	201			Lamp	12	1 12/29/88
1004	Brian Bell	203	201	202	203	Lamp	4	3 12/29/88

Press any key to continue

Notice that the description, quantity, and date fields relate only to the first associated field (field 11) on the above report and product codes seem to be in random order even though they are sorted on the individual fields. The above report is not very useful and actually misrepresents the data.

Other Uses include calculating values

Associated fields are also valuable for calculating values. You can obtain a total_cost in a processing table by multiplying an associated field for quantity x unit cost.

$$@wlfA4): Total_cost = A2) * A4)$$

where: A2) = quantity; A4) = unit cost

This offers a great advantage in processing since you only need one line of programming code to calculate all associated subfields.

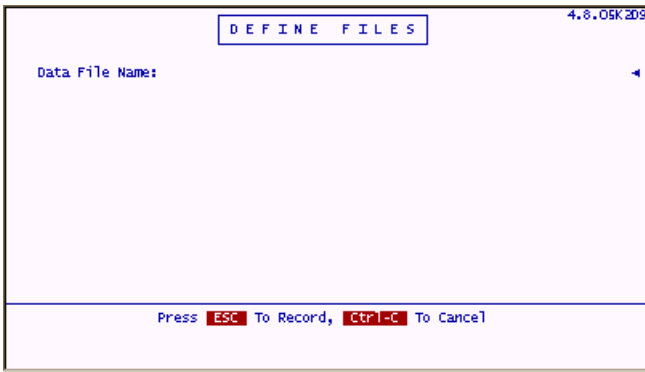
NOTE: You can have up to 32 instances of any subfield.

Non-filePro Files

With filePro, you can maintain data from other programs. The differences between filePro and non-filePro files are:

System maintained fields like @CD (creation date), @CB (created by), etc. are not available for Non-filePro files.

New records are only added to the end of the non-filePro file. Therefore, you can not overwrite deleted records. This will leave spaces in the non-filePro file that you may have to periodically scrub to keep the files as small as possible. After selecting the "Non-filePro file" option from define files, you will see the following screen.



Enter the full path (up to 58 characters) of your non-filePro data file and include the extension of the file if appropriate.



If the file does not exist, you will be prompted to create the empty file. At this point, there is no difference between defining a new non-filePro and filePro file.

You are limited to using ASCII type files with fixed record lengths. If you have an existing alien file that you want to maintain using filePro, you will need to have a list of the file's fields, field lengths, and know the record length. Keep in mind that special characters like CR/LF (carriage return/linefeed) have to be added to your file definition if these are used in the alien file.

Non-filePro File Utilities - filePro offers conversion utilities for some non-filePro file formats.

Click on Utilities .

General rules:

Respect the 8-character limit of WINDOWS if you want portability across platforms. The *nix limit is 14 characters, and WIN95/98/NT is 32 characters.

Wildcard-group your applications by name prefixes.

Example:

Nexapcd	Nexapvd	Nexarinv
Nexapch	Nexapvnd	Nexarmenus
Nexapmenus	Nexarc	Nexarsls
Nexaptd	Nexarcus	Nexartemp
Nexaptemp	Nexargs	Nexctrl
Nexarhst	Nexenv	Nexperms

This way all of any wildcard group can be treated together, for example: The "ap" databases can be referred to at one time (nexap*), or all of the "ar" databases (nexar*)... ALL databases starting with "nex" can be moved to another system (nex*). This naming convention is not mandatory, it is only a helpful convention.

Qualifiers

Description:

Qualifiers allow you to develop and run the same application for up to 161 separate sets of data. For example, if you managed an accounting firm that performed payroll operations for other companies, you could develop a payroll program and use it on several separate sets of data. The separate sets of data would be qualified versions of the file. (It is possible to manipulate more than 161 sets of data, but that is the current limitation of the qualifier editor.)

With qualified files, there will be copies of the key segment, data segment and of each automatic index for every qualified version of the file.

Advantages

In this case, you want the data to remain totally separate. You never want to see data from company #1 together with company #2.

As far as the program is concerned, you only have one copy of the file definition, screens, formats, and processing tables. This will save you time in maintaining your systems since you will only have to make the change in one place on your system for all of your customers.

Offers additional flexibility and simplifies programming when dealing with large corporate systems.

Allows you to keep databases smaller and segregated thus improving performance and avoiding a complete (corporate) shutdown when a single division's data becomes corrupted.

Creating Qualified Sets

Use "Define Files" and when asked to "Create Qualifiers (Y/N):" - Answer "Y".

The Edit Qualifiers table appears. Enter the specific qualifiers. The qualifier is a code to differentiate the separate sets of data.

Limits:

7 letters and/or numbers to a qualifier name (Unix and WIN95/98/NT)

3 letters and/or numbers (Windows)

Suppose you enter two qualifiers "act" and "bb", there will now be three separate sets of data, the unqualified version ("key") and two qualified versions ("keyaa" and "keybb").

Accessing Qualified Sets

To access the unqualified file use Inquire, Update, Add or Request Output as normal. To access either qualified file, use Inquire, Update, Add or Request Output with the following user menu flags:

- ?mcode uses the indicated file name qualifier
- md displays the "Enter File Name Qualifier" prompt after the "Enter File Name" prompt
- mq "message" displays a user-defined prompt for the qualifier

Important Define Files Functions

"filePro & non-filePro files"	All filePro files contain a hidden 20 byte header, Non-filePro files are structured ASCII files with no hidden header per record, therefore, no system maintained fields.
Fields	The real fields of any file.
Data types & Edits	Keep data clean and organized.
Associated fields	Unique feature of filePro. Group fields for searches and comparisons
System Maintained Fields	Record created by, creation date, updated by, updated date, etc.
Qualifiers	Unique to filePro and allowing multiple data sets with the same processing, screens and outputs.
Password	Security feature for locking the creation side of the menu on any file.

Expert Level

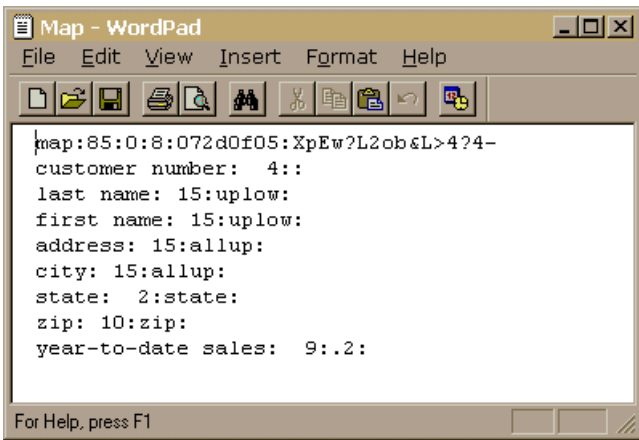
Access Qualified Sets in Processing

When defining processing, if you are standing in a qualified set of a file and do a lookup into a second file, the program will look for a qualified set of the same name in the second file. You can lookup data from a non-qualified set from a qualified set and vice-versa using the LOOKUP flags properly.

Make sure to use the LOOKUP filename@ when standing in a qualified set to access the non-qualified set. You also have the environment variable " PFQUAL " available to you, so this could be used to your advantage in conjunction with the " GETENV " command.

The "map" file

When you enter information in the "Define Files", filePro creates a definition layout called "map". If you look at the "map" for "FPCUST" with an editor, you will see something similar to the following:



The first line of the map file is as follows;

```
Map:xxxyy:zz:XXXXXXXXX:PPPPPPPPPPPP
```

Where:

- Map is the literal "map".
- xxx is the record length of the key (excluding the 20 byte header).
- yyy is the record length of the data file.

XXXXXXXXXX is the 8 digit hexadecimal checksum of the encoded password.

PPPPPPPPPPPPPPPP is the encoded password.

The second and subsequent lines contain references to each field by name, length and data type. As you can see, the references are separated by a ":" colon.

Dates are stored as the number of days since January 1, 1983.

User IDs are stored as the user ID number from the /etc/passwd file (UNIX/XENIX) or as "0" (Windows)

Password - Keep in mind that a browse window will not display fields in the current file if the field is not included on the screen when the filePro file has a creation password.

v6.1 Added logging to ddefine.

ddefine can now optionally track changes made to filePro file layouts. This includes the name of the file, who changed it, and what fields were changed. Requires a logging configuration file to be added under the /fp/logs directory named 'ddefine.cfg'. Format of the config file is the same as the servlog.cfg file that comes shipped with filePro.

Example ddefine.cfg:

```
ROLLING,DEBUG,ddefine.log,60000
```

Define Screens - Option 2

Contents of this section

Description

Using Define Screen Help

Copying a Screen

Adding Fields

Toggle Graphics

Box Functions

Adding Color

Resolving Fields

Advanced Section

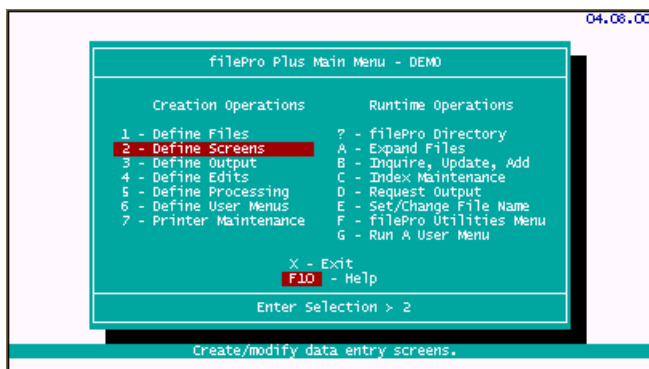
Description:

Use this program to create Input Screens (subsequently referred to as "Screens") for use in adding and changing record information in a filePro file. Input screens are often referred to as "forms" in other programming languages. An almost unlimited number of screens can be created for each filePro file. Multiple screens can be created when you have more data in each record than can fit on a single screen or to control input functions. Suppose you are developing an "Accounts Payable" application. You will probably want separate screens for entering payments and charges. Payments may be entered on screen named "pay" and charges on a screen named "chg".

Separate screens can be used to protect certain data from change or access by unauthorized users such as a "Payroll" system. Although you have a payroll entry clerk, you may not want the payroll clerk to have complete access to all employee information. This sensitive information can be maintained by a separate screen with password protection to allow access by management personnel only.

As you design screens, keep in mind that you can use system-maintained fields, "real" fields defined through the "[Define Files](#)", and variables created by "Define Processing". System maintained fields contain information 'remembered' by the system, e.g. the date a record was first created, when the record was last updated, etc. System maintained fields are built into filePro and can't be redefined.

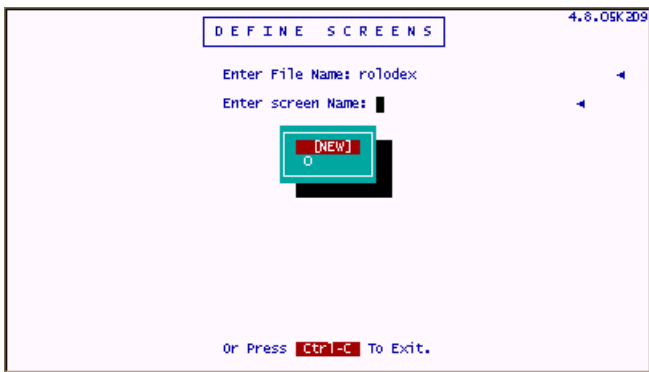
Select "Define Screens" option 2 from the filePro Main Menu.



Select a file name from the list of files by highlighting the filename and press Return.



After selecting the filename you will only see option [NEW] for a new screen and "0", the default screen 0, which was created by "Define Files" for file "Rolodex".



Notes: The screen name is limited based on the Operating System you are using. If you want to maintain WINDOWS compatibility, use screen names of three characters or less.

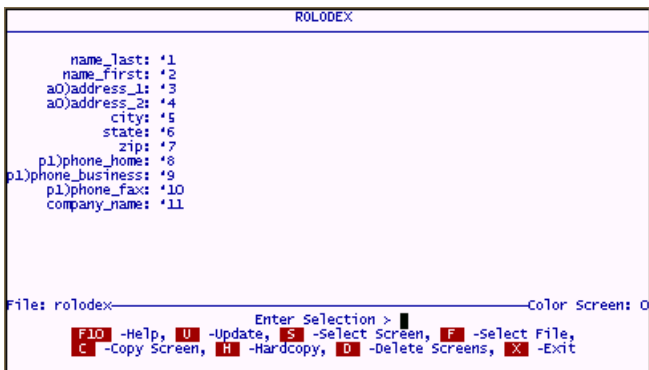
If you assigned a creation password to a file during the "Define Files" operation, you will be prompted for this password. This prevents unauthorized users from creating screens for your password-protected files.

If you press Return while [NEW] is highlighted, you can enter a new screen name. In our case, we are going to select "0" to use screen 0. This screen was the default screen created when defining the file.

Enter "0" and press Return.

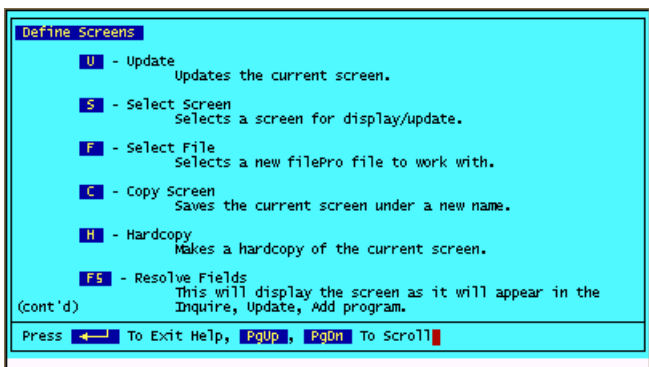


You will be presented with default screen 0 with selection options at the bottom of the screen depicted as follows;



Using Screen Help

Press F10 for help and the following "Help Screen" is presented for define screens.



The help screens provide descriptions of the selection options shown for the previous screen. When you see the option "PgDn To Scroll", there are additional help screens provided included. Press PgDn to retrieve the following screen.

```

Define Screens (cont'd)

D - Delete Screens
A list of all existing screens for this file will be
displayed. Position to the screens you want to
delete. When the name is highlighted, press ←.
An asterisk will appear next to the name.
Pressing ← again removes the asterisk.

Once you have selected all the screens, press ESC.
The computer will then ask for confirmation that the
list of screens to delete is correct.

M - Monochrome attribute editor (Color systems only)
Screens that are defined in color may not look good on a
monochrome monitor. The monochrome attribute editor will
allow you to define how the screen should appear on such
systems, rather than letting filePro decide for you. More
detailed help is available from within the editor.

Press ← To Exit Help, PgUp, PgDn To Scroll Last page

```

This is the last page of help for the selections as indicated in the lower right corner of the screen. Press Return To Exit Help and return to "rolodex" screen 0.

```

ROLODEX

name_last: '1
name_first: '2
a0)address_1: '3
a0)address_2: '4
city: '5
state: '6
zip: '7
p1)phone_home: '8
p1)phone_business: '9
p1)phone_fax: '10
company_name: '11

File: rolodex-----Color Screen: 0
Enter Selection > |
F10 -Help, U -Update, S -Select Screen, F -Select File,
C -Copy Screen, H -Hardcopy, D -Delete Screens, X -Exit

```

Press "U" to update screen "0"

```

ROLODEX

name_last: '1
name_first: '2
a0)address_1: '3
a0)address_2: '4
city: '5
state: '6
zip: '7
p1)phone_home: '8
p1)phone_business: '9
p1)phone_fax: '10
company_name: '11

File: rolodex-----Color Screen: 0
F9 -Toggle Graphics, F6 -Display Fields, F5 -Resolve Fields
F10 -Help, F7 -Box Func's, F8 - Extended Func's
1, 1 Ctrl F10 -Color Help, ESC -Record, Ctrl-C -Cancel

```

Notice that the options have changed at the bottom of the screen. Help is provided for the screen options, so press F10 to retrieve screen help.

```

Placing text literals on the screen
Simply move the cursor where you want the text to appear and type.

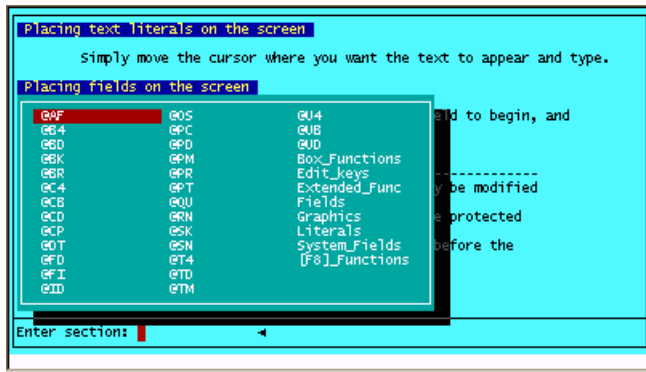
Placing fields on the screen
Position the cursor to where you want the field to begin, and
type one of the following field indicators:

Indicator      Meaning
-----
'              The contents of the field may be modified
               by the user.
!              The contents of the field are protected
               from modification.
x              The field must contain data before the
               record can be recorded.

(cont'd)
← - Exit, ↑ PgUp PgDn - Scroll, F9 - Search

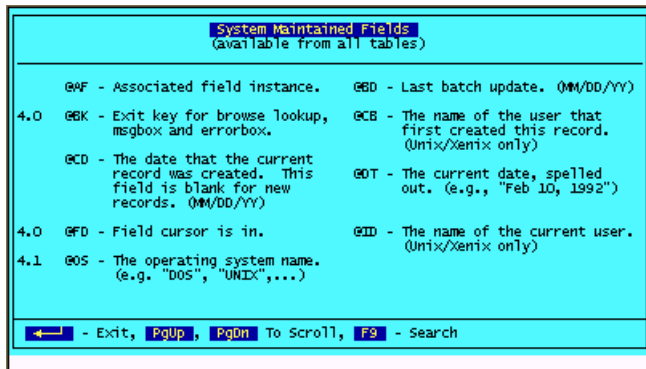
```

Notice that the help provides an additional option "F9 - Search" to search using indexed help. This feature was added in filePro version 4.5 to allow you to quickly access help on specific functions and topics based on your current selection options. Press F9, to get a list of available indexed help for this section of "Define Screens".



Generally, help follows standards which uses upper and lower case. Those items in all uppercase are filePro functions, System-Maintained-Fields or commands, whereas items in upper and lower case are HOW-TO or general topics.

Example - selecting @CD (creation date) will go to a description of the system-maintained field @CD and other system-maintained fields. When selecting "Fields", you are placed in the section which describes how to place fields, including @CD, on a screen. Highlight @CD and press Return to get the following screen.



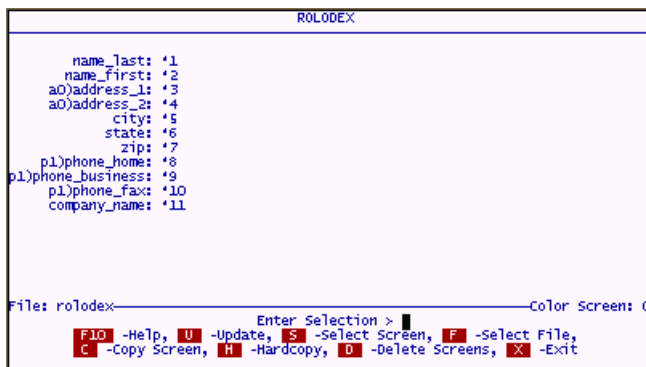
When pressing F9 again and highlighting "Fields", you are returned to the original help screen when starting discussion on this sub-topic.

Press F9 and highlight "Fields"



Copying a Screen

Press Return to exit help and return "Rolodex" Screen 0.



Although you could use Screen 0 to enter data for the Rolodex, any changes made to screen 0 will be potentially overwritten when redefining the file. For this reason, it is a good practice to copy the default Screen to another screen name. This will ensure that any changes we make to the screen are preserved.

Press C to copy the screen.


```

ROLODEX

name_last: *1
name_first: *2
a0address_1: *3
a0address_2: *4
city: *5
state: *6
zip: *7
p1phone_home: *8
p1phone_business: *9
p1phone_fax: *10
company_name: *11

[NEW]
0
File: rolodex- Color Screen: 0
Copy to...
Enter screen name:

```

Since we will be copying Screen 0 to a new screen, highlight [NEW] and press Return. At the screen name prompt, enter 1 (meaning copy to screen 1) and press Return.

```

ROLODEX

name_last: *1
name_first: *2
a0address_1: *3
a0address_2: *4
city: *5
state: *6
zip: *7
p1phone_home: *8
p1phone_business: *9
p1phone_fax: *10
company_name: *11

File: rolodex- Color Screen: 1
Enter Selection >
[F10] -Help, [U] -Update, [S] -Select Screen, [F] -Select File,
[C] -Copy Screen, [H] -Hardcopy, [D] -Delete Screens, [X] -Exit

```

Adding Fields

Notice how the screen name has changed in the lower right hand corner to Screen: 1. Press "U" to update the screen. Use your arrow keys to go to row "18" column "4". Use the cursor position number in the lower left corner of the screen to get to the starting position. Type in a literal "Date Record Created: ". Use your right arrow key to go right two positions and type "!!@cd" for the system field "Creation Date". Press the TAB key twice and type "Last Updated: !@U4". Your screen should look similar to the following screen.

```

ROLODEX

name_last: *1
name_first: *2
a0address_1: *3
a0address_2: *4
city: *5
state: *6
zip: *7
p1phone_home: *8
p1phone_business: *9
p1phone_fax: *10
company_name: *11

Date Record Created: !@cd      Last Updated: !@U4

File: rolodex- Color Screen: 1
[F9] -Toggle Graphics, [F6] -Display Fields, [F5] -Resolve Fields
[F10] -Help, [F7] -Box Func's, [F8] -Extended Func's
19, 1 [Ctrl F10] -Color Help, [ESC] -Record, [Ctrl-C] -Cancel

```

We have just added two system-maintained-fields. The exclamation point "!", preceding the system-maintained fields, is a "field indicator" and has the meaning "the field is protected from update". Notice that the other fields are prefixed with an asterisk "*". The "*" field indicator means that the contents of the field can be optionally modified. There is another field indicator, the "%", and meaning that the field is a "must fill" or mandatory field. Lets change the screen to make the name_last, name_first, address_1, city, state, ZIP mandatory by replacing the "*" with a "%" field indicator. Your screen should look similar to the following after the changes.

Toggle Graphics

When pressing the F9 key notice that the word "Graphics" is displayed in the lower right and corner of the screen.

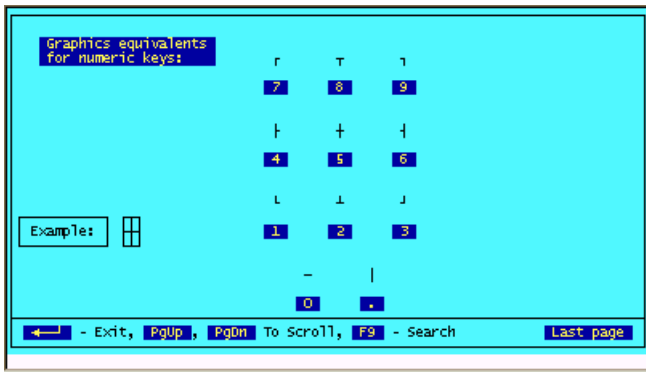
```

Last Updated: !@U4

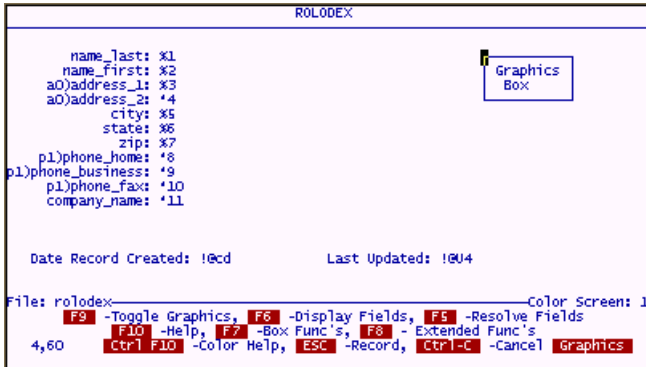
Color Screen: 1
Display Fields, [F5] -Resolve Fields
Func's, [F8] -Extended Func's
[ESC] -Record, [Ctrl-C] -Cancel Graphics

```

When pressing the F9 key again, the word "Graphics" disappears. This toggle feature allows you to change to the mode of the number keypad on your keyboard. (On single-user systems, the number-lock must be on.) When "Graphics" is displayed, the number keypad will place graphics characters on your screen instead of numbers. By pressing the help key and then pressing F9 for index search, select the help for "Graphics". You should see a screen that looks similar to the following and shows the relationship of the number keys and graphics characters.



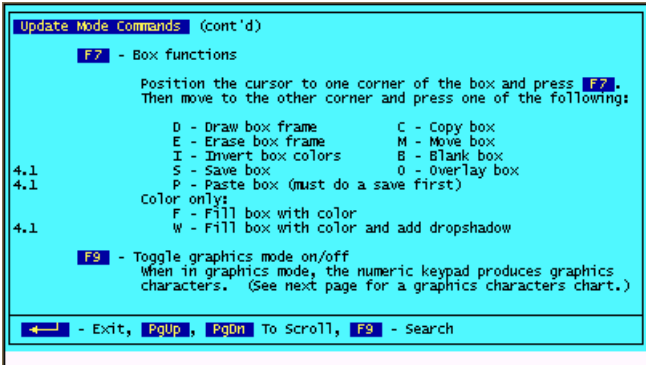
Place your cursor in a blank area on the screen (4,60) and draw a simple box by pressing the applicable keys on the number keypad and type a literal in the box "Graphics Box" as depicted in the following screen.



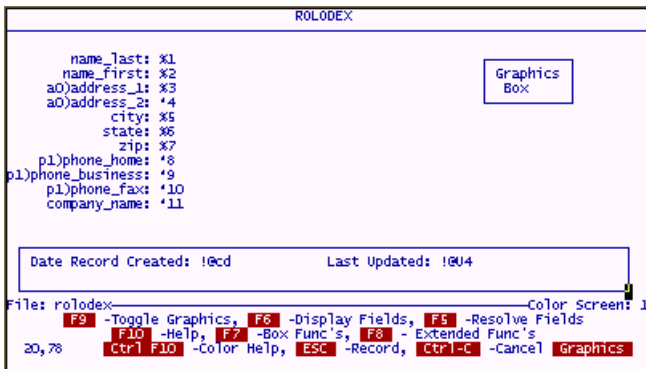
Box Functions - F7

Use of the F9 key and number keypad was the only way that you could enter graphics in older versions of filePro. This feature is useful when you want to repair a graphics box but a complete set of "BoxFunctions" are provided with later versions of filePro when pressing the F7 key. To get a description of the box functions, do the following.

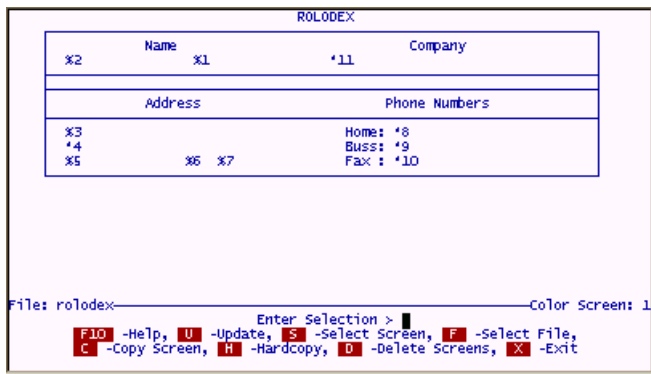
Press the F10 - Help key, F9 for search and select Box_functions help as follows.



As you can see in the previous help screen, you have a range of options for manipulating the information while designing your screen with the "Box functions". To demonstrate how easy it is to use this function, draw a box around the system-maintained-fields that you previously added near the bottom of the screen. Place you cursor at position (17, 2) of your screen and press the F7 key. This will place a plus sign as an anchor for one corner of the box. Use the cursor keys to move right and down to position (20, 78) and press "D" to draw a box. Your screen should look similar to the following screen when done.



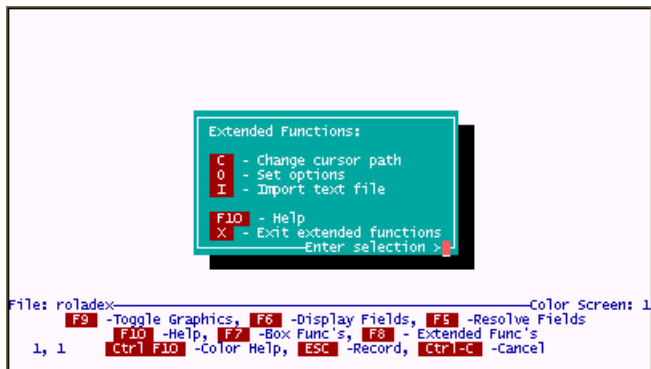
Try erasing the box by repeating the above procedure but instead of pressing "D" to draw press "E" to erase. Try drawing boxes around in other areas of the screen and move the information using "M". After you are done experimenting, press Ctrl-C to cancel the changes that you have made. If you inadvertently save the changes to screen 1, return to the "Copy" function, select screen 0, and re-copy screen "0" to screen "1". Change screen 1 to look like the following screen and preserve your screen by pressing ESC to record the changes.



Change Cursor Path

After moving the information around on the screen, the fields are now in a different order. If you used the screen for inputting data without making changes to the cursor path, (the way the cursor moves between fields) the cursor would jump all over the screen based on the field number order e.g. field 1 (name_last), field 2 (name_first), field 3 (address_1) etc. since the default cursor path follows the sequence of fields as they were defined and placed on default screen 0. You may want to print a hardcopy of the screen 1 to refer to the information at this point.

We can define a new cursor path by using the Extended options. Press "U" to update the screen, and press F8 for "Extended Functions". When pressing the F8 key for extended functions, the following screen is presented.

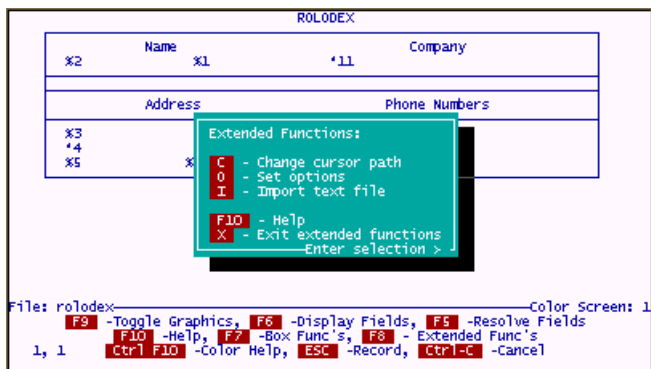


Press "C" to change the cursor path. You will see the original cursor path based on the sequence of the fields in the map. This cursor path was established when the default screen "0" was created during "Define Files". When in the cursor path screen, highlight field "1" and press the F4 key to delete "1" as the first field in the cursor path. Highlight field "3" and press F3 to insert a blank after "2". You can use the F3 and F4 to insert and delete respectively while editing the cursor path. Insert TABS by pressing the TAB key. Change the cursor path to agree with the following screen and press ESC to save the new path.

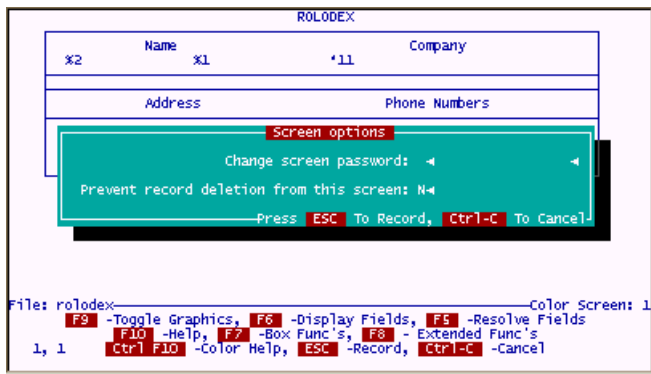
Note: The cursor-path program doesn't accept fields that are not used on the screen, protected fields, or undefined fields. Instead, it gives you an error message and returns you to the cursor-path screen so that you can correct the mistake. However, it does accept dummy fields created by " Define Processing ". There are no dummy fields included in this example.

Other Extended Functions

While in update mode, let's take a quick look at the other "Extended Functions" by pressing the F8 key. You should see the extended options selection menu once again as follows.



Select "O - Set options" to retrieve the following screen.

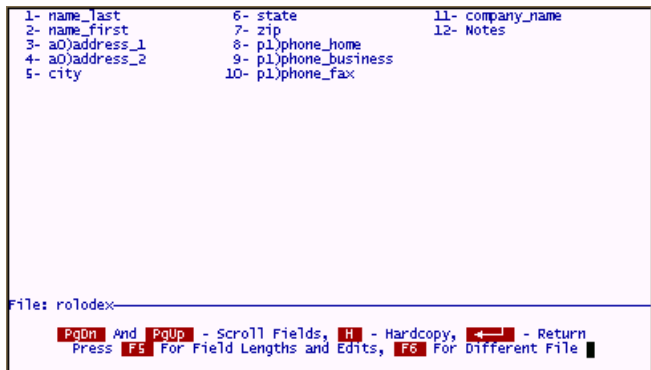


This extended function allows you to add a password for your screen and prevent record deletion when using this screen.

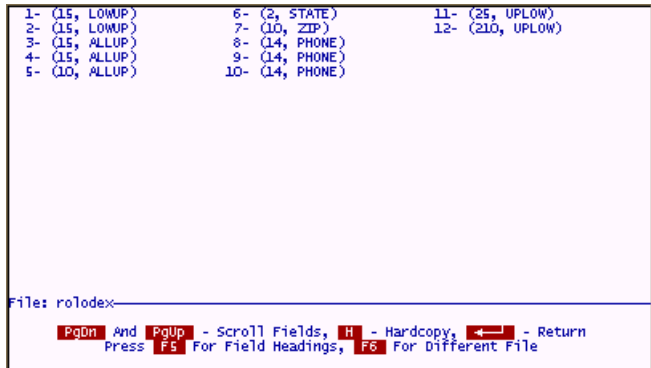
Press Ctrl C to cancel this option and to return to update mode. Press F8 again and select "I" for "Import text file". This feature allows you enter a path and file name to import text directly into your screen. Make sure to place the cursor in a blank area on your screen prior to importing text. Press Ctrl C to cancel this option and return update mode.

Display Fields

Press F6 to display fields previously defined for the "Rolodex" file.



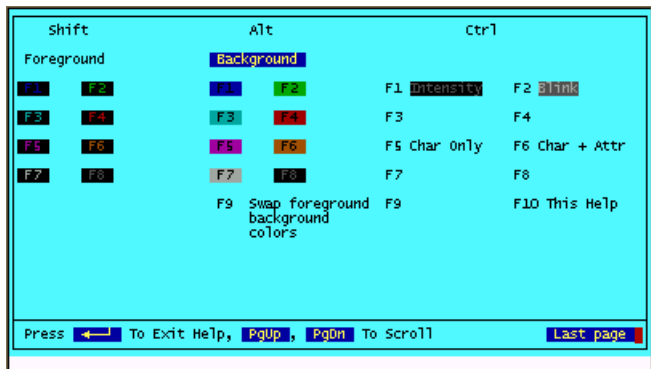
When pressing F5, field lengths and edits are displayed as follows.



Press Return to go back to update screen mode.

Adding Color

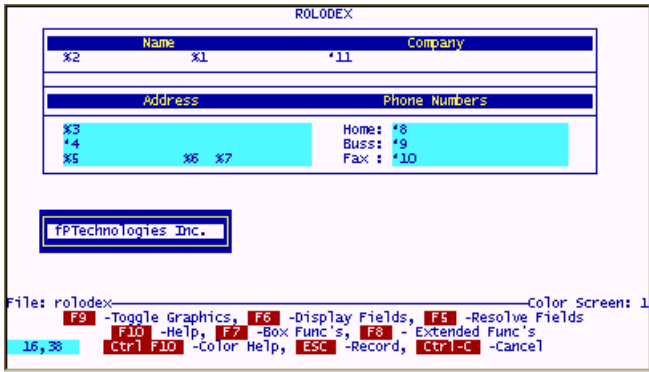
If you have color capability on your system, you may want to spice up your screens to keep them interesting to the users by highlighting data entry areas and titles. You can press Ctrl F10 to retrieve help on this topic.



By pressing combinations of Shift, Alt & Ctrl with a function key, you can set color combinations to perform highlighting on your screens. Return to screen update mode and try setting various combinations of foreground and background colors. When pressing the key combinations you will notice that the color combination you have selected will show up

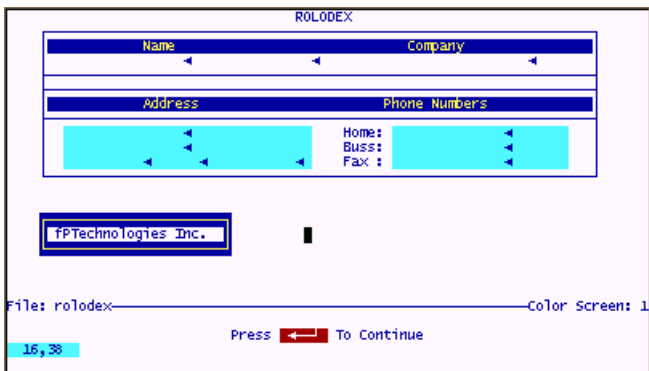
in the lower left-hand corner of the screen. Try changing screen "1" to highlight the literals and draw a colored box around your name as depicted in the following screen.

After selecting your color combination, use the F7 key with "F" to fill color, much like you used it previously to draw boxes. In fact, you will use the "D" to draw the colored box around your name.



Resolve Fields

After completing the screen color changes, press F5 key to show how the screen will look to the users when using IUA. You should see a screen similar to the following when pressing F5. Press Return to continue updating your screen. Modify the screen colors, boxes etc. until you have a color combination that you like. Go to IUA and test your screen design.



When exiting "Define Screens", if you assigned a creation password, you will be prompted to change password or to press Return.

Note: See Technical notes on "Blinking" attribute.

Advanced Design - Define Screens

Field Indicators

The following field indicators can be used on screens to control data input.

- * Accepts data.
- ! Protected ---no data can be typed in.
- % Must-fill ---there must be data in this field before the record can be recorded.
- \ Truncates field.

Use the TAB key to move quickly within a line. This moves the cursor right eight characters at a time. Make sure that you leave enough room for each field. "Enough room" is the length of the field plus one. An end-marker will appear in that last space. (End-markers on single-user systems are usually small triangles; on multi-user systems, periods or small boxes.) In other words, if field 1 is 15 characters long, leave 16 spaces counting from the field indicator:

Name_last: %1

If you leave only 15, the first character of the next field is lost. You may also wish to leave a second space for visual separation between the fields. Use F5 to check for visual separation.

Aesthetics: Lines, borders, boxes and screen headings have two purposes. They make screens:

- Attractive
- easy to use

Use subheads and borders to separate screens into logical sections. Think about the organization of your file --- does the information fall into two or three parts (e.g., name-and-address information vs. account information)? If so, consider visually breaking the screen into two parts with a line or box.

Reverse video and graphics. Do you want instructions to the user on the screen? Consider doing them in reverse video--- dark letters on a light background instead of light letters on a dark background. For reverse video, press ALT F9 to return to normal mode, press ALT F9 again. When reverse is on, the cursor-position box is also in reverse.

For graphic character mode, press F9 to return to normal mode, press F9 again. When graphic mode is on, the word Graphics appears at the lower right corner of the screen. The graphic characters themselves are accessed with the numeric keypad at the right of the keyboard. (On single-user systems, the number-lock must be on.) To remember which keys

access which characters, think of the keypad as a box with lines running through its center. The corners of the keypad are the "corner" graphics characters.

Use the F7 and F8 keys to draw lines and boxes quickly. To draw a box, you simply pick two diagonal corners of the area you want to enclose; press F7 at the two points. To draw a line, press F7 at the starting and stopping points. For vertical lines, put the two points within the same column on the screen. For horizontal lines, put the points on the same line. To erase a box, press F8 at any two diagonal corners. To erase a line or portion of a line, press F8 at the points where you want to start and stop the erasure. For reverse boxes, go into reverse mode and press F7 at two corner points. This creates a narrow black box within a highlighted box. For a highlight-only box, create a reverse box, then erase it with F8. The highlighted portion of the box will remain on the screen.

Note: All graphic and reverse characters can be overwritten and erased, reproduced and moved using the regular function keys.

Check List: Look over the format for errors, using this checklist:

- a. Are there any spelling mistakes in the headings?
- b. Does every field have a field indicator? Use only protected-field indicators for system-maintained fields.
- c. Did you leave enough space for each field?
- d. If you have more than one field on a line, is there at least one blank space between the end of one field and the beginning of the next, and at the right edge of the screen?

Use F5 to "resolve" (analyze) the screen. Check that fields don't overlap or run into each other ---look for each field's end-marker. If you have forgotten a field indicator, the field number appears; if you put a nonexistent field on the screen, the field number and indicator appear. If you've typed the wrong field number, the field length will look too long or too short. Only dummy and system-maintained fields do not change (see "Defining Processing" for information on dummy fields). After noting which fields need to be corrected, press Return to go back to the format screen.

You may want to pre-set the movement of the cursor through the record. For instance, rather than having the cursor move from left to right, top to bottom through the screen, as it usually does, you can have it go up and down through columns. If the screen is split up into sections, you may want to set tabs that let you jump from one to another quickly.

Note: If you've restructured the file or revised the screen, a message saying that the cursor path is invalid may appear.

How to Revise Screens

The only difference between creating and revising a screen is that, when you change field numbers, you invalidate the cursor path. If you see a message saying that the cursor path is invalid, you should revise the path to match the new format.

Color on Screens

Color can be added to screens created in the Windows versions of filePro. If you add color using the Windows version, the color version of the screens can be transferred to non-WINDOWS versions (even though you can not create colored screens in non-WINDOWS versions) and used if your terminals or workstations are capable of displaying color screens.

If your IBM PC/XT has a color monitor and adapter card you can set both background and foreground colors for the entire filePro program. Use the "Set Color" utility to modify the default screen colors.

Some Extra Considerations

Avoid including Screen 0 (created by "Define Files") in your finished application design. If you modify this screen and later, by accident, ask Define Files to build a screen for you, it will wipe out any changes or modifications you have made to Screen 0 and build you a fresh screen. For this reason, it is sometimes helpful to "Copy" Screen 0 to another screen, usually Screen 1. Then, delete Screen 0 and continue modifying and enhancing Screen 1.

Cursor Paths: If you have eliminated fields from a screen that already has a cursor path, the program may warn you that the cursor path is no longer valid. If you forget to put a field on the cursor path chart, the program adds it to the end of the list automatically. Unless that is where you want it, you should go back to Define Screens and add it to the cursor path chart at the appropriate spot. Use the F3 and F4 keys to open and close spaces.

When you update records in Inquire, Update, Add, the program refers to the cursor-path chart as you move from one field to the next. If you have no chart, the program uses its own cursor-movement rules.

The up and down arrow-key rules have particular significance because the keys will seem to skip fields if indicators aren't aligned. The rules are:

When you press the down key, the cursor goes to the first character of the field just below it; if it finds no first character, it goes to the first character of the field to the right of its previous position.

When you press the up key, the cursor goes to the first character of the field just above it; if it finds no first character, it goes to the first character of the field above and to the left of its previous position.

For example, say that this is your format:

Company: * 1 Contact: *2

Street: * 3 City: *4

If you update field 1 and press the down key, the cursor jumps to field 4, not field 3. If you're in field 4 and press the up key, the cursor jumps to field 1, not field 2.

To get the cursor to move straight down when you press the down key and straight up when you press the up key, simply align the field indicators.

It is possible to send the operator from screen to screen during data input. See SCREEN in "Defining processing."

filePro Style Headings: If you want your headings to match the ones applied with filePro, use all uppercase letters, separate each letter from the next with a space (three spaces between words), center the heading, and use an underline.

Concatenating fields on a screen:

There may be instances when you want to combine fields like name_last, name_first and middle_initial on the screen so that a lookup name is combined e.g. Smith, James, G. to be displayed as James G. Smith. This can be easily done by creating a dummy field that combines the 3 fields and concatenates the fields leaving a single space between name_first,

middle initial and name_last.

Create the dummy field in automatic or input processing to hold the value of the displayed field as follows;

Then: n(32)=2<3 {"."<1

where "n" is defined as name with a length of 32 characters (15 characters for Name_last & name_first, and 2 characters for middle_initial with the "." period), field 2 = name_first, field 3 = middle_initial and field 1 = name_last. The "<" pushes a field left leaving a single space between values and "{" pushes left and leaves no space between the values. The literal "." is placed tight up against the middle_initial.

Place the dummy field on your screen using "!n".

Concatenated fields are normally used for display purposes only and therefore should be treated as protected fields by using the "!" field indicator. If you have a need to update the fields, keep them separate as defined in your file.

Note: You can put arrays of dummy fields on screens. See " Defining Processing " for more information on arrays and dummy fields.

Truncating Fields:

Use the slash "/" field indicator to truncate fields on a screen. This will establish a limit to prevent wrapping of the fields to the next line when the field length has not been defined.

New in Version 5.8.02 and higher

To implement a scrolling field, place a field as you normally do, but then place a backslash ("\") at the location you want to truncate the visible part of the field. Then, in *clerk, the field will only display as wide as the place you specified with the "\". However, when you are in the field, you can scroll horizontally. The EOF (end of field) designator will now also show a > indicating additional characters for a field when viewing a record.

Screen Limits:

Width and depth: 80 characters across, 20 lines down.

Size per screen format: up to 2,270 bytes (characters).

Maximum number of fields per screen: 200.

Maximum number of tabs per cursor path: 20.

Tricks:

Apply highlights to field contents by highlighting the field indicator preceding the field.

Remove end-markers by making the foreground and background colors the same at the end-marker position on the screen. Make sure that you use the screen background color to hide the end-markers.

Centering - hidden function under (M)ove box function. You can press (h) or (v) to center the boxed area horizontally or vertically.

Use Ctrl and F7 when in update mode to get the current color settings for an area on the screen. This is useful to duplicate existing color areas in other areas on the screen.

For portability to WINDOWS, it is important to keep the screen names to 3 characters or less. Otherwise, screens can be up to 7 characters long under Unix, and 25 characters in WIN95/WIN98/NT.

Define Output - Option 3

Version 6.0.00 enhancement

from the F8 – Options in Define Output, you can select F – Hide or Show Forms. This will allow you to manage the .out files that are hidden from clerk. A list of .out files will appear when you press F. This list will display showing .out file hidden with an asterisk and those not hidden unmarked. You can manually toggle the .out files or you can press F7 to reverse toggle them all. Once you have the list the way you wish it to be, press SAVE to process the new settings for this one directory.

Also allows you to switch just the .out file that you have selected by selecting F8 – Options. The line "Hide forms from clerk N" determines rather this form is hidden or not from clerk (IUA).

Contents of this section

Features

Kind of Output

Formats

Forms

Labels

Reports

Define Processing Only Format

Grouping [Subtotal & grand total breaks]

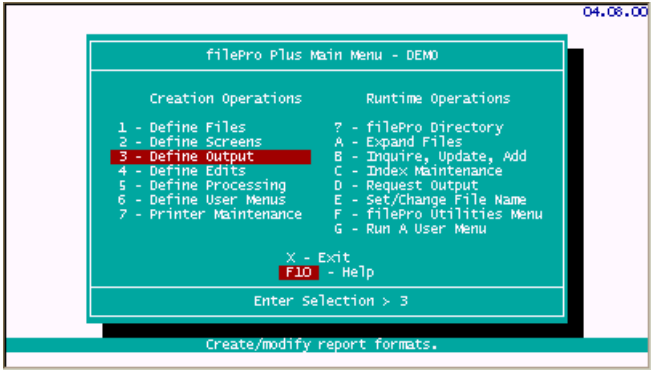
Sorting

Extended Functions

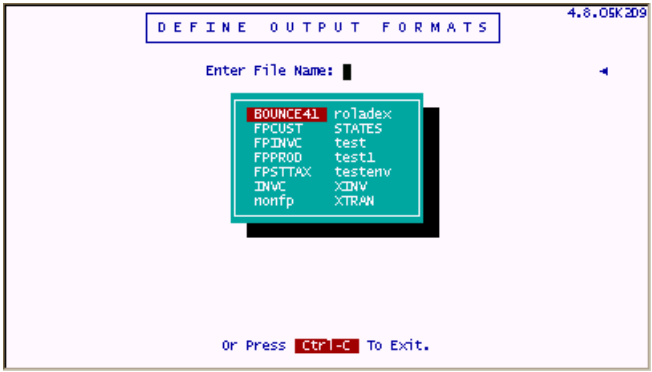
Features:

The "Define Output" option is used to create printed output and process only formats. Any output you define with this option can have output processing associated with it to do things like; printing a field value from other files; posting field values from the current file to other files; posting calculated data to other files. Output formats can contain real fields, system maintained fields and dummy fields as well as any literal text for things like 'Report Titles', 'Field Headings' etc. The presentation of the printed output is controlled through the use of print codes to change fonts, line spacing and other characteristics of the output.

Select Option 3



You will see a list of filePro files from which you can choose.

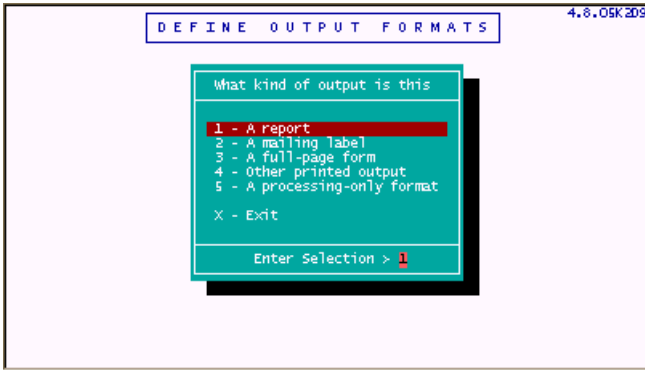


After selecting the filePro file from the list, you can select an existing output format, if you want to modify the format, or create a new format by selecting [NEW].

When selecting [NEW] from the listbox, you can enter the "Output Format" name of your choice. In this case, we will create a new list output format by typing "list".



After entering a format name ("list" in this example) and pressing enter, you are presented with various options for defining your output format as follows;



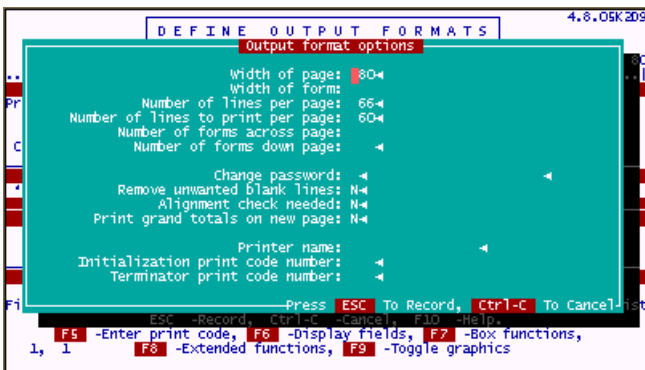
Kind of Output:

- Option 1 A report is the standard output format that provides three sections including;
 - Heading/Title Lines;
 - Data Lines;
 - Total/Sub-total Lines
- Option 2 A mailing label provides for printing several labels across the page.
- Option 3 A full-page form is typically used when printing a single page for each record.
- Option 4 Other printed output is similar to option 1 but a catchall option.
- Option 5 A processing only format is used when you do not want a printed output (normally used for posting to lookup files).

The above options give you a lot of flexibility to rapidly design any output you will ever need.

A Report:

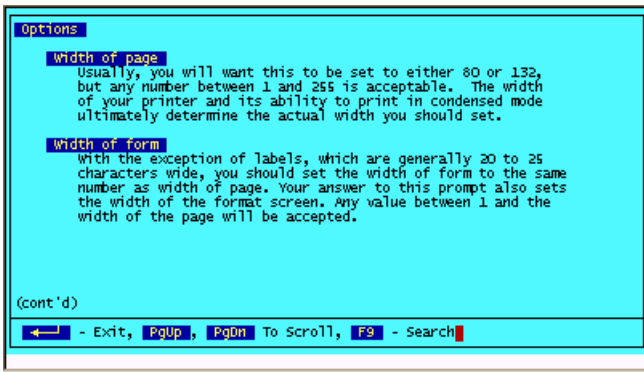
When selecting option 1 for "A report", you are presented with a screen to set the options for your output as follows;



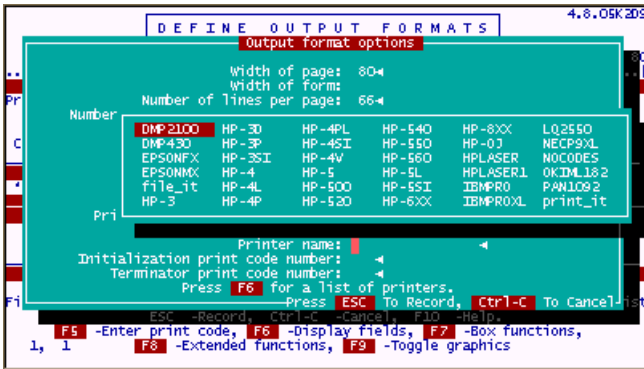
The "Output format options" are self-explanatory for standard output formats, but it is important to consider the purpose and controls you wish to invoke when printing your reports. For example, if you include payroll check printing in a system, you will probably want your checks to always go to a specific printer where your checks are pre-loaded and provide for printer alignment checking. A "check printing" format would probably also have password protection whereas most other printed output would not require that level of control. So for "Check Printing", you would want to enter a "Y" at "Change password", change the "N" to a "Y" at "Alignment check needed", and specify your check printer in the "Printer Name" option. Note that the "Printer Name" is assigned through the "Printer Maintenance" option.

Getting Help

Although a help key option is not displayed as an option in the above screen, when pressing [F10] you can get help and recommended settings for each option. An example is provided as follows.



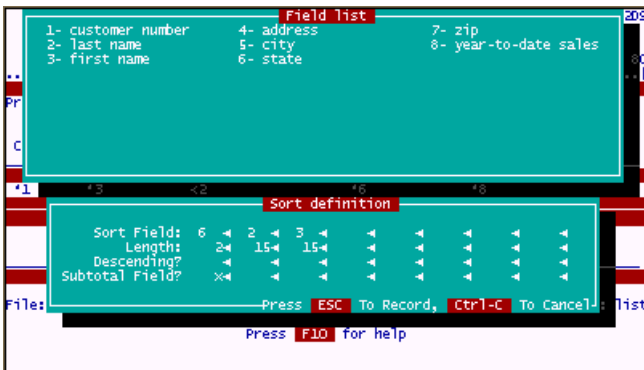
For options "Printer name", "Initialization print code" and "Terminator print code number", you can press the [F6] to lookup the predefined values, highlight the value you want and press "Enter" to plug in to the options. The following is an example for the "Printer Name" option.



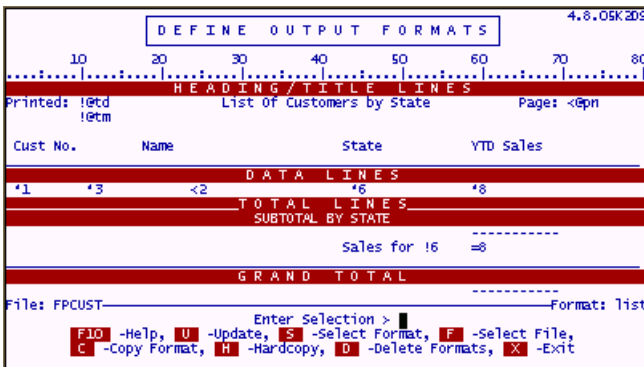
As with most other filePro options, there is always more than just one way to accomplish your objective. You can leave the "Printer name" option blank, and use a "Request Output menu flag" instead. This is even true for the password option since you can prompt a user for a password through processing by using the command "INPUTPW" or "INPUTPW POPUP". In fact, using the processing commands is probably a better method if you have a multi-user system for a large office and want to provide system administration features with your system design. This will prevent having to change the system when the payroll clerk leaves the company for another job. You could do this by creating a password file that is accessed for sensitive data (like payroll) so that password(s) can be periodically changed by the office administrator instead of a single password controlled through the output format. This would be a better method if you are distributing the system to more than one company.

Sort Fields

When defining a report, you can specify the order of data or sort fields to be used. For our list, we have selected the "last name" and "first name" as our sort fields.



The default sort is in "ascending" order, but typing an "X" in the "Sort Definition" for your fields, you can sort in "descending" order. You can also identify what fields you want a sub-total break to occur on by typing a "X" or "F" (Form feed) in the Subtotal Field row. In the above screen, there is a subtotal on field "6" or "state" to calculate the total sales by state. The customer list is further sorted by name within each state. A typical report definition might look similar to the following.



Field Indicators

The following field indicators can be used on reports.

- * Data printed starting at this position.
- ! Same as asterisk.
- < Push left ---- data is moved left leaving one empty space between fields.
- = Subtotal or total this field (subtotal and total lines only)
- \ Truncates field.

Note that an equal sign is used as the field indicator inside subtotal and data areas to obtain the appropriate total for any numeric field. Also note that system maintained fields are used on this example, i.e., @td (today's date), @tm (current time), @pn (page number). These will be replaced with their proper values when printing or viewing the output. Note finally, that the "push left" operator (<) can also be used on report formats. It is used here to push the page number one space away from the literal "Page:" and to push the customer's last name to the left.

Sample Output: The sample definition yields the following results.

Cust No.	Name	State	YTD Sales
7008	John Booker	CA	333.00
1002	Harry Gold	CA	560.00
4011	Mary Smart	CA	200.00
Sales for CA			1093.00
9001	Larry Last	CT	100.00
Sales for CT			100.00
5012	Jane Jamerson	DE	434.00
Sales for DE			434.00
1004	Brian Bell	FL	256.76
Sales for FL			256.76

Press any key to continue

Sample Output: Last page

1005	Able Baker	KS	324.00
Sales for KS			324.00
1003	Sam Davis	NY	207.69
6010	Harry Harrison	NY	950.00
4001	Howard Jones	NY	434.00
1001	Bob Smith	NY	434.84
Sales for NY			2026.53
8009	Mary George	OH	222.00
5029	Larry Smith	OH	25.00
Sales for OH			247.00
6002	Sam Spater	TX	566.00
Sales for TX			566.00
Total Sales			5047.29

Press any key to continue

IMPORTANT: Processing tables can be associated with output formats, making use of fields, text, calculations or data from any filePro file (or other system resource). This attached processing is automatically given the same name as the output format (be it a report, a form, or a label). You can choose to run an output format with a "different" processing table. This is done by using the -z option flag from a menu or system prompt for dreport or rreport, and placing the name of the desired processing as the argument following the -z (after a space).

Refer to output [flags](#) for additional information on use of -z and other output flags.

Example 1:

dreport filename -f formatname -z othertable -a

If there is an attached processing table to an output format and you do not wish to run it (or any other processing either) then you can put a "" after the -z option. This will let you employ the format with no processing at all.

Example 2:

dreport filename -f formatname -z ""

IMPORTANT: While in Inquire, Update & Add, the processing attached to any output format will be executed if the format is selected by pressing the (F)-Print Form option. However, if the form is called from within INPUT processing as in:

Then: form "invoice"

the attached processing will NOT be run. You must make sure that all variables and calculations you need for the output called in this manner are present on the INPUT table before this line is reached.

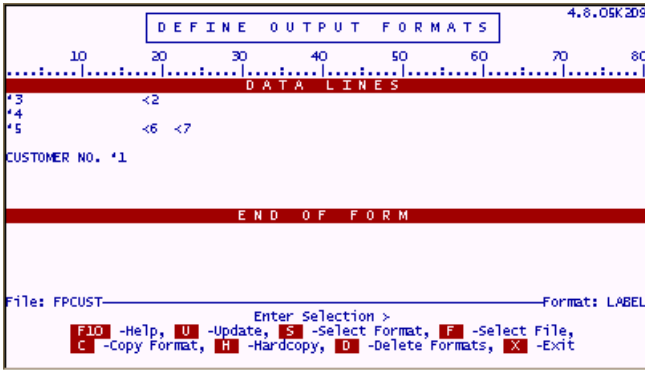
A Mailing label

This option can be used to print mailing labels, rolodex cards, index cards etc. where you do not have headings or footings, or want to print multiple forms across a page. You can also use this type of format to print from IUA using the "form" processing command or selecting a form name while IUA.

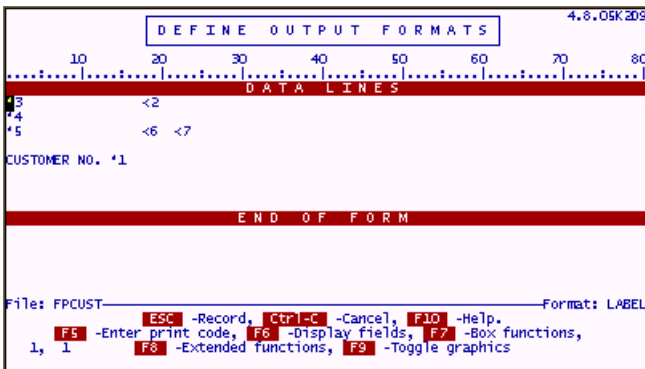
Select LABEL



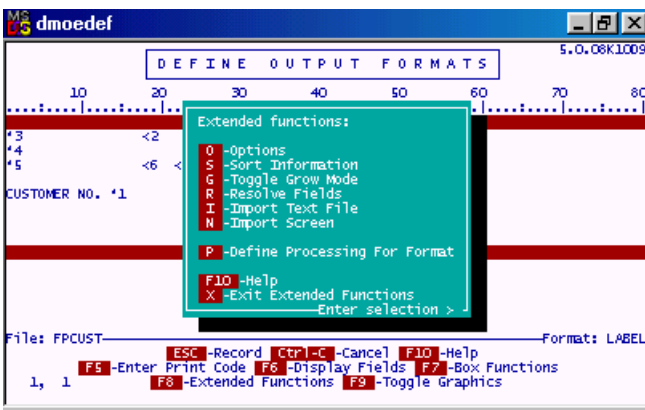
This label format has been previously defined for FPCUST.



Press [U] - update, to edit the format.



When pressing [F8], the following screen is presented.



When defining a new label or when selecting O - Options, you will see a screen to control your report format.

"Width of Page" - how wide your paper is in terms of the number of characters. The standard 8.5 inch paper is 85 characters wide when using 10 pitch (10 characters per inch). The widest that you would want to go when using standard letterhead is 80 characters to allow for margins. The maximum number of characters for 14 inch wide paper would "144". Typically you will use "132" character width of page (default) to provide a "6" character margin on each side the page. The program accepts a maximum of "255" character width of page.

"Width of Form" - With labels, the width will probably never be greater than 25 characters. If you are printing index cards with two across the page, then your width would be 40 characters. The maximum is the same as the maximum width of page i.e. "255" characters.

"Number of Lines per page" - Use the vertical measurement of your individual labels for label stock, or cards for card stock or other printing stock. Include blank lines at the top and bottom of the label in determining the number of lines to print. The number of lines to print will equal the number of lines per page for this format type. By leaving blank lines at the top and bottom, it makes it easier to position your printing stock in the printer at runtime since the top edge of the paper can be used for alignment.

"Change passwords" - Use this option to add or change your format password.

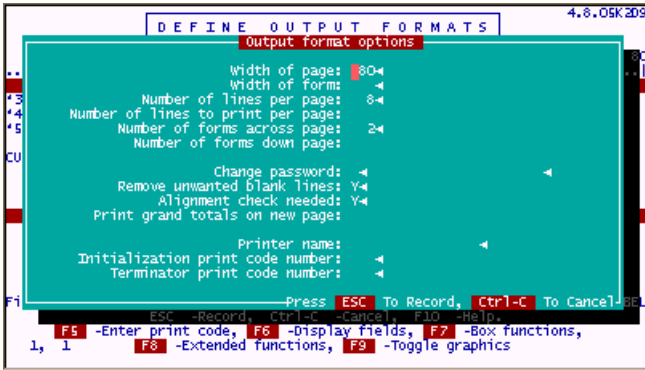
"Remove unwanted blank lines" - This will remove blank lines so is not recommended for labels or other report formats where data needs to be positioned in a certain place on the form.

"Alignment check required" - This will force an alignment check before printing your data. This is useful for labels and other formats where page alignment is critical.

"Print grand totals on new page" - This is useful if you want to separate the grand totals from the detail report to create (in essence) a separate report for grand totals. This option will not be available for reports unless grand totals apply e.g. it does not apply for labels and full-page forms.

"Initialization print code number" - You can apply a print code to be sent to the printer before the job is started. This overrides the default initialization code for the printer driver and can be useful for things like setting another font, changing from portrait to landscape, etc.

"Terminator print code number" - Sends a print code to get ready for the next print job e.g. send print code for portrait mode after printing this report in landscape mode.



A full-page form

A full-page form is usually used to print checks, insurance forms, invoices etc. In general, anytime you want to print a page of information for a single record in a file e.g. checks, invoices, statements etc. use this output type. You can also use this type of format to print from IUA using the "form" processing command or selecting a form name while IUA.

Other Printed Output

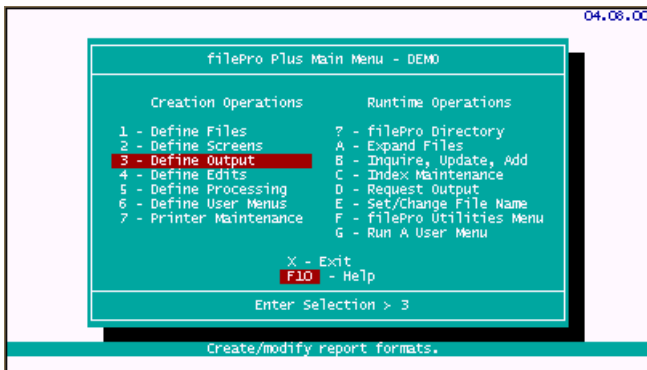
This option lets you design a form free-hand. The program gives you heading, data and sub-totals/totals and is very similar to option 1.

Process-Only format

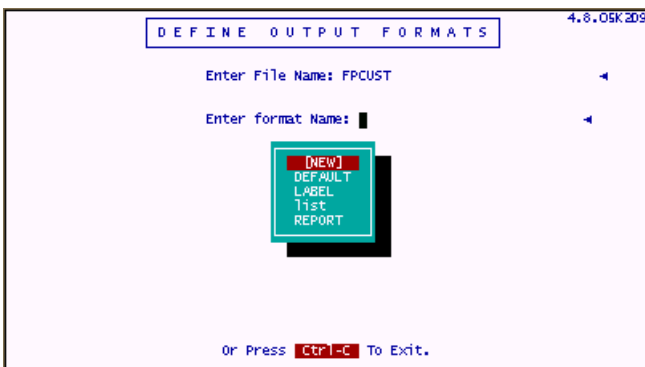
This format is primarily used for batch posting operations, archiving, recalculating results, creating export or merge files to be used with word-processing and spreadsheets etc.

There are two steps to defining processing-only formats. First is setting up the sort instructions and password (accomplished through this option) and defining the processing. Only the first step is covered on this topic. See " Define Processing " for the 2nd step.

Select 3 - Define Output



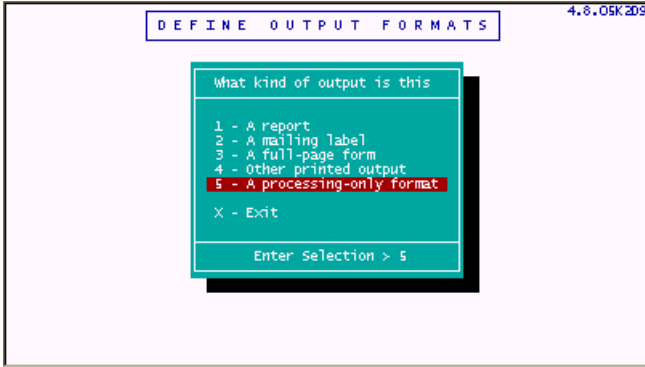
Select FPCUST and Select [NEW] to create a new format.



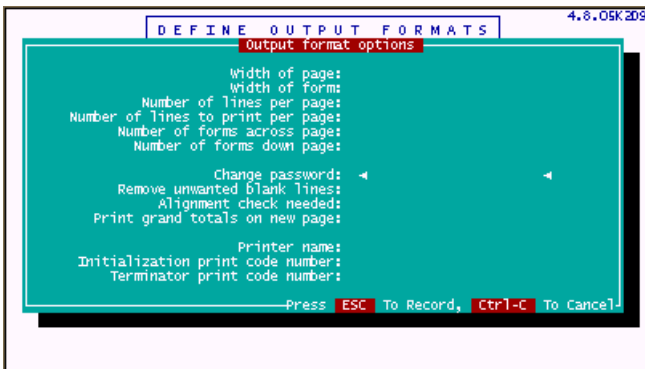
Enter "export" as the format name. This would be a typical processing-only format name.



Press Return

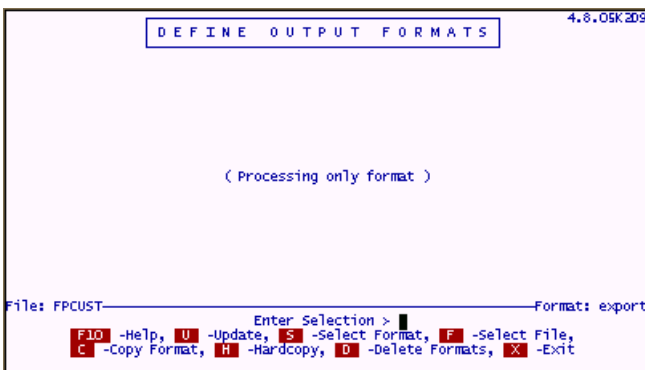


Select 5 - A processing-only format



Enter Y or N for change passwords

Note: If you assign a password, the user will be prompted for the password to run the output. If you intend to use scripts or batch files to run the processes unattended, do not assign passwords.



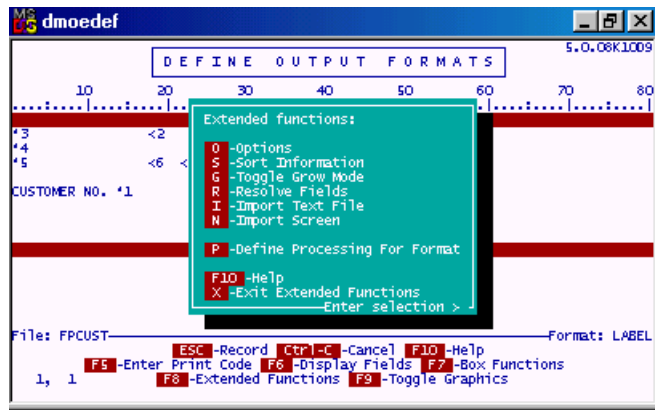
This completes the processing-only format. Press ESC to save the processing table and then Press X to exit to the filePro main menu.

See " Define Processing " for creating the "export" output processing table.

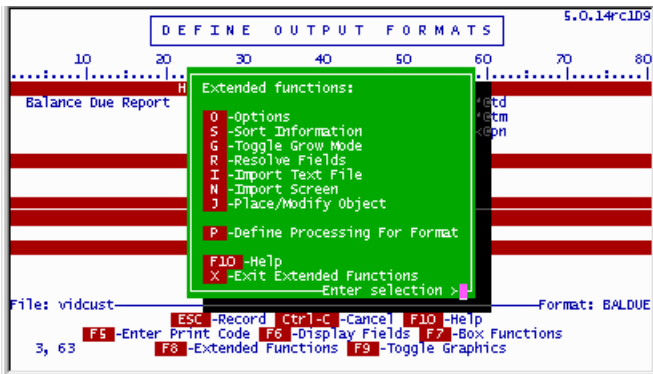
Extended Functions - [F8]

We have previously covered some of the extended functions options such as "O-Options", "S"-Sort Information, etc. but these are worthwhile mentioning again so that they are all covered in the same place in this documentation.

After pressing "U"-Update to update a report format, press the [F8] key to see the "Extended functions" menu as follows.



"O-Options" - Allows you to set various options for the report format as follows.



"Width of Page" - how wide your paper is in terms of the number of characters. The standard 8.5 inch paper is 85 characters wide when using 10 pitch (10 characters per inch). The widest that you would want to go when using standard letterhead is 80 characters to allow for margins. The maximum number of characters for 14 inch wide paper would "144". Typically you will use "132" character width of page (default) to provide a "6" character margin on each side the page. The program accepts a maximum of "255" character width of page.

"Width of Form" - With labels, the width will probably never be greater than 25 characters. If you are printing index cards with two across the page, then your width would be 40 characters. The maximum is the same as the maximum width of page i.e. "255" characters.

"Number of Lines per page" - Use the vertical measurement of your individual labels for label stock, or cards for card stock or other printing stock. Include blank lines at the top and bottom of the label in determining the number of lines to print. The number of lines to print will equal the number of lines per page for this format type. By leaving blank lines at the top and bottom, it makes it easier to position your printing stock in the printer at runtime since the top edge of the paper can be used for alignment.

"Change passwords" - Use this option to add or change your format password.

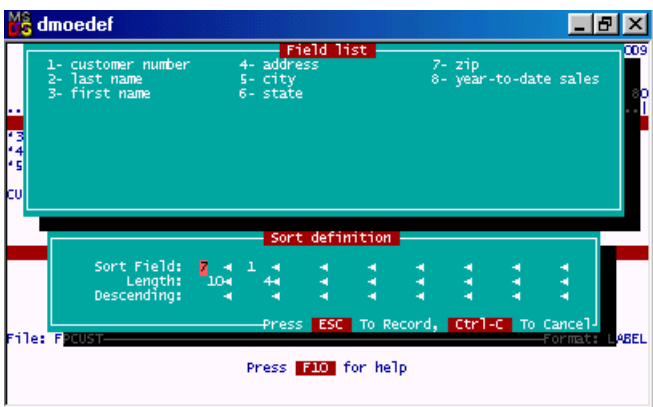
"Remove unwanted blank lines" - This will remove blank lines so is not recommended for labels or other report formats where data needs to be positioned in a certain place on the form.

"Alignment check required" - This will force an alignment check before printing your data. This is useful for labels and other formats where page alignment is critical.

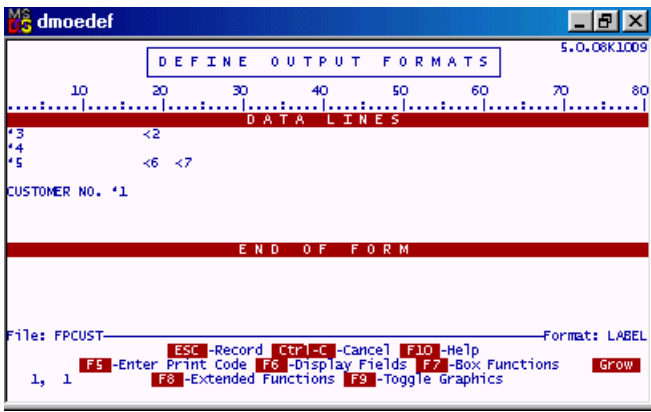
"Print grand totals on new page" - This is useful if you want to separate the grand totals from the detail report to create (in essence) a separate report for grand totals. This option will not be available for reports unless grand totals apply e.g. it does not apply for labels and full-page forms.

"Initialization print code number" - You can apply a print code to be sent to the printer before the job is started. This overrides the default initialization code for the printer driver and can be useful for things like setting another font, changing from portrait to landscape, etc.

"Terminator print code number" - Sends a print code to get ready for the next print job e.g. send print code for portrait mode after printing this report in landscape mode.



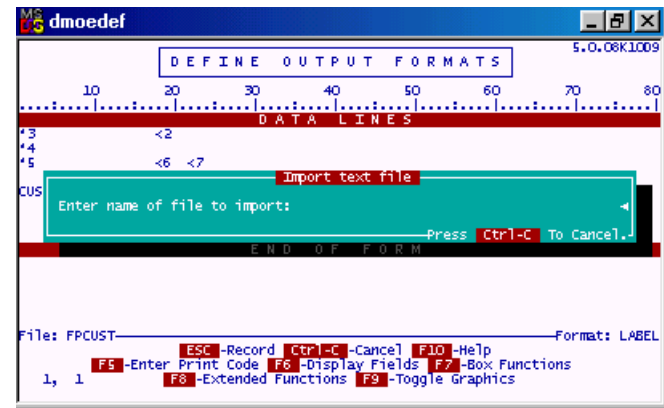
"S - Sort Information" - Allows you to control the sort order by field number, field lengths and ascending or descending order.



"G- Toggle Grow Mode" - Allows you to change how the [F3] insert key and [F4] delete key works. When grow mode is toggled on, you will see "Grow" in the lower right corner of the screen as depicted above. While in "Grow" mode, you can add or remove lines to the bottom of your format. When "Grow" mode is off, the [F3] and [F4] inserts and deletes a line under the current cursor position.

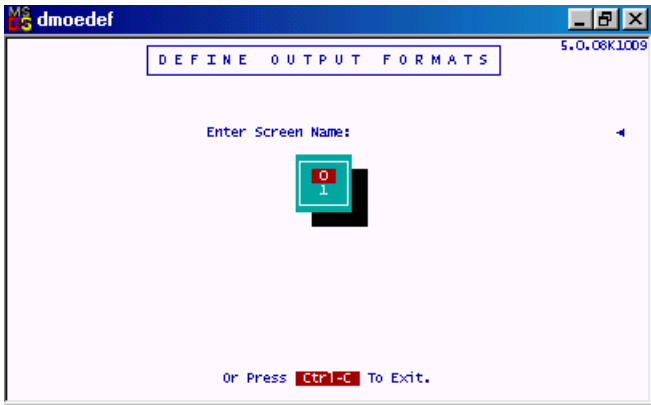
"R- Resolve fields" - This will redraw the screen and display the end of field markers to assist you in making sure that you have enough room for each field.

"I - Import Text" - Allows you to import a text file at the current cursor position.

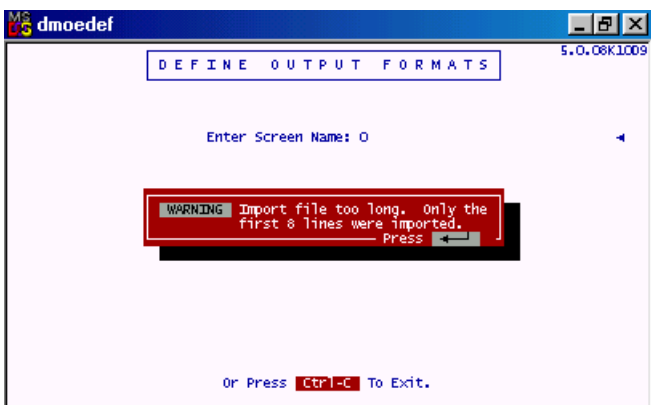


"N - Import Screen" - Allows you to import a screen format at the current cursor position. Select the applicable screen number.

Note: Make sure that your report length and report area have been adequately expanded to handle the screen that you are importing or you may overwrite things like subtotal break fields or other previously defined report variables.

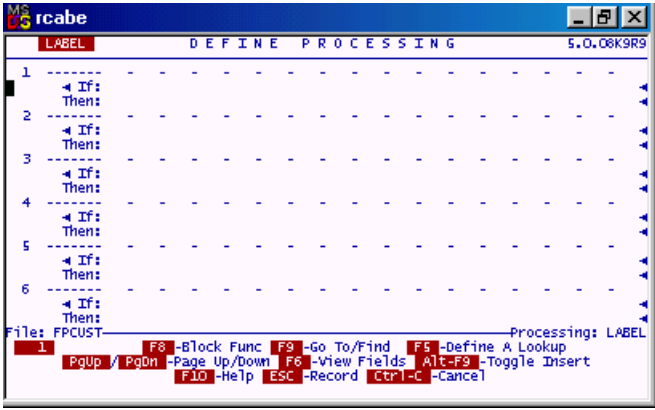


Select the screen name you want to import to your report format. The entire screen will be imported to the current cursor position. If there is not enough room for the entire screen to be imported, you will see a WARNING message similar to the following.



Press Return to accept or [Ctrl - C] to abort the import.

"P - Define Processing For Format" - This option allows you to define a processing table without exiting the report format.



Enter your processing table lines and press [ESC] to return to the defining the format.

Note: This option will provide the same functionality as using the "Define Processing" option from the filePro Plus main menu.

"X - Exit Extended Functions" - Returns you to the regular "Define Output" functions.

Define Edits - Option 4

Contents of this section

Description

Edit Dictionaries

Edit Syntax

Prompted Edits

Define Edit Screen Commands

Building Block Edits

The Global Edits Dictionary

Description:

The "Define Edits" option provides an ability to perform sophisticated validation of data as it is entered into any field or when creating outputs such as reports. This feature of filePro can save you an enormous amount of time in validating data without having to write a single line of programming code. This powerful tool is one feature of filePro that sets it aside from most other programming languages since most languages require many lines of programming code to accomplish the same things that you can do in a few minutes by creating a simple edit. This tool is sometimes call a "filter" in other program languages since it filters out unwanted data.

Edit Dictionaries

NOTE: There is a system limit on how many edits you can have. Between GLOBAL and LOCAL edits added together, they cannot exceed 200. If the total at runtime is greater than 200, the user will get a Run-Time error message letting them know that certain edits may not be recognized and applied.

A "Global Edit Dictionary" is predefined for you (referred to as GLOBAL) that will satisfy many of your validation requirements. Edits in this dictionary are available to all filePro files and normally resides at /fp/lib/edits . You can add "User" edits to this dictionary by selecting GLOBAL when entering "Define Edits".

"Local Edit Dictionaries" can be created to accommodate edits unique to a file. These edit dictionaries are also accessed, in addition to the GLOBAL edits, when working with filePro files. "Local Edit Dictionaries" are maintained in the respective filePro directory as /filePro/filename/edits.

As a general rule, only modify the "GLOBAL Edits Dictionary" if you plan to use the edit many times and across all of your applications. As you develop programs in filePro, you will find that edits that you develop for "Key fields" (a field used to link multiple files) will generally be the only edits you add to the "GLOBAL Edits Dictionary". Examples of "Local Edits" would be "Part#", "UPC#" or "Acct#" which control how a user enters data into your inventory or accounting systems. You may design several systems with different requirements on each system for "Part#" and "Acct#". The "UPC#" might be a candidate to be added to the "GLOBAL Edits Dictionary" since there are standards for formatting this type of number and you designed your systems to only use vendor provided UPC codes. However, if you also print non-standard barcodes for "Point-of-Sale" scanning, you may want to change the edit rules for "UPC#" from system to system and therefore include the "UPC#" in a "Local Edits Dictionary".

Edits Syntax

In the following descriptions, "X" and "Y" are edit expressions, and "L" is any literal, surrounded by quotes.

(X)	Parentheses may be used to separate expressions as in algebra.
[X]	The expression X is optional.
{ X }	The expression X may occur any number of times, but must occur at least once.
< L >	The literal may appear, but if it doesn't, filePro will add it. For example, Y<es> " will accept either "Y" or "Yes" as input and will turn a "Y" into a "Yes".
! L !	The literal must appear, and filePro will delete it.
X Y	Either expression is permitted. For example: "N" "N"!o! will accept only "N" or "No", and will turn a "No" into an "N".
X & Y	The data must conform to both expressions.
*	Accept any single character.
\	At beginning of line. Right justifies the resulting field.
^	Ignores case differences. Takes effect where it occurs on line.
%	Turns off case conversion. Takes effect where it occurs.
~	Converts data to uppercase. Takes effect where it occurs.
_	Converts data to lowercase. Takes effect where it occurs.

Example: edit ~"N" _<o> will accept any of the following as input and turn it into "No".

"N", "n", "no", " NO", "No", "nO"

Punctuation may be combined to form the following functions:

{ X }	The expression may occur any number of times, or not at all.
! L !	If the literal appears, it will be deleted.

Prompted Edits

filePro lets you add prompts to your edits by using and apostrophes around the prompt message. When the user moves to a field that uses a prompted edit, the prompt will appear at the bottom of the screen before the user types anything into the field.

Syntax

name prompt normal edit syntax

where "name" is the name of the edit and "prompt" is the prompt text enclosed in apostrophes (do not use quotation marks).

Define Edit Screen Commands

F - Select File	Selects another edit table or allows you to define a new file edit table for an existing file.
U - Update	Updates the edit table currently displayed.
C - Copy From	Prompts you for the edit table to append to the end of the current edit table.
Pgup/PgDn - Scroll	Scrolls you through the current edit table a page at a time.
F10 - Help	Invokes the filePro help screens.
H - Hardcopy	Prints the current edit table.
T - Test Edits	Allows you to test any system or global edit as well as any of the file edits currently displayed.
X - Exit	Exits from the Define Edits program.

Building Block Edits

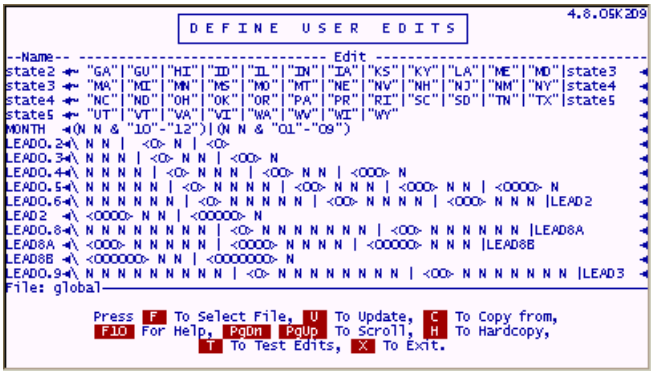
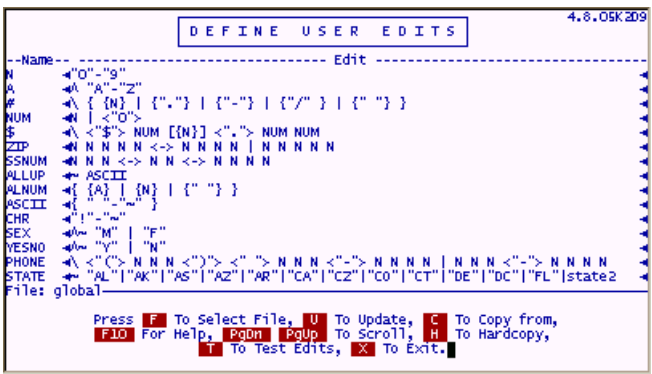
The simpler edits such as N, A and NUM are often referred to as building block edits. You can define more complex edits by using any other edit as a building block edit. In the following table, notice the way some edits make use of more than one line by naming another line to include in the edit (Example: STATE uses state2, state3, state4 etc.). Also, notice how some edits make use of other simpler edits by referring to them. Notice the prompted edit for SSNUM is enclosed in apostrophes.

```

N : "0"- "9"
A : ^ "A"- "Z"
NUM : N | <"0">
$ : A <"$"> [ "." ] NUM [ { N } ] <" "> NUM NUM
SSNUM : 'Enter social sec no' N N N <"> N N <"> N N N N
STATE : ~ "AL"|"AK"|"AS"|"AZ"|"AR"|"CA"|"CZ"|"CO"|"CT"|"DE"|"DC"|"FL" |state2
state2 : ~ "GA"|"GU"|"HI"|"ID"|"IL"|"IN"|"IA"|"KS"|"KY"|"LA"|"ME"|"MD" |state3
state3 : ~ "MA"|"MI"|"MN"|"MS"|"MO"|"MT"|"NE"|"NV"|"NH"|"NJ"|"NM"|" "NY" |state4
state4 : ~ "NC"|"ND"|"OH"|"OK"|"OR"|"PA"|"PR"|"RI"|"SC"|"SD"|"TN"|"TX" |state5
state5 : ~ "UT"|"VT"|"VA"|"VI"|"WA"|"WV"|"WI"|"WY"

```

The Global Edits Dictionary-The following edits are some of the edits predefined for you in the "Global Edits Dictionary". For a complete list of edits, go to the "Define Edits" option 5 of the filePro Plus Menu, and select [GLOBAL].



Customizing Global Edits

The "Global Edits Dictionary" can be customized for your specific needs. The following is an example of adding user edits to the "Global Edits Dictionary".

```

4.8.05K209
DEFINE USER EDITS
--Name-- Edit
state3 ← "MA" "ME" "MI" "MS" "MO" "MT" "NE" "NV" "NH" "NJ" "NM" "NY" | state4
state4 ← "NC" "ND" "OH" "OK" "OR" "PA" "PR" "RI" "SC" "SD" "TN" "TX" | state5
state5 ← "UT" "VT" "VA" "VI" "WA" "WV" "WI" "WY" | state6
state6 ← prov
MONTH ← (0 N & "10"-"12") | (0 N & "01"-"09")
LEAD0.2 ← \ N N | <0> N | <0>
LEAD0.3 ← \ N N N | <0> N N | <00> N
LEAD0.4 ← \ N N N N | <0> N N N | <00> N N | <000> N
LEAD0.5 ← \ N N N N N | <0> N N N N | <00> N N N | <000> N N | <0000> N
LEAD0.6 ← \ N N N N N N | <0> N N N N N | <00> N N N N | <000> N N N | LEAD2
LEAD2 ← \ <0000> N N | <000000> N
LEAD0.8 ← \ N N N N N N N N | <0> N N N N N N | <00> N N N N N | LEAD8A
LEAD8A ← \ <0000> N N N N N | <00000> N N N N | <000000> N N N | LEAD8B
LEAD8B ← \ <000000> N N N | <00000000> N
LEAD0.9 ← \ N N N N N N N N N | <0> N N N N N N N N | <00> N N N N N N N | LEAD3
File: global
Press F10 For Syntax Help Screen,
ESC To Record, Ctrl-C To Cancel.

```

```

4.8.05K209
DEFINE USER EDITS
--Name-- Edit
pepnum2 ← ( (108! <AUG> ) | (109! <SEP> ) | (110! <OCT> ) | (111! <NOV> ) | (112! <DEC>
mhabr ← "JAN" "FEB" "MAR" "APR" "MAY" "JUN" "JUL" "AUG" "SEP" "OCT" "NOV" "DEC"
delnum ← ( 10! | 11! | 12! | 13! | 14! | 15! | 16! | 17! | 18! | 19! )
UNPHONE ← { "!" "C" | "!" "!" "!" | "!" "!" "!" "!" | "!" "!" "!" | N }
zip ← uszip10 | canzip | uszip5
uszip10 ← \ N N N N N <"-"> N N N N N
uszip5 ← \ N N N N N
canzip ← cza czn cza <"-"> czn cza czn
cza ← 10! <0> | 11! <1> | 12! <2> | 13! <3> | 14! <4> | 15! <5> | 16! <6> | A
czn ← 18! <8> | 19! <9> | 10! <0> | 15! <5> | 12! <2> | 11! <1> | N
stprov ← \ State or Province state prov
prov ← "AB" "BC" "LB" "MB" "NB" "NS" "NT" "ON" "PE" "PQ" "SK" "YT"
File: global
Press F10 For Syntax Help Screen,
ESC To Record, Ctrl-C To Cancel.

```

Notice how the Canadian province abbreviations have been added and a new line "state6" has been added to the edits table. Canadian ZIP Code "canzip" and other building block edits have been added to the pre-defined "zip" edit so that all U.S. and Canadian ZIP codes are handled properly.

TIP: You may find the following useful if you need a Spanish CHEQUE edit.

```

CKESP:{" "}|(cerol <***> (esp3|esp2|mil|cien|diez|unos) chele | <***zero> chele)
esp3:N N N N N N & (1000! | ( 100! diez | cien ) < mil > ) ( mil|cien|diez|unos ) | tt
esp2:N N N N N N & (100! | ( 10! unos | diez ) < mil > ) ( cien|diez|unos )
mil:N N N N N & (10! | xunos < mil > ) ( cien | diez | unos )
cien:N N N N & (10! | xunos < cientos > ) ( diez | unos )
xunos:10!|11!|2!<dos>|3!<tres>|4!<cuatro>|5!<cinco>|6!<seis>|unos2
unos:10!|1!<uno>|2!<dos>|3!<tres>|4!<cuatro>|5!<cinco>|6!<seis>|unos2
unos2:7!<siete>|8!<ocho>|9!<nueve>
diez:10!<diez>|11!<once>|12!<doce>|13!<trece>|14!<catorce>|diez2
diez2:15!<quince>|16!<diez y seis>|17!<diez y siete>|18!<diez y ocho>|diez3
diez3:19!<diez y nueve>|20!<veinte>|30!<treinta>|40!<cuarenta>|50!<cincuenta>|diez4
diez4:60!<sesenta>|70!<setenta>|80!<ochenta>|90!<noventa>|diez5
diez5:(10!|diez6 <" "> " ") unos
diez6:12!<veinte>|3!<treinta>|4!<cuarenta>|5!<cincuenta>|6!<sesenta>|diez7
diez7:7!<setenta>|8!<ochenta>|9!<noventa>
chele:< con > [!.!] num num </100 ****>
zero:zero1 <*** zero> !.! chele
zero1:(N N N N N N & 100000!) | (N N N N & 10000!) | zero2
zero2:(N N N N & 1000!) | (N N & 100!) | (N & 10!)

```

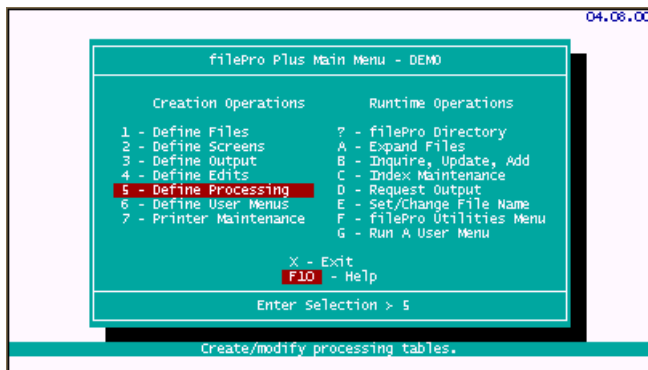
Define Processing - Option 5

Contents of this section

- Option Screens*
- Simple Processing Examples*
- Sample Processing Tables [INPUT & AUTO]*
- Use of @KEY [triggers]*
- Use of @SN [screen name]*
- Output Processing Uses*
- Flags_Define_Processing_rcabe_dcabe>Second [cabe.flags link]*

Advanced Concepts

- Arrays*
- Debugger*
- Drop ALL*
- Help Screens*
- ListBox Command*
- Lookup Dash*
- Menu Command*
- O/S File I/O Functions*
- Sort Select Processing Tables*



Option Screens

When selecting "Define Processing", you will see a listbox of available files to choose from.

Dont be concerned if the listed files are different than what you see on the above screen since you will have likely created files with different names when using the "Define Files" option. When selecting a file, you will see "Define Processing" options for creating processing as follows;



The above options provide for defining 3 different types of processing tables that control data entry and affect the results of output options. Notice that option "4" allows you to change the filename you previously selected so that you do not have to back out of this menu to reselect a file if you make a mistake or when defining processing for more than one file.

Simple Processing Table Examples

This section contains sample INPUT, TRIGGER, and AUTOMATIC processing. These are 3 of the most used and useful types of filePro processing. Learning how to read code like this is not difficult. Usually, a good English explanation for each line of code serves well enough to describe how to use a command or function. The Processing Reference contains the syntax and description of every filePro processing function.

INPUT

INPUT processing is performed just after the user SAVES the record being added or updated. The following code asks the user if everything on his screen looks okay. Element "1" captures an answer and puts it into a dummy field "q". Element "2" acts on this answer by testing the value of "q". If the answer is anything but "Y", the users cursor is put back onto the screen and he is given a chance to fix whatever didnt look right. The next the user SAVES the record this question is asked again, and again, and again, until he answers

with a Y, then the process falls through the test of q and ends. The (1,yesno) after the dummy field q limits the users answer to only one character, and further limits that character to either a Y or an N or an ENTER. (In this process, both N and ENTER mean the whole process will restart.)

```

LABEL          DEFINE PROCESSING          4.8.05K409 DEMO
1  -----
  < If:
  Then: input q(1,yesno) "Does ewrything look alright? (y/n) "
2  -----
  < If: q ne "y"
  Then: restart
3  -----
  < If:
  Then: end
4  -----
  < If:
  Then:
5  -----
  < If:
  Then:
6  -----
  < If:
  Then:
File: test1-----Processing: input
1  F8 -Block Func, F9 -Go To/Find, F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

TRIGGER (@key)

"filePro" can act on many different triggers. These can be events within the program or actions taken by the user. The @keyX trigger is activated whenever the user presses key X. (or whatever key is specified @keyA, @keyB, @key?). @key processing is written on the INPUT table but does not happen when the user SAVES the record. It only happens when the designated key is pressed. This code will put the time on the screen for 3 seconds and then clear it off the screen. (The Unix operating system uses seconds for filePro's SLEEP command. Windows uses milliseconds, hence the test to see which operating system is running this code. @os is another system maintained field within filePro.)

```

LABEL          DEFINE PROCESSING          4.8.05K409 DEMO
1  -----
  < If:
  Then: input q(1,yesno) "Does ewrything look alright? (y/n) "
2  -----
  < If: q ne "y"
  Then: restart
3  -----
  < If:
  Then: end
4  -----
@keyT < If:
  Then: show "The current time is \r" < @tm < "\r"
5  -----
  < If: @os eq "DOS"
  Then: $sleep "3000" ; end
6  -----
  < If: @os eq "UNIX"
  Then: $sleep "3" ; end
File: test1-----Processing: input
1  F8 -Block Func, F9 -Go To/Find
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

AUTO

AUTO processing (called "automatic" under Unix) is performed immediately after a record is retrieved from the disk and just before it is displayed on the screen (or used on an output of some other kind (printouts, etc.). This means fields on a record can be tested and actions taken based on these values before the user ever sees the data.

The following code tests whether this table is being run under IUA. If it is, there will be a current screen. When automatic tables run along with output processes, there is no current screen. FilePro maintains a dummy field that holds the current screen name, @sn.

The first thing this table does is check the value of @sn (the current screen name). If @sn is empty, the process just ends and nothing is done. Therefore this table will only do something under IUA. If this is running under IUA, the next "if" condition will be tested. It says, is the Balance_Due field of this record greater than "500". If it is, sound the speaker. If the test is false, the process falls through to the next line and just ends.

```

LABEL          DEFINE PROCESSING          4.8.05K409 DEMO
1  -----
  < If: @sn eq ""
  Then: end
2  -----
  < If: Balance_Due gt "500"
  Then: beep
3  -----
  < If:
  Then: end
4  -----
  < If:
  Then:
5  -----
  < If:
  Then:
6  -----
  < If:
  Then:
File: test-----Processing: auto
1  F8 -Block Func, F9 -Go To/Find
PgUp / PgDn -Page Up/Down, F6 -View Fields, Alt-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctrl-C to Cancel.

```

These are very simple examples of processing code. It does not get too much harder than this to understand, you just have to read the syntax on how to use a command and the description of what it does. Then, try it out. Experimentation is the best way to learn any programming language.

Output Processing Tables Uses

"Output" processing tables are usually defined to support output formats i.e. reports, labels and posting actions. The "Define Output Processing" option can also be used to define both "Input" and "Automatic" processing since these types of processing do not have to be called by their respective names when using option flags. Although this may sound confusing, keep in mind that these options are available to you and will be discussed in more detail in the "Define Menus" topic. In essence, the names "INPUT" and "AUTO" or "automatic" can be replaced by names of your choice when using the appropriate flags.

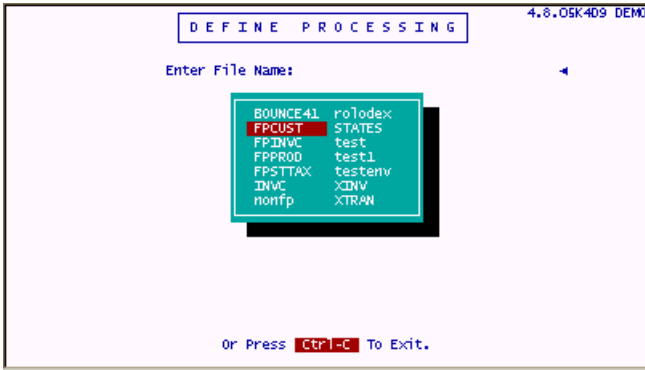
Exporting Data

A common requirement for output processing table is exporting data to ASCII or comma-delimited files for word-processing mail merge. This is easily accomplished with just a few lines of code in most cases.

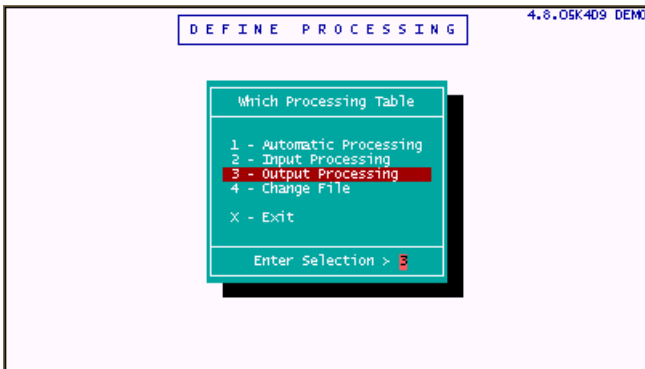
Example: You want to export name and address information from your customer file to be merged into a "Newsletter".

Select option 5 - "Define Processing" from the filePro Main Menu.

When presented with the following screen, select a filename (**FPCUST** in this example).



Select 3 - Output Processing

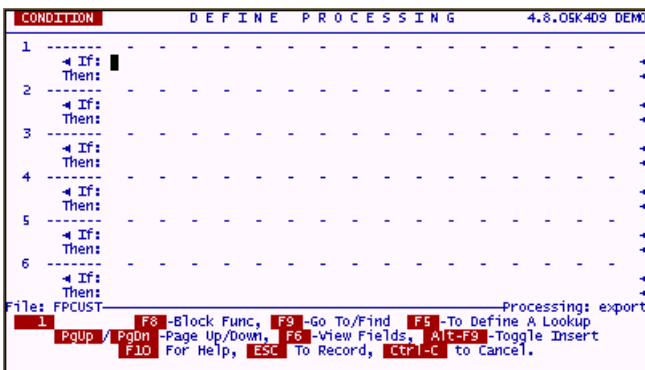


Select [NEW] for a new output processing format name.

Type the name "export" (for this example) and press Return.



This screen is the same as the screens previously used to create automatic and input processing and contains a Label, Condition and Action sections for each element.



At this point, we need to find the field numbers for the name and address for the mail merge output. Press the F6 key to see the FPCUST fields. For a "word" merge file, the "first name" and "last name" should be combined into a single field so that it reads properly in the newsletter e.g. "John Smith".

We would also want to combine the "City", "State" and "ZIP" into a combined field since they will usually be treated as a single field when being merged into the newsletter.

```

DEFINE PROCESSING 4.8.05K409 DEMO
1- customer number 4- address 7- zip
2- last name 5- city 8- year-to-date sales
3- first name 6- state

File: FPCUST
PgDn And PgUp - Scroll Fields, H - Hardcopy, ← - Return
Fs - Field Lengths/Edits, D - Dummy Fields, F6 - New File

```

Enter the following lines to export the FPCUST data.

```

LABEL DEFINE PROCESSING 4.8.05K409 DEMO
1 -----
  < If: 'Combine first name & last name
  Then: n = 3 < 2
2 -----
  < If: 'Combine City, State & zip
  Then: c = 5{", "< 6 < 7
3 -----
  < If: 'export the file to the \temp directory; make sure \temp exists
  Then: export word merge = \temp\fpcust.wp
4 -----
  < If: '1st fld = first name & last name
  Then: merge(1) = n 'assign "n" to merge field #1
5 -----
  < If: '2nd fld = address
  Then: merge(2) = 4 'assign "4" to merge field #2
6 -----
  < If: '3rd field = city,state ZIP
  Then: merge(3) = c 'assign "c" to merge field #3
File: FPCUST Processing: export
F8 -Block Func, F9 -Go To/Find, F5 -To Define A Lookup
PgUp / PgDn -Page Up/Down, F6 -View Fields, ALT-F9 -Toggle Insert
F10 For Help, ESC To Record, Ctr-C to Cancel.

```

The six lines of code entered in the above table will create a comma-delimited file that can be merged into your newsletter.

Element 1 uses the "<" operator to push the last name left to the "first name" leaving a space between the fields.

Element 2 uses a combination of operators and literal text to combine the "city", "state" and "zip" fields into a single field for the output. The "{" pushes the comma right up against the "city", and the following "<" leaves a single space between the comma and "state", and between "state" and "zip".

Element 3 reflects an "alias assignment" name e.g. "merge" and identifies the output path/filename.

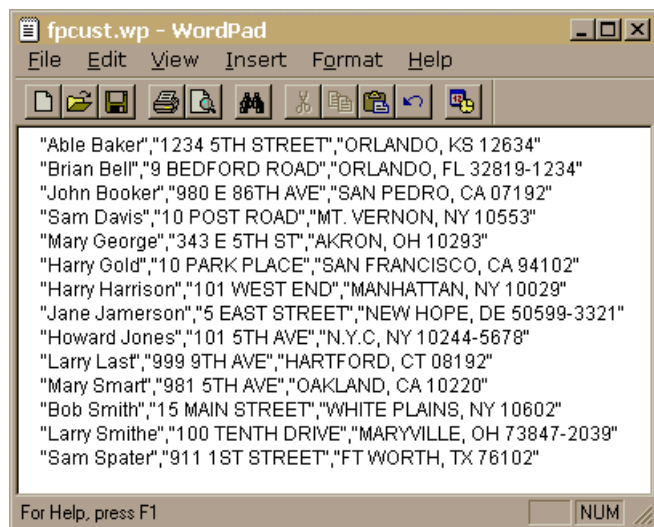
Elements 4 through 6 simply assign the dummy and real field values to positions in the output file "\temp\fpcust.wp" e.g. output position 1 of "merge" is assigned the value of "n" (the combined name); output position 2 is assigned the value of field "4" (address); output position 3 is assigned the value of "c" (combined field for city, state,zip).

When defining processing, make use of comments using the apostrophe to document your code. For the little time that it takes, it will save a lot of time when you want to revise the code. This is imperative when more than one programmer is involved in maintaining a system.

After entering the above lines, **press ESC** to save the "export" processing table. Answer "Y" to check the syntax and print hardcopies as desired.

Sample Export Output

The following is the results of the "export" processing example for file "FPCUST". You can create this sample output by going to Option "D" of the filePro Main Menu, selecting file "FPCUST", selecting format name "export", and selecting all records, or by clicking [Run Report](#). If the format name is not shown, go to "Define Output" and create the Processing-only format sample "export" for file "FPCUST".



Sort & Selection Processing

You will also use the "Define Output Processing" option to develop "Sort and Selection" processing.

NOTE: There are [flags](#) for Define Processing (*cabe) which allow you to customize operations from a menu or command line.

Defining files now allows you to tag a specific auto processing to use with the process for tokenizing and syntax checking. This alternate auto process is displayed at the bottom of the window. *clerk and *report will use this alternate process if there is no -y flag on command line.

Define User Menus - Option 6

Contents of this section

Defining User Menus to call filePro and other programs

One line actions and commands

Multi-line commands [scripts, batch files]

Menu Flags

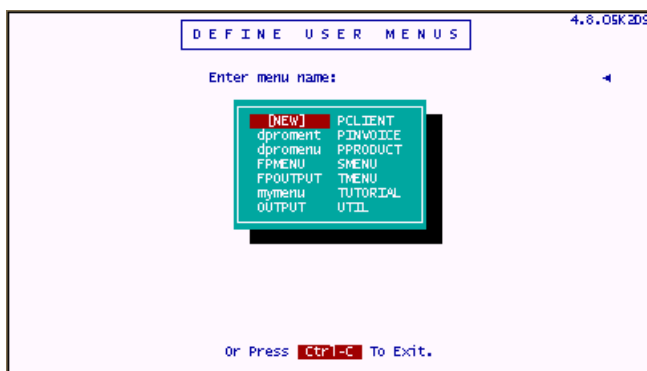
Advanced Concepts

It is surprising how many filePro users access all of their programs and data through the main filePro menu. This menu is more than adequate for most operations that concern designing and testing files, but once the program is up and running, it is overkill. The entire creation side and a good percentage of the runtime side of the menu are not required to operate the newly created program. In fact, most business users out there rarely do anything more than Inquire, Update & Add and Request Output from this powerful 4GL menu. Actually, it is even worse than that. A good majority of the users don't ever use the Set/Change file Name function either, which means they spend a great deal of their time answering lots of questions about; which file, what screen, browse mode, what record selection mode, etc.

You can make your computer time infinitely more productive by designing custom menus of your own that take you directly to the things you want to do without asking all those questions. You give the menus that you design names and you can run these "User Menus" from the command line or from the "G" option on the main menu, or most commonly from another menu.

Define User Menus

When entering "Define User Menus", a screen similar to the following screen is presented. You may see additional menu names not reflected in this example.

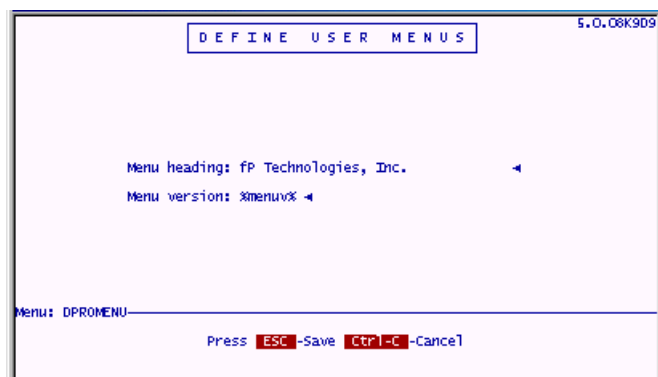


After selecting a menu name or [NEW] option you will get the following screen which allows you to enter "Menu Heading" and "Menu Version" for your user menu.



Menu Version

With filePro version 5.0.9 and later, you can use a system variable to control the menu version that is displayed for the user. Identify your variable name enclosed with the "%" symbol e.g. %menuv% where "menuv" is the variable included in your filePro configuration or system environment. You can use any variable within the limits of the version field



After saving the menu name and version, you will see the menu editing screen depicted below. You can enter up to 24 choices per menu, but it is often better to leave blank lines or to place sub-titles for your menu by leaving the choice blank and entering just the "Description Line". This will place a category title on your menu i.e. "My Creation Operations" in

the following example.

```
CHOICE          DEFINE USER MENUS          4.8.05K209
-----
1  My Creation Operations  <
-----
2  1 Define Files         <
   /fp/ddefine           >
-----
3  2 Define Screens      <
   /fp/dscreen          >
-----
4  3 Define Processing (Runtime) <
   /fp/rcabe - -t 120000 -ty 30000 >
-----
Menu: mymenu
F5 - Create batch file, ESC - Save, Ctrl-C - Cancel, F10 - Help
```

Choice

Any single letter, number or punctuation mark except x and X. The entry "X - Exit" is automatically added to every menu.

Description

Any text that would tell the user what the menu entry does, such as "Print end-of-month report" or "Run Word Processor".

Action

The program, single or multiple operating system commands that you want executed when a user chooses this element.

You can use flags with many of the filePro programs to apply parameters.

In the above screen, "-t 120000" and "-ty 30000" are flags to control the processing table tokenization sizes.

Note: Use a minus "-" after filePro program names when using flags. This reserves space for the filePro filename variable so that the following parameters are recognized as flags.

Example: /fp/rcabe - -t 120000 -ty 30000

Menu Batch/Script Files

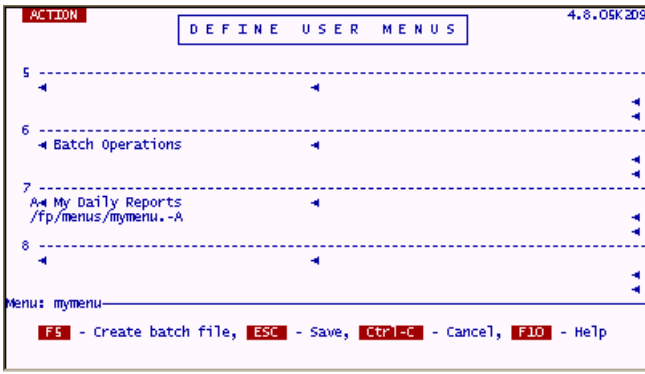
You can press the F5 key on the action line to create a filePro batch /script file when you want to perform more than one action per menu element.

```
ACTION          DEFINE USER MENUS          4.8.05K209
-----
5  <
-----
6  Batch Operations      <
-----
7  A My Daily Reports    <
-----
8  <
-----
Menu: mymenu
F5 - Create batch file, ESC - Save, Ctrl-C - Cancel, F10 - Help
```

Example: You need to run several reports each day before leaving work. Set up an option for "My Daily Reports", go to the action line and press the F5 key.

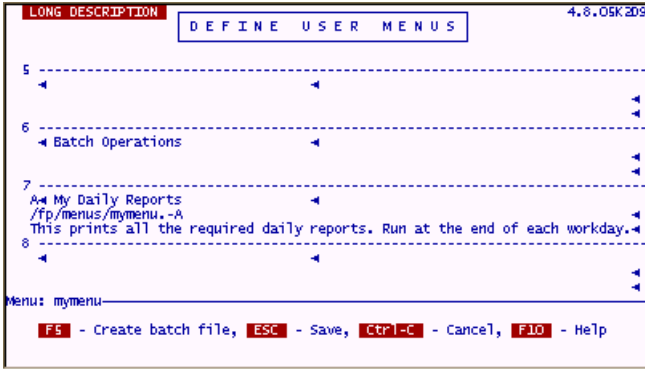
```
CREATE BATCH FILE          4.8.05K209
-----
echo off
/fp/dreport cust -f balance -s balance -h "Daily Customer Balance"
/fp/dreport invoice -f postgl -s postgl -h "Post Invoices to General Ledger"
rem
rem This demonstrates use the filePro menu batch/script option
rem
-----
Menu: mymenu
ESC - Save, Ctrl-C - Cancel, Alt-F9 - Toggle Insert, F10 - Help.
```

When pressing ESC to save the batch file, notice how the menu action line is entered for you. The action line contains the menu name with a suffix ".A" which corresponds to the CHOICE you have entered.

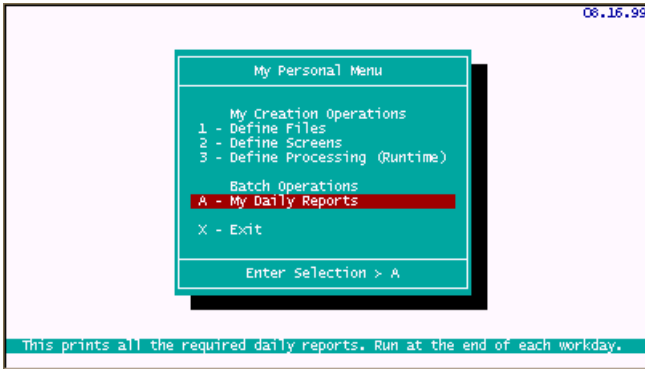


Long Description

This string will be centered on the bottom line when running your user menu when the menu cursor is on this element. "Long Descriptions" allow you to explain what the option does to the user, but can be left blank.



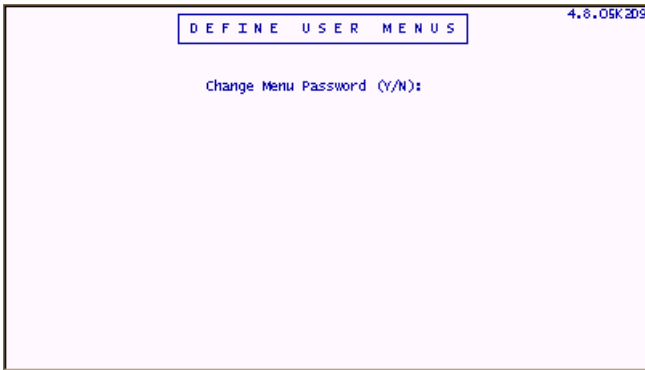
The long description entry in "MYMENU" option "A" would be presented to the user as follows;



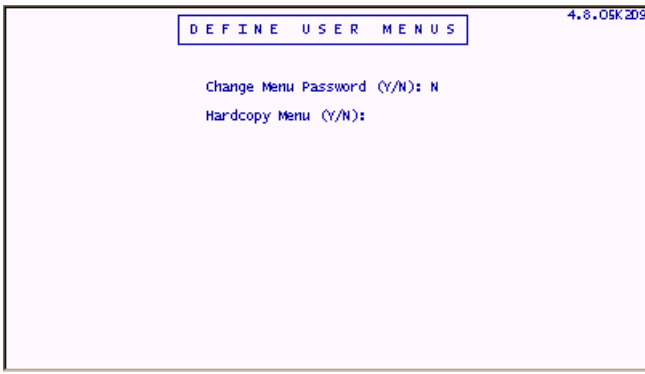
The filePro "Define Menus" option is not limited to filePro programs. You can include your any system utility programs like editors, word processors, and generally apply about anything that you can do from the system prompt to manage your user applications.

Menu Passwords

Each menu can be password protected. The password is case sensitive in all operating systems.



You have the option to print a hardcopy of the menu when leaving the "Define User Menus" option.



Flags

Add flags to the menu action lines to control how your data is presented or output. Different flags apply depending on the program that you are running i.e. clerk, report, dxmaint etc. Any flag that can be used in a menu option can also be used from a command line. Refer to the following topics for lists of menu action flags.

[Flags - Inquire Update & Add](#) - Use these flags to control user input.

[Flags - Request Output](#) - Use these flags to control the creation of reports, printer redirection, selection etc. (report programs)

[Flags - Expand Files](#) - Use these flags to control expanding filePro files to reserve blank records. (dexpand program)

[Flags - Index Maintenance](#) - Use these flags to perform Index rebuilding. (dxmaint program)

Note: You can not put passwords on an action line or in a batch/script file. The user will be asked for it at the appropriate point in the program.

User Menu Indicators - Use these indicators at the beginning of the action line or lines in a filePro batch/script file indicate menu operational options.

@ Waits for user acknowledgement after executing command.

Returns to previous menu after executing command.

! Tells filePro that the following command is a user menu. (This speeds the loading on UNIX/XENIX systems.)

Advanced Concepts

Flags

[Flags - Inquire Update & Add](#) - Use these flags to control user input.

[Flags - Request Output](#) - Use these flags to control the creation of reports, printer redirection, selection etc. (report programs)

[Flags - Expand Files](#) - Use these flags to control expanding filePro files to reserve blank records. (dexpand program)

[Flags - Index Maintenance](#) - Use these flags to perform Index rebuilding. (dxmaint program)

Note: You cannot put passwords on an action line or in a batch/script file. The user will be asked for it at the appropriate point in the program.

User Menu Indicators - Use these indicators at the beginning of the action line or lines in a filePro batch/script file indicate menu operational options.

@ Waits for user acknowledgement after executing command.

Returns to previous menu after executing command.

! Tells filePro that the following command is a user menu. This speeds the loading of the menus.

Define User Menus allows you to create or modify any filePro menu (including the Main Menu itself which has a name of "dpromenu").

NOTE: Any changes you make to "dpromenu" will not become apparent until you exit the Main Menu or load a different menu and then come back to the Main Menu).

Advanced Example

Below is a User Menu called "test" to demonstrate a variety of things. You can copy some of the lines to a test menu and experiment with the actions shown here.

Two notes: The "-h" is used on menu action lines to place a heading at the top of the screen when the action is being performed. The heading must be enclosed in quotes. Here it is used to describe the actions a little better and, of course, not needed to run the action in any way. Choice "E" is a script which you created while in the Define Menus program. The script is created on a screen behind the menu (so to speak) and is reached by pressing F5 (check the prompts on your screen). This script is stored in a *nix file and the name "menuname.-CHOICE" is substituted on the command line for you automatically. You will need this feature if you want to do things that cannot fit on one action line.

Below is "test menu".

1	CUSTOMER FILE	D	CATALOG FILE - to screen
2	CUSTOMER FILE (screen 1)	E	PRINT 3 REPORTS IN A ROW
3	CUSTOMER FILE (Add Records)	F	SCRIPSIT WORD PROCESSING
4	PRODUCT FILE (Index B)	G	MAIL SYSTEM
5	INVOICE FILE (Selection XXX)	H	Edit a File
6	CATALOG (Browse Mode On)	I	SHELL ("exit")
7	QUIKSTART (large token size)	J	FILEPRO MAIN MENU
8	VENDOR FILE (output to file)	K	SOME OTHER USER MENU

9	VENDOR FILE (debugger on)	L	WHO IS ON THE SYSTEM
A	CUSTOMER FILE - Summary	M	Utilities
B	CUSTOMER FILE - Print Invoices	N	BACKUP FILEPRO FILES
C	VENDOR FILE - Report (Index.A)	O	Windows

Below are the action lines associated with the above menu.

```

1 /fp/dclerk cust -h "Customer File - Ask for screen number"
2 /fp/dclerk cust -s 1 -h "Customer File"
3 /fp/dclerk cust -s 1 -xa -h "Customer File - Add Records"
4 /fp/dclerk prod -s 1 -xib -h "Product File - Direct index B."
5 /fp/dclerk prod -s 1 -xs XXX -h "Invoice by selection XXX."
6 /fp/dclerk catalog -s 1 -b -h "Catalog - Browse Mode On."
7 /fp/rcclerk - -h "Asks for filename because of the -."
8 /fp/dclerk vend -s 1 -p /tmp/file -h "Any output to /file."
9 /fp/dclerk vend -s 1 -db -h "Vendor File with debugger on"
A /fp/dreport cust -f summary -a
B /fp/dreport cust -f invoice -s new
C /fp/dreport vend -f reportname -iA "Uses index A"
D /fp/dreport catalog -f inventory -pq -h "Menu of output choices"
E /fp/menus/test.-E
F cd /usr/wp ; wp
G mail
H edit
I /bin/sh (Windows=command.com)
J !dpromenu or "p dpromenu"
K !usermenuname or "p usermenuname"
L @who (Windows=n/a) -h "The @ pauses the output to see it!"
M !util or "p util"
N cd /appl/filePro ; tar -cvf /dev/rctO . (Windows=pkzip -rp filepro /appl/filepro/*.* )?
O win

```

The following script relates to choice E and stored in file /fp/menus/test.-E

```

/appl/fp/dreport filename -f report1 -a
/appl/fp/dreport filename -f report2 -a
/appl/fp/dreport filename -f report3 -s sset

```

Hints:

Use the Menu Version number to print the menu name on the screen. It is not worth much as a version number, but it is valuable as a name. Users can tell you immediately which menu they are using when they run into trouble.

If **PFNAME** is set, then the value of PFNAME will be show at the top of all menus/submenus. This allows the programmer to use PFNAME in manners not originally intended, but nonetheless is very useful. You could use PFNAME to display the qualifier being used.

In Unix, where all commands specify -m\$qualify by setting PFNAME = \$qualify.

In WINDOWS/Windows where PFQUAL has been set, set PFNAME = %pfqual%.

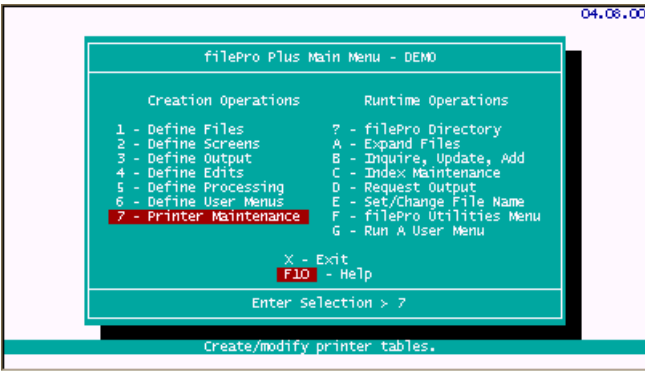
Printer Maintenance - Option 7

Contents of this section

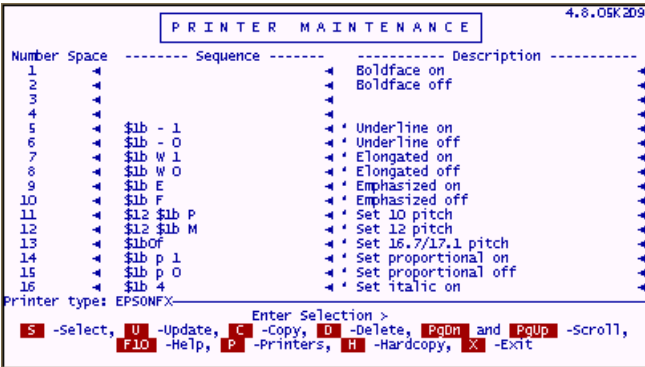
- Printer Maintenance Description
- Entering Print Code Sequences
- Escape Sequences
- Decimal Control Codes
- Hexadecimal Control Codes
- Printer Routing
- Copy and Modifying Printer Drivers
- Advanced Concepts
- Expert

Description:

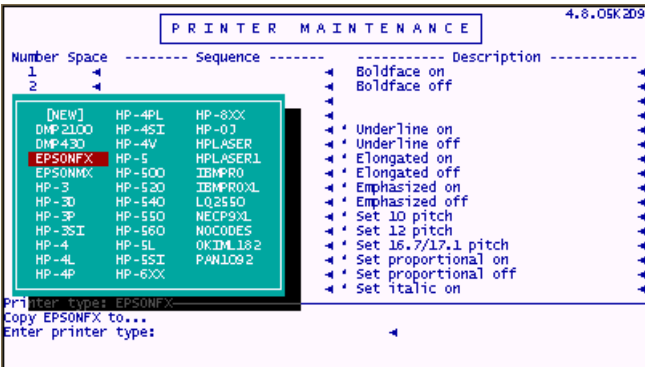
Option 7 allows you to configure printers and to create printer drivers for use with filePro. filePro has pre-defined many of the printer drivers for you. These pre-defined drivers can be used as templates to create a new printer driver if your specific printer is not reflected in the list.



When you select the "Printer Maintenance" option "7", you will see a table similar to the following;



When pressing "S" - Select, a listbox of available printer tables is displayed. This presents a range of pre-defined printer drivers that you can modify or copy to a new printer driver name.



filePro accepts print code sequences in ASCII, decimal, and hexadecimal form. Control character combinations are also accepted. So you have maximum flexibility and can use whatever sequences you prefer. You can even mix these different forms in one print code sequence. Blank spaces are ignored but you may wish to use them in your sequences for clarity.

- U - Update** Update the print code table currently loaded. Note: Print codes 1 and 2 cannot be changed.
- S - Select** Selects a different table or define a new one.

C - Copy	Copies the current table of print codes to a new table or replace The print codes of an existing table with the current table.
F10 - Help	Displays help similar to what you are reading now.
D - Delete	Selects print code tables to delete.
P - Printer Maintenance	Goes to the Available Printers table to tell filePro what printers are available, which print code tables they use, and how they are connected to the computer.
H - Hardcopy	Prints the current print code table.
Number	The print code number.
Space	The number of printed spaces, if any (0-99), that the print code will take up when it is sent to the printer. For example, a "Horizontal line" symbol (code 45) will take up one space, While "Underline on" (code 5) wont take up any spaces. A Zero or a blank in this column means no spaces.
Sequence	The print code sequence, from your printer manual, for the print code desired. Help (with sequence syntax) can be retrieved by pressing F10 while you are updating a table.
Description	These are simply descriptions of what a print code entered here would be expected to do.

The first 54 code descriptions are pre-defined. For print code tables you create new or edit, you can fill in the number of spaces (if any) and the print code sequence that match the description. Print code descriptions can be either global or local. Global descriptions are the default descriptions that are predefined for every print code table. Local descriptions apply to the table in which they are defined. Whenever you edit a description, filePro asks whether the description should be saved as global or local. To edit a global description press F5.

Entering Print Code Sequences

filePro Plus accepts print code sequences in ASCII, decimal, and hexadecimal form (note examples below). Control-character combinations are also accepted. You can even mix these different forms in one print code sequence. The program ignores blank spaces, but you may wish to use them in your sequences for clarity.

# - Decimal	Type a pound sign (#) in front of each decimal number. Example: Epson FX, subscript on, is: #27 #83 #1
\$ - Hexadecimal	If the number has an odd number of digits such as 0 or 1, add a zero to the beginning of number. Example: Epson FX, subscript on, is: \$1b \$53 \$01 or (does not work in decimal): \$1b5301
^ - Control Characters	Type a caret (^) in front of the character. The ESC Character in your printer manual can be entered as ^[.
% nn - Nested Sequences	Type a percent-sign (%) and the code number of the Code whose sequence you want to insert (nest) in. Another sequence. Example: Epson FX, you can Nest the sequence for code 5 in code 12 by typing: \$1b M %5. Since the sequence for code 5 is "\$1b - 1", you are entering the equivalent of: \$1b M \$1b - 1.
% name	Download a file to the printer "name" is the file to Send to the printer. Use with environmental variable PFDLDIR.
\\ - Literals	Type a backslash (\\) in front of a character to be Taken literally if it is ordinarily used for another Purpose by a print code table, as the \$, #, and % Symbols are, for example. Example: If you needed to use the code "ESC-\$", you would enter ^[\\$ rather than simply ^[\\$. \\\$ tells filePro that the dollar-sign is a dollar-sign and not the flag for a hexadecimal number. Any other character will be interpreted as an ASCII literal.

Table Options

Type the codes shown in your printer manual following these rules.

Printer description	A description of the printer limited to 40 Characters.
Initialization print code number	The number of the print code, if any, that you Want to be sent to the printer before anything else.
Terminator print code number	The number of the print code, if any, that you Want to be sent to the printer after the output has been printed.
Graphic start print code number	The print code number at which the graphic sequences begin in the current table. On the default table the graphic sequences run from code 44 through code 54.
New-Line options	1=CR/LF; 2=CR; 3 = LF; 4 = "\\ "LF. Enter the number 1,2, 3 or 4 for the kind of code that this printer requires to move to the beginning of the next line. Note: Code 4 was added in 5.0.9 to support the fp2RTF printer option.
Printer can form feed? (Y/N)	If the printer feeds a whole page of paper in response to the print code "\$0c", then answer "Y", otherwise, "N".
Printer can backspace? (Y/N)	If you get a backspace in response to code "\$08", then press "Y", otherwise press "N".
Page width	The maximum number of characters to print Across the page. (Usually, you will set this to either 80 or 132).
Page length	The length of the paper measured in printed lines. (For 8 1/2" X 11" paper, you would set this to 66).
Print length	The number of lines to print per page. This no may be less than the page length and will give the page top and bottom margins.

Printer Routing

Configure printer routing with options by pressing "P" for Printers;

```
PRINTER MAINTENANCE 4.8.05K209

--Name-- --Type-- --Destination File-- -----Comment-----
1 print_it EPSONFX \lpt1 send the output to the printer.
2 file_it NOCODES \hardcopy.txt send the output to a file.
3
4
5
6
7
8
9

Printer type: LQ2550 1

Press ESC -Record, Ctrl-C -Cancel, F10 -Help
      F6 -Set default printer
```

New in version 5.8.03

New interface that now allows managing up to 99 printers.

- Insert, Delete, Move printers
- Search by printer name or number
- Follow the prompts for the available functions

New in version 6.0.00

When browsing printers in Options for output formats, filePro now shows only valid printers. Pressing F6 again will display the complete printer and printer type list as it did in previous versions. F6 will toggle between the two lists

Printer Maintenance Options

Name	A unique phrase or number to help the users of the System identify the printer. The name can be up to sixteen characters and may not contain embedded blanks. Examples: printer3, joes, acctdept.
Type	The name of the print code table used by the printer. If you're not sure of the name, press F6 to select a name from a list of existing print code tables. This column must be filled in for each printer.
Destination File (Windows)	Type in the device name of the printer or a filename. If you leave the "Destination File" column blank, filePro will use the device name PRN". Examples: COM1, LPT1, PRN or a filename such as OUTPUT. Version 5.0 - Enhanced to use Windows's printer spooling. Examples: WIN:LPT1, WIN:PRN, WIN:\\machine_name\printer_share_name. Examples: PDF:filename, PDF:[open], (See PDF Printing) Version 5.8 and higher
Comment	You have a maximum of 80 characters for comments.
Destination (Linux/Unix)	Type in the command that the printer output will be sent to. Example: lp -dprintname -s where "printname" is the printer name that the operating system uses. If the line starts with a >, the rest of the line is the name of a file to send printer output. Examples: >/dev/lp01, >/tmp/output If the destination file is left blank, filePro will use a default print spooler command, typically: "lp -s". Examples: PDF:filename, PDF:[open], (See PDF Printing) Version 5.8 and higher

NIX Warning:
Report will crash if lp destination does not exist. The problem has to do with how the operating system handles pipes and the fact that Joe's printer command was a composite of lp and cat. There does not appear to be any way to trap a valid pipe error when lp is used. We can set SIGPIPE to ignore errors to test ermo after a write, however ermo is being set even if the pipe is valid. As a result the actual write to lp is performed in any case and does not crash until it gets to fflush. At fflush we should be able to turn off SIGPIPE and test fflush, but apparently, on Linux anyhow, the fflush function sets it back to default and it will SegV. So it's very unlikely we can do anything about this. If lp is not valid - beware.

Windows Spooling

Spooling for Native Windows filePro Windows doesn't spool print jobs, sent by native windows console applications to local printer ports, the same way that it does for MS-WINDOWS programs. (That is, open "lpt1" as a file and write to it.) We have added the necessary code to the native windows version of filePro to use the Windows printer routines (ie: OpenPrinter, StartDocPrinter, etc.) which do respect the Windows spooler. However, the spooler is also limited to those printers defined in the printer control panel. Therefore, we have made it a requirement that, in order to use the Windows spooler, you must prefix the filePro destination with "win:", as in "win:lpt1:". The rest of the destination must be the exact port name or printer name as you have defined it to Windows. So, if you have a printer attached to LPT1 that is named "HP DeskJet 870Cse", you would use either: win:lpt1: or win:HP DeskJet 870Cse If you have a network printer "\\server\printer" that is captured to LPT2, called "Bob's printer", and the Windows destination is "\\server\printer" then you would use either: win:\\server\printer or win:Bob's printer. You could not use "win:lpt2:" as "lpt2" is not the destination that Windows knows the printer by. (Though you could use "lpt2" without the "win:" and go directly to that port without the spooler.)

Remember: You can only use the exact port name or printer name that Windows uses. Anything else will result in a "the parameter is incorrect" error when filePro tries to open the printer.

FP2RTF

The files in the fP2RTF install shield are used to setup filePro to print through Rich Text File (RTF) printer drivers. This provides the capability to print standard filePro reports with little or no modification in Rich Text format to "Windows Only" printers.

Prerequisites and Limitations

- fP2RTF is LIMITED for use with character-based filePro in the Windows environment and is NOT intended as a solution for graphic-based filePro e.g. fileProGI or *NIX systems' printing.

- filePro version 5.0.9 or later is required to use fP2RTF.

- Maximum Report Widths for 8.5" x 11" paper

- Portrait Printing:

10 pitch - 80 characters

12 pitch - 96 characters

condensed - 132 characters

- Landscape Printing:

10 pitch - 100 characters

12 pitch - 120 characters

condensed - 168 characters

Environment

PFPOSTPRINT: The Rich Text drivers rely on pfpstprint being properly set. This variable should be set in startup batch file for filePro.

PFPRTC: Identifies the Rich Text printer driver to be used. The RTF66 driver will allow you to print reliably for most applications and should be set in your startup batch file.

PFPRT: Identifies the output filename to use for fP2RTF. This variable is normally set by the "~\fp\fp2rtf\getname.bat" file.

Example

```
set pfpstprint=runbatch.bat
```

```
set PATH=%PFPROG%\fp\fp2rtf;%path%
```

```
set PFPRTC=rtf66
```

```
call %pfprog%\fp\fp2rtf\getname.bat
```

The above lines need to be in your startup batch file e.g. fplus.bat to implement fP2RTF. This is automatically done for you if you have used the fP2RTF install shield and use a standard startup batch file name e.g. "fplus.bat", "hcf.bat", etc. If you use a batch file with a name other than one of the standard batch file names, add the above lines near the end of your startup batch file and before the line that executes the p.exe menu name. Although you can set pfpstprint=Atlantis.exe, the above method is preferred since it will prevent sharing violations in a network environments and cleanup temporary files.

Atlantis Editor: Although other editor programs can potentially be used for viewing and printing the output, Atlantis should be used to ensure that the output is properly formatted and printed from filePro when using the filePro Rich Text drivers. Atlantis provides for proper margin and page control unlike other Rich Text editors/viewers.

fP fonts: ftype1a.ttf and ftype1b.ttf fonts have been specifically designed for fP2RTF.

Although other fonts can be potentially used, fP Technologies Inc. doesn't guarantee that they will be properly rendered or printed with fP2RTF.

Installation and Setup

Run setup.exe - This will install the fonts, make the fonts immediately available to Windows, will install the fP2RTF printer drivers and will create the reserved directories required for fP2RTF e.g. ~\fp\fp2RTF, ~\fp\fp2RTF\tmp.

Modify startup batch file - Modify your filePro startup batch file to include the following lines if it has not been automatically updated by the install shield.

```
set pfpstprint=runbatch.bat
```

```
set PATH=%PFPROG%\fp\fp2rtf;%path%
```

```
set PFPRTC=rtf66
```

```
call %pfprog%\fp\fp2rtf\getname.bat
```

Note: The above lines should be added near the end of your filePro startup batch file and after any other "set path=" line. "Sample.bat" is provided in the root of the source media as an example.

Note: File "~\fp\fp2rtf\Getname.bat" invokes the "RTF66.PRT" driver for development since filePro defaults to 66 lines for hardcopies created by "Define Files", "Define Processing", etc. You can change the "pfprtc" variable to use the "RTF60.PRT" driver for runtime applications if your dominant report format is 60 lines. Either driver can accommodate the less dominant report format by including the applicable print code in the report or as an initialization print code.

Add fP2RTF Printer line - You should also add a printer line to your filePro configuration so that it can be selected when using the -PQ printer flag. Use the "Printer Maintenance" option from the filePro Plus main menu, and then "P" for printers.

Examples:

Name Type Destination Comment

RTF66 RTF66 report.rtf To print 66 line RTF reports

RichTxt RTF60 report.rtf To print 60 line RTF reports

Troubleshooting fp2RTF

Reports run but are not displayed in the Atlantis viewer - This is due to the way some systems handle font registration. Although we have made every effort to make our fonts immediately available without restarting your system, rebooting is sometimes required to read the font locations from your system registry. If reports are not displayed after rebooting your system, contact the applicable technical support team.

%fuser%%reportn%.rtf created instead of a report with the actual user name - You are running out of environment space so the required variables are not set. Increase your initial environment space by going to "icon properties" and then memory. Recommend setting initial environment to a value of "4096".

Another method for gaining environment space is to add a line near the beginning of your batch file used to start filePro e.g. "fpplus.bat". Add a line to shorten your path when running filePro. This will not affect other applications since the original path is restored when exiting filePro.

Example:

```
set path=c:\windows;c:\windows\command
```

Lines do not wrap properly - Reports are displayed but the lines do not wrap properly.

You are probably attempting to use an old version of filePro. Version 5.0.9 or later is required. Check your filePro path variables to make sure you are accessing the correct version of filePro.

Report Lines or Boxes are shown with dashes "-" and "+" - The fonts supplied with fp2RTF are either missing or not properly registered. Restart your system and then re-install the fp2RTF update. As a last resort, you can manually install the fonts using your control panel "fonts" option. This will install the fonts to your windows system directory. The fonts are available on the source media under directory ~/fp/fp2rtf as "fPtype1a.ttf" and "fPtype1b.ttf", etc.

Copy Print Code Example

Many printers use the same sequences to control common functions such as "Underline ON", Underline OFF", "Boldface ON", etc. You can quickly create a new printer driver by copying a generic printer driver like the EPSONFX (which applies to a narrow carriage printer like the EPSON Model FX80) to a table for your EPSON letter quality printer (like the EPSON LQ2550 132 column letter quality printer). Once the table is copied, you can modify the table by changing and adding sequences of codes to create a printer driver tailored to your specific printer model and then access the extended features for printing such as printing in letter quality mode and 132 column mode. For the sake discussion, we will copy the EPSONFX print driver in the list box to a new table labeled "LQ2550";

Number	Space	Sequence	Description
1			Boldface on
2			Boldface off
3			
4			
5		\$!b - 1	Underline on
6		\$!b - 0	Underline off
7		\$!b W 1	Elongated on
8		\$!b W 0	Elongated off
9		\$!b E	Emphasized on
10		\$!b F	Emphasized off
11		\$!2 \$!b P	Set 10 pitch
12		\$!2 \$!b M	Set 12 pitch
13		\$!bOf	Set 15.7/17.1 pitch
14		\$!b p 1	Set proportional on
15		\$!b p 0	Set proportional off
16		\$!b 4	Set italic on

Printer type: EPSONFX
Copy EPSONFX to...
Enter printer type: LQ2550

The above table contains "Escape Sequences" that are identical to the sequences required for the EPSON LQ2550 printer except for the added features pertaining to letter quality printing. The sequences are usually provided in the printer manual and referred to as "ESC Sequences" and "Control Codes". The following tables apply to the LQ2550 printer.

Commands in Numerical Order

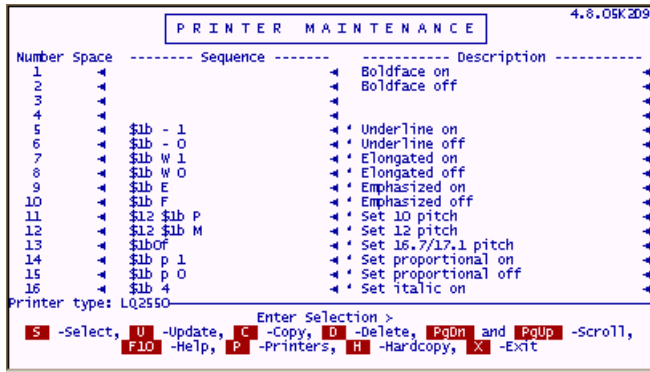
The following list shows control codes and ESC sequences (with their decimal and hexadecimal values), and the page number where the complete command description can be found.

ASCII	Dec.	Hex	Description	Page
BEL	7	07	Beeper	8-12
BS	8	08	Backspace	8-19
HT	9	09	Tab horizontally	8-20
LF	10	0A	Line feed	8-14
VT	11	0B	Tab vertically	8-17
FF	12	0C	Form feed	8-13
CR	13	0D	Carriage return	8-12
SO	14	0E	Select double-wide mode (1 line)	8-25
SI	15	0F	Select condensed mode	8-24
DC1	17	11	Select printer	8-8
DC2	18	12	Cancel condensed mode	8-25
DC3	19	13	Deselect printer	8-9
DC4	20	14	Cancel double-wide mode (1 line)	8-26
CAN	24	18	Cancel line	8-12
DEL	127	7F	Delete character	8-9
ESC SO	14	0E	Select double-wide mode (1 line)	8-25
ESC SI	15	0F	Select condensed mode	8-24
ESC EM	25	19	Cut sheet feeder mode	8-10
ESC SP	32	20	Set intercharacter space	8-30
ESC !	33	21	Master Select	8-22
ESC #	35	23	Cancel MSB control	8-11
ESC \$	36	24	Set absolute print position	8-19
ESC %	37	25	Select user-defined set	8-33
ESC &	38	26	Define user-defined characters	8-32
ESC *	42	2A	Select graphics mode	8-35
ESC (-	40	28	Select Line	8-29
ESC +	43	2B	Select n/360-inch line spacing	8-16
ESC -	45	2D	Turn underlining on/off	8-29

ASCII	Dec.	Hex	Description	Page
ESC U	85	55	Unidirectional mode on/off	8-10
ESC W	87	57	Turn double-wide mode on/off	8-25
ESC Y	89	59	High-speed double-density graphics	8-35
ESC Z	90	5A	Quadruple-density graphics	8-35
ESC \	92	5C	Set relative print position	8-20
ESC a	97	61	Select justification	8-30
ESC b	98	62	Set vertical tabs in channels	8-17
ESC g	103	67	Select 15 cpi	8-23
ESC k	107	6B	Select typestyle family	8-22
ESC l	108	6C	Set left margin	8-18
ESC p	112	70	Turn proportional mode on/off	8-24
ESC q	113	71	Select character style	8-29
ESC r	114	72	Select printing color	8-27
ESC t	116	74	Select character table	8-31
ESC w	119	77	Turn double-high mode on/off	8-26
ESC x	120	78	Select Letter Quality or draft	8-21

Notice that the sequences can be represented in 3 different ways in the above example i.e. ASCII (ESC Sequences), Dec (decimal values) and Hex (hexadecimal values). filePro provides the flexibility to use any of these representations as a sequence when defining the filePro printer driver.

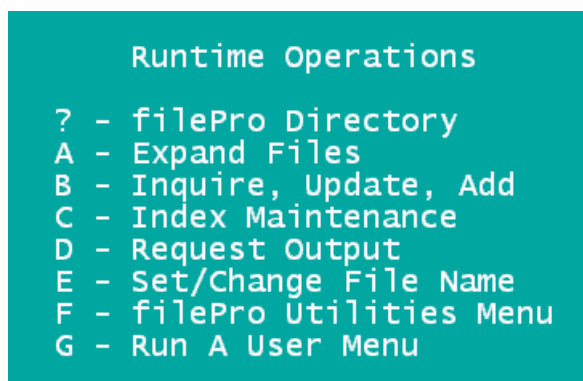
After you have created the preliminary version of the new printer driver for the LQ2550, you can update the driver to add or modify print codes by pressing "U" for Update.



Tricks: Extend the number of defined printers from 9 (available in this maintenance editor) to as many as you need, by adding the additional printers in the filePro "config" file (found in the "fp/lib" directory). These printers must be added sequentially as printer10, printer11, printer12 and so forth. You can not skip a number in the sequence. If you do skip a number above printer10, that is as far as filePro will count, to the skipped number and no further. You will not be able to access the other printers.

Runtime Operations

The right side of the filePro Main Menu contains Runtime Operations.



You can look at the "Runtime Operations" as those options that you will make available to your end-users. You will typically create user menus to control where what your end-users are able to do, so the above menu would not normally be used exactly as depicted but does help to separate the filePro options for the developer in logical groupings i.e. "Creation Operations" and "Runtime Operations".

Prior to getting into a detail review of the following sections, I suggest that you browse through the topics to get an overview of what each of the options entails.

FilePro Directory - Option ?

Contents of this section

Description of filePro Directory Option

Precautions

filePro Directory Option Screens

[Flags](#) [Flags_Directory_ddir_dprodir_>Second](#) [[flags link](#)]

Description:

This program allows you to view the configuration of your filePro files.

Among other things, it provides file format information all in one place, delete utility for all filePro formats e.g. screens, selection sets, browse formats, etc. and a hardcopy file info function.

Precautions:

It is relatively easy to completely erase a database, so be careful when you distribute runtime versions of the directory program "dprodir". If the user needs to delete "lockfiles" or perform other maintenance furnished by "dprodir", create a maintenance menu for the specific options to prevent inadvertent deletion of the users data, or establish creation passwords which to prevent killing the files.

FilePro Directory Option Screens:

```
FILEPRO DIRECTORY 4.8.05K209
Enter File Name:
BOUNCE41 roladex
FPCUST STATES
FPINWC test
FPPROD test1
FPSTTAX testenv
INWC XINW
nonfp XTRAN
F7 To Hardcopy All File Names
F8 Select Files to Hardcopy
Or Press Ctrl-C To Exit.
```

Directory Information

```
FILEPRO DIRECTORY 4.8.05K209
File: FPCUST Records: 15 Record Length: 105
Extent Location Records
KEY C: 15
Automatic Indexes
Index Description
A - Field (2 ) Length ( 10) last name
B - Field (1 ) Length ( 4) customer number
Screens 0 1
Demand Indexes
Index Location Field Len
Processing Tables AUTO INPUT
File: FPCUST
Enter Selection >
H To Hardcopy File Info PgDn For Next Page
X Exit D Delete Formats Q Qualifier F File Selection
```

Qualifier displayed

```
FILEPRO DIRECTORY 4.8.05K209
File: FPCUST Records: 0 Record Length: 105
Qualifier: tst
Extent Location Records
KEY C: 0
Automatic Indexes
Index Description
A - Field (2 ) Length ( 10) last name
B - Field (1 ) Length ( 4) customer number
Screens 0 1
Demand Indexes
Index Location Field Len
Processing Tables AUTO INPUT
File: FPCUST Qualifier: tst
Enter Selection >
H To Hardcopy File Info PgDn For Next Page
X Exit D Delete Formats Q Qualifier F File Selection
```

The Delete Option

Selecting a file.

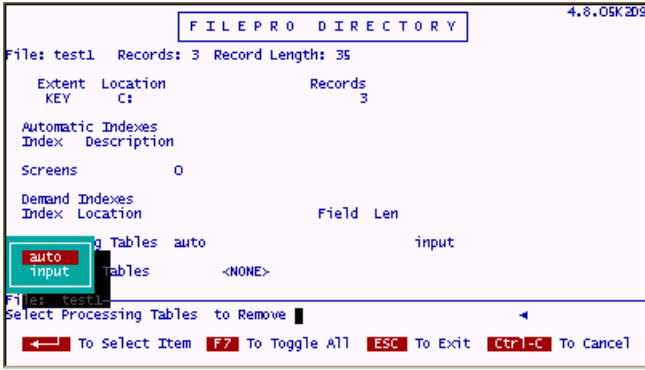


Selecting a delete option

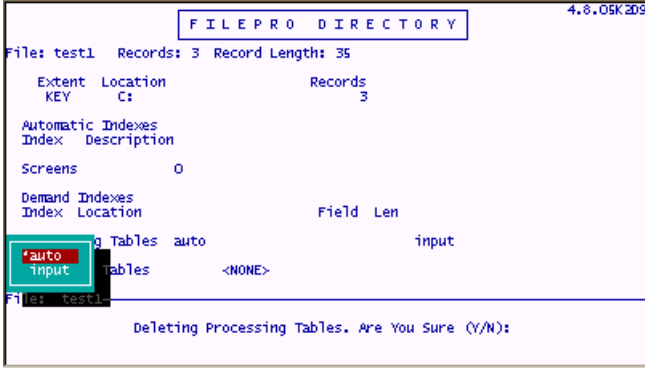
Delete Options



Selecting formats to delete



Confirming delete



Flags:

The filePro Directory (ddir) has flags to help you manipulate filePro databases from the command line or inside script or batch files.

Expand Files - Option A

Contents of this section

Description

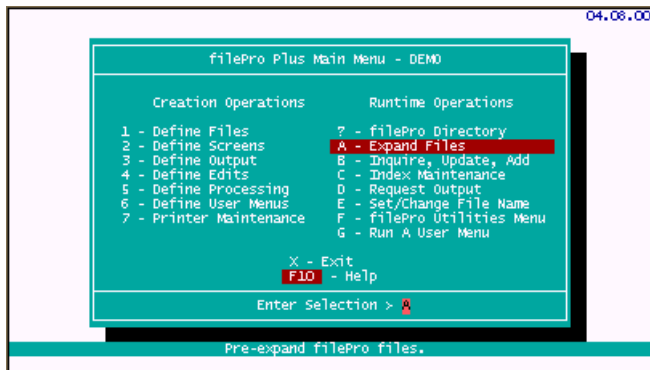
Extending Key Files [key1, key2, key3]

Key & Data Mismatch error

Description:

Expand Files lets you pre-allocate space for a file by adding unused records to it. This technique obviates the "free chain" under Unix. It also uses up disk space non-dynamically. This practice is not recommended under normal circumstances.

Select option A - Expand Files



Select a file from the list of filePro files.



Enter the number of additional records.



Extending Key Files (key1, key2, key3) :

Expand Files also lets you switch drives for new records being added to a key (and data) file. (You might want to do this when your data gets too large for the current filesystem.) If, at the "Number of Records to Expand File By" prompt, you type the word "switch", filePro will allow you to designate a new hard drive (filesystem). From this point on, all records in this file will be added to a file called "keyx1" (and datax1) in the same hierarchical path as the primary drive. If there already is a keyx1, filePro will add keyx2 and then keyx3. You are limited to 3 expansion filesystems for a total of four filesystems when considering the original "key" and "data" files. Extending "key" files is not recommended as normal procedure since it is better to move the entire key file to a larger drive. This feature is a holdover from earlier days when hard drives were very small. However, extended keys can be used for very large files when you have exceeded the maximum file size for the filesystem. Maximum file size is determined by the operating system/partitioning method.

Key & Data Mismatch error:

Expanding files by one record can sometimes fix a key/data length mismatch error. If a file that uses both a key and a data segment gets corrupted in a particular way, sometimes adding just 1 field from the Expand Files function will correct the situation.

Inquire, Update & Add - Option B

Contents of this section

Using Inquire, Update & Add

[IUA Flags](#)

Retrieving Records

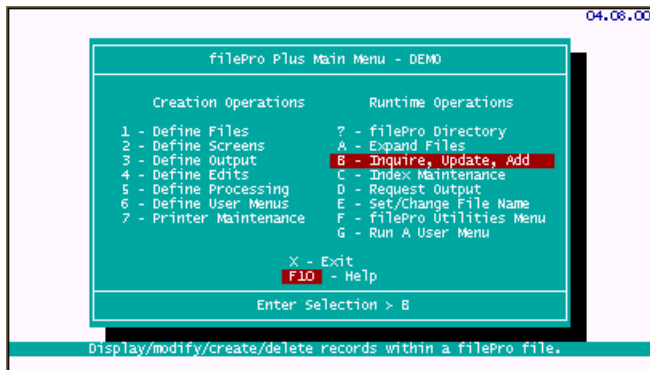
Selection Sets

Use of AND, OR

Field to Field Comparison

Associated fields

Using Inquire, Update & Add:



When entering Inquire, Update & Add (IUA) from the Main Menu, filePro presents you with the available files (filePro databases).



Multiple screens (named or numbered) are supported in filePro. When you enter IUA from the Main Menu, filePro displays all the "numbered" screens for your selection. (Named screens are reached through programming.)

Screen.0 is the default screen created for you by Define Files.



Once you have selected a file and a screen, filePro deposits you at the "clerk" menu. It gives you the ability to have all of your filePro files (and applications) have the same look and feel.



Retrieving Records

Records can be retrieved in a variety of ways through the "clerk" (IUA) menu.

Record number	Direct system maintained number.
Selection Sets	Criteria matching based on record content.
Indexes	Fast organized retrieval, alphabetic, numeric, date sorted indexes. (Combinations supported).
Browse	Screens/rows - Display records in row orientation instead of one full screen per record.
Fuzzy search	Excellent "sounds like" or "soundex" search.

Processing tables can be associated with IUA, making use of fields, text, calculations and data from any filePro file (or other system resource).

The normal behavior of IUA is to display a full screen of data for each record. At the users request, records can be shown in a line-oriented manner. This is called "browse mode" because 18 records are displayed on the screen at one time. Users can browse through these records using the up and down arrows and Page Up and Page Down keys to view many records very quickly. Once a desired record is found, the user moves a highlighted bar to that record and presses ENTER to go to a full screen view of that record. (The user may also press "U" while on the browse screen and this will not only take them to the highlighted record, but also put them in "update mode" on that record.)

The browse format of data display is easily designed. Any user can store tailored browse formats. Named formats can be modified quickly and resaved or just used temporarily. Not only the field names, but the lengths and edit types are available as you build a browse format. Unless specified otherwise, the browse format originally called to the screen is named "default". If there is no browse format named "default" then a format is composed of the field numbers in sequential order starting from field 1. In this case, only those fields that will fit on the screen are used.

Selection Sets

Selection sets allow you to pick criteria on-the-fly by which to retrieve records. These selection sets can be used in Inquire, Update, Add or in Request Output (printed reports and forms). By testing the real fields of a filePro record against your criteria (values, other fields, system maintained data, etc.) you can scan through the data and pick out only the records that meet the criteria on the selection set. These selection sets can be stored by any user by any valid name.

Adding lines of tests means the record must match every line of criteria. Giving any line a group designation and it becomes an "or" line or group.

The Selector Sentence allows you to use designated groups (and other selection sets) in virtually any combination of "and", "or" and "not" tests. Use parentheses liberally in a selector sentence for good clarity.

The lines in a group "or" with each other.

Groups always "and" with each other.

A plus sign (+) at the beginning of a group line makes them "and" instead of "or" with other members of the group.

Mass Update [F4]

When accessing records using an index or scan selection, you can enter "Mass Update" mode by pressing the F4 key from the IUA selection menu of a screen. This feature will remain active for the session until all records are updated or you exit the update session. This feature eliminates the need to press "U" to update each record and similar to using the "-N" menu flag for clerk.

Special Field-to-Field Comparison for Selections

The NNf Relationship

There is an additional relationship that does not appear in the normal scan prompts. This relationship tests one field against another. You might want to test if @cd eq @td on a group of records. (Was the record created "today" whenever "today" happens to be.)

Use the "NNf" operator to compare the contents of one field to the contents of another field.

NN stands for the relationship, i.e., eqf, gtf.

Sample

Field	Relationship	Value
1 (Title)	Eqf	2 (Artist)
11 (Qty)	Ltf	14 (Reorder Point)

Selecting with Associated Fields

Selecting an associated field can be done using the code such as "A)", which would search all other members of the associated group. If you select an associated field using the specific field number it will only search the designated field.

Usually the Equal or Contains relationships are used for selecting with associated fields.

Other relationships aren't often practical.

Sample

Field	Relationship	Value
b)	Co	day
a)	Eq	jazz

There are many flags to IUA (dclerk) which allow you to customize its operation. The flags can be used from a system prompt or on the " [Define User Menus](#) " action line when defining your menus.

Index Maintenance - Option C

Contents of this section

Description
Automatic Indexes
Demand Indexes
[Flags](#)
Index Comments

There are two kinds of indexes in filePro, Automatic and Demand.

AUTOMATIC

Automatic indexes contain every record in the file. Automatic indexes are constantly updated as you add, modify and delete records in a file, whether you do the modification from IUA or another filePro program.

DEMAND

Demand indexes can be built on a specified group of records. They do NOT have to contain every record in the file.

Demand indexes are static. They do not change regardless of what you do to the file. They can be built on ascending and descending sorts, and can make use of selection set criteria to build on only selected records.

Flags

There are [flags](#) to the indexing program that will allow it to run unattended as part of a script or batch file.

Example:

```
dxmaint filename -r -oA -e
```

This will rebuild the A index of the file "filename".

Index Comments:

You may put a comment on an index. This is a custom name for the index. Using comments will distinguish an index from the default name it is usually given, i.e., the map designation of the starting field of the index. This is very useful when you have an index sorted on multiple fields. Suppose that you have associated fields for e-mail addresses so that you can maintain more than one e-mail address in a rolodex file.

Number	Field Heading	Len	Type
1	name_last	15	lowup
2	name_first	15	lowup
3	aoaddress_1	15	allup
4	aoaddress_2	15	allup
5	city	10	allup
6	state	2	state
7	zip	10	zip
8	plphone_home	14	phone
9	plphone_business	14	phone
10	plphone_fax	14	phone
11	company_name	25	uplow
12	Notes	210	uplow
13	e1)email_address-personal	30	'
14	e1)email_adrrsss-business	30	'
15	e1)email_address-other	30	'
16			

File: rolodex Key
Press F1 for help, F10 to record, ESC to cancel,
F5 to go to field number, F6 for block functions.
Previous record length: 359 Current record length: 449

Without using comments for your indexes, it would be difficult to distinguish between an index built on "All e-mail addresses" and "personal e-mail addresses". This would be particularly true if you left the map description blank for the 2nd and subsequent occurrences of the associated fields. By using index comments, you can clearly identify which fields the indexes are built on.

You can also improve the aesthetic presentation of the indexes to the users when indexes are built on multiple fields e.g. name_last, name_first. The following screens are examples of a rolodex file with and without index comments.

Example without Comments:

Index	Field
A	name_last name_first
B	e1)email-personal
C	e1)email-personal

Press Ctrl-C To Exit.

Example with Comments:

INQUIRE, UPDATE, ADD

4.8.05K409 DEMO

- A - Last Name, First Name
- B - All E-mail Addresses
- C - Personal E-mail Address

Press **ESC** To Exit.

Version 5.8.01 - [Automatic Index Selection set](#)

Request Output - Option D

Description:

Request output can be run from the command line (a menu action line). It lets you run processes and/or printouts on any filePro file. There are many flags which can be used with the output program to customize its operation.

Request Output (dreport) can be run in interactive mode allowing for the following control.

Choose output format Sort and Select records in one choice by picking an index

or

Choose Sort order first, and then Select which records to output second

Tricks: Put a report into the background on-the-fly by pressing !g while the report is counting down through its records.

Set/Change File Name - Option E

Description:

This feature lets you set the filename to any existing filePro filename. This way you do not have to keep selecting the filename during all the various operations on the filePro Plus Main Menu since it is retained for you, and this prompt is automatically skipped.

Setting the filename is useful during development stages. For some people it is also useful during runtime operations. In most cases, you would take care of "setting the filename" by running user menus which have the desired filenames pre-packaged on each command line. This is a small step of automation in that direction.

Choose "E - Set/Change Filename" from the filePro Plus Main Menu and enter in the filename you want. Some programs will let you change it internally, but for the most part, this allows you to work with one file alone until you reset the name or enter no name with Set/Change Filename.

Note: If **PFNAME** is set, then the value of PFNAME will be show at the top of all menus/submenus. This allows the programmer to use PFNAME in manners not originally intended, but nonetheless is very useful. You could use PFNAME to display the qualifer being used.

In Unix, where all commands specify -m\$qualify by setting PFNAME=\$qualify.

In Windows where PFQUAL has been set, set PFNAME=%pfqual%.

Run A User Menus - Option G

Description:

You can use !menuname on the command line of a menu and that menu will be called. This is a faster route to the menu than using "p menuname" as the text of the new menu is simply read into place without executing the calling script again.

You can run any user menu from the Run a User Menu choice on the filePro Plus Main Menu (choice G). It is a normal practice to call up user menus from a users login script. Under Unix, this would be done in the users .profile script. Under Windows, this would be done in the personalized .bat file that calls up the desired filePro user menu e.g. fpplus.bat, fulldev.bat, user.bat, etc.

----- * * * * * General Info * * * * * -----

GIServer 6.0.1 no longer requires Java to run on any platform. Each binary is now compiled for a specific platform, removing the need for a JVM.

`lserv` is no longer required to run GIServer on *nix systems.

Passwords are no longer stored in the user's *.cfg file. Tools have been provided to make changes to user accounts.

Many configuration variables have been deprecated. Please check the configuration settings below to see what you may be able to remove from your configuration files.

Heartbeat messages are no longer required for GIServer 6.0.0+.

GIServer 6.0.1 can now take advantage of a new threading model and various network optimizations have been implemented (See changelog for more information).

Improved error and crash handling has been introduced to GIServer. On most fatal errors, GIServer will attempt to write a 'giserver_crash.log' file to the installation directory, and dump core where system settings allow.

- Programs provided by GIServer 6.0.1:
- giserver - The main application.
 - giaccount - Local administration module for on system changes.
 - gilioctool - Local license management/metric reporting.

giserver:
This is the main service application, responsible for handling account logins and authentication, as well as networking the messages between filePro, fileProGI, and fileProWeb.

On *nix based systems, the binary only supports the -ver/--version options.
giserver -ver

This prints out version and build information for giserver.

On Windows, an additional -sa option is supported. This allows the program to run in 'Stand Alone' mode, not requiring the use of a system service.

GIServer now has an alternate licensing mechanism required to enable fileProWeb. This is an online license server which enables flexibility for new licenses.

On first run, GIServer MUST contact the remote server to start (remote licensing only). After first contact, GIServer will drop into grace period if it cannot contact the server.

- giaccount:
Usage: giaccount option [arguments]
- m | --mode giaccount -m username (true|false)
Set administrator flag for account.
Requires current user to be root.
 - l | --list giaccount -l
List all accounts saved in the password file.
 - p | --passwd giaccount -p username [current_password] new_password
Change a user's password.
Requires current password if user is not root.
 - i | --invalidate giaccount -i username
Invalidate a user's password.
Requires current user to be root.
 - a | --useradd giaccount -a username password [template_user]
Add a new user.
Requires current user to be root.
Can take an optional template user for initial config values.
 - x | -d | --userdel giaccount -x username
Delete a user.
Requires current user to be root.
 - c | --convert giaccount -c [username]
Convert pre-6.0 GIServer passwords to new format.
Requires current user to be root.
 - v | --version Display version information.
 - h | -? | --help Display this help message

This is the local administration tool. Some options are only available as a superuser/system administrator.

Administration accounts can also be tied to a user account. This allows for multiple administrators per installation (One active connection).

gilioctool:
usage: gilioctool [-c <path> [-v] | -t <path> | -h]

- options:
- h, --hwid Generate device ID.
 - c, --check Check license file.
 - t, --transfer Transfer license to this machine (3 day minimum

between transfers).

`-v, --verbose` Enable verbose output.

`-ver, --version` Show version information.

`-, --help` Show this help message.

This is the license/metric reporting tool. It can be used to determine your device's hardware identification string, check the license file for errors, or to transfer your license to a new machine (remote licenses only).

To transfer a license:
Stop GIServer.
Run ``gilictool --transfer``.
Install GIServer on the target machine with your current license.
Start GIServer.

----- * * * * * Configuration * * * * * -----

Logging:

`logLevel=(off|severe|warning|info|config|fine|finer|finest)`, default info
Sets logging level from least->most verbose.

`logAdmin=(true|false)`, default false
Sets if admin interface events should be logged.

`log=(true|false)`, default false
Sets if any events should be logged.

`logSize=(integer)`, default 0 (infinite)
Sets the maximum log size before rotation in bytes (cumulative).

`logSizeKB=(integer)`, default 0 (infinite)
Sets the maximum log size before rotation in kilobytes (cumulative).

`logSizeMB=(integer)`, default 5
Sets the maximum log size before rotation in megabytes (cumulative).

`maxLogRotations=(integer)`, default 5
Sets the maximum number of log rotations to keep (0 is infinite).

General Configuration:

`useActiveDirectory=(true|false)`, default false
Sets if the server should authenticate using Active Directory or built in authentication.

`activeDirectoryDomain=(string)`, default is Undefined
Sets the active directory domain for authentication, e.g. `my_domain.com`.

`activeDirectoryServer=(string)`, default is Undefined
Sets the active directory server for authentication, e.g. `local.my_domain.com` or the IP address.

`serverName=(string)`, default is from system
Sets the name of the server.

`serverAddress=(string)`, default is from system
Sets the address for the server to listen on.

`idleout=(integer)`, default 0 (never)
Sets how long before a session is automatically disconnected due to being idle (in seconds).

`adminPort=(short)`, default 4450
Sets the port the admin interface can use to connect.

`admin=(true|false)`, default true
Sets whether the admin account is active for this server.

`remoteShutdown=(true|false)`, default false
Sets if remote shutdowns are allowed from the administrative interface.

`maxLoginAttempts=(integer)`, default 0 (infinite)
Sets how many failed logins (GIclient) before automatic disconnect.

`doesNotExpire=(true|false)`, default false
Sets if passwords expire or not (Can be set in the user's *.cfg file)

`exec=(string)`, default `fpdaemon`
Sets the path to the `fpdaemon` program for the server to use.

`restrictClientVersion=(true|false)`, default false
Sets strict mode for client version verification, prevents outdated clients from logging in.

`maxLoginsPerUser=(integer)`, default 0 (infinite)
(6.0.01.08) Sets the maximum number of sessions that can be in use by a single user at once.

`expireWarnDays=(integer)`, default 7
(6.0.01.11) Sets the number of days prior to expiration that the end

user will be warned. Max value 120 days.

heartbeatInterval=(integer), default 60

(6.0.01.12) Sets the number of seconds before a wakeup message is sent to the client application. Setting this value to 0 will disable the feature.

Password Configuration:

minimumLength=(integer), default 5

Sets minimum length of a password.

maximumLength=(integer), default 48

Sets maximum length of a password.

expireDays=(integer), default 0 (does not expire)

Sets how many days until a password is marked as expired. Overrides general configuration.

reuseDays=(integer), default 0

Sets how many days until a password can be reused.

reuseInstances=(integer), 3

Sets how many password changes until a password can be reused.

requireUpper=(true|false), default false

Sets if a password requires at least one uppercase letter.

requireLower=(true|false), default false

Sets if a password requires at least one lowercase letter.

requireNumeric=(true|false), default false

Sets if a password requires at least one number.

requireSymbol=(true|false), default false

Sets if a password requires at least one symbol.

complexityErrorMsg=(string)

defaults to "Entered Password is not sufficiently complex."

Sets the error message to display on a complexity check failure.

reuseErrorMessage=(string)

defaults to "Too soon to reuse this password."

Sets the error message to display on a reuse failure.

GIserver server configuration file format.

The configuration files can take advantage of comments by prefixing a line with either '#' or 'REM'.

There are also a few tags that can be used to augment the behavior of these config files.

[GLOBAL]

This is the default tag. It suggests to GIserver to include this variable for all client types.

[GI_ONLY]

This tag suggests to GIserver that it only sets this variable for a fileProGI client.

[WEB_ONLY]

This tag suggests to GIserver that it only sets this variable for a fileProWeb client.

A blank line will also reset the tag to GLOBAL

NOTE: These tags are ignored in the 'giserver.cfg' and 'password.cfg' files.

Example:

```
# set startcmd to open the menu 'mainmenu_gui' if launched with fileProGI
[GI_ONLY]
startcmd="p mainmenu_gui"
```

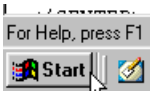
```
# set startcmd to open the menu 'mainmenu_web' if launched with fileProWeb
[WEB_ONLY]
startcmd="p mainmenu_web"
```

```
# these are in the global namespace
# the space between the last [WEB_ONLY] tag and the next line reset the
# namespace
somevar1=definition
somevar2=definition
somevar3=definition
```

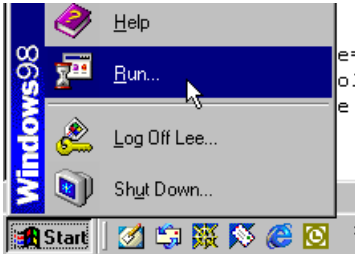
Installation of fileProGI Runtime

fileProGI is a Windows® only graphical interface to filePro. It comes on a CD complete with an Install Shield® that will step you easily through the initial installation of fileProGI Runtime.

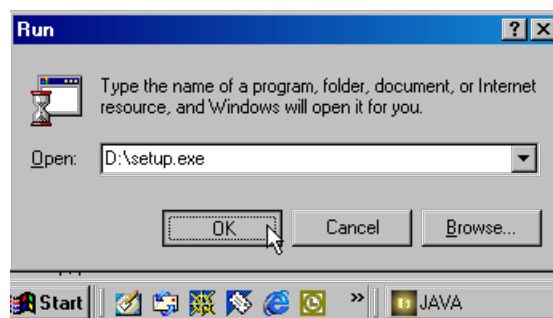
- Insert the fileProGI Runtime CD into your CD-ROM Drive and Click on the **Start** icon at the lower left hand corner of your screen:



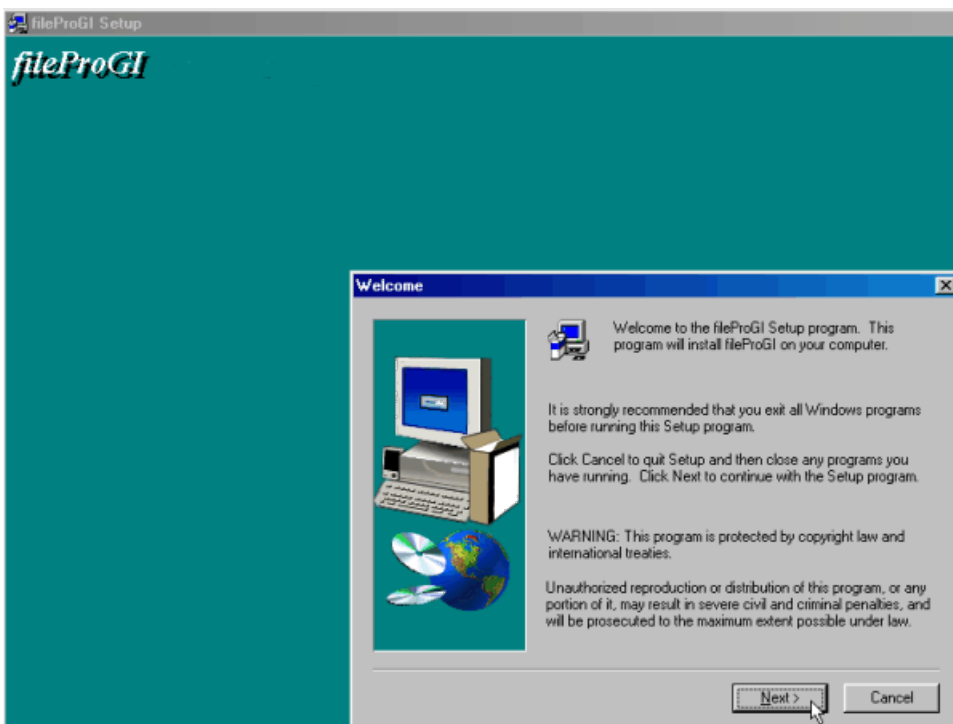
- and then Select **Run**:



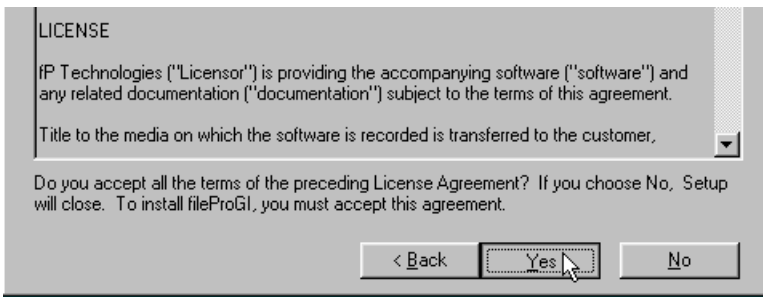
- When the Run dialog box appears type in X:\setup.exe where **X** is the drive letter of your CD-ROM Drive [example shows the D Drive] and then click on the **OK** button:



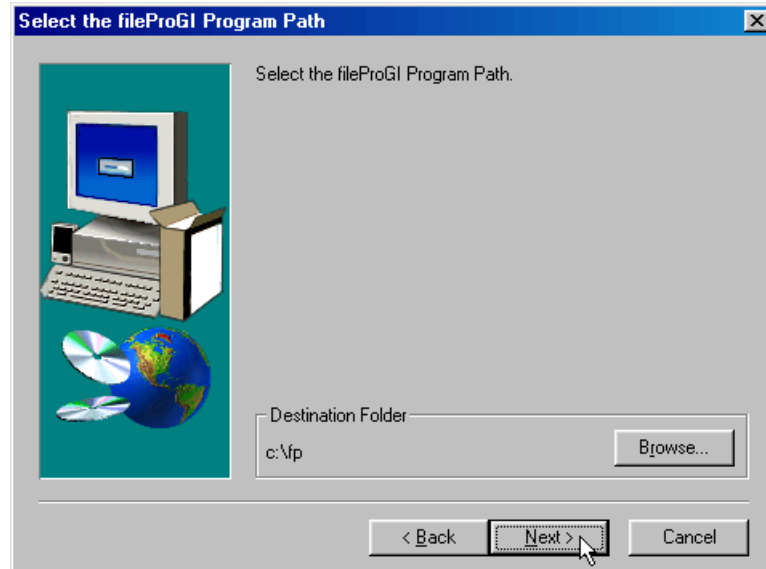
- The initial fileProGI Runtime installation screen should appear. Click on Next to continue:



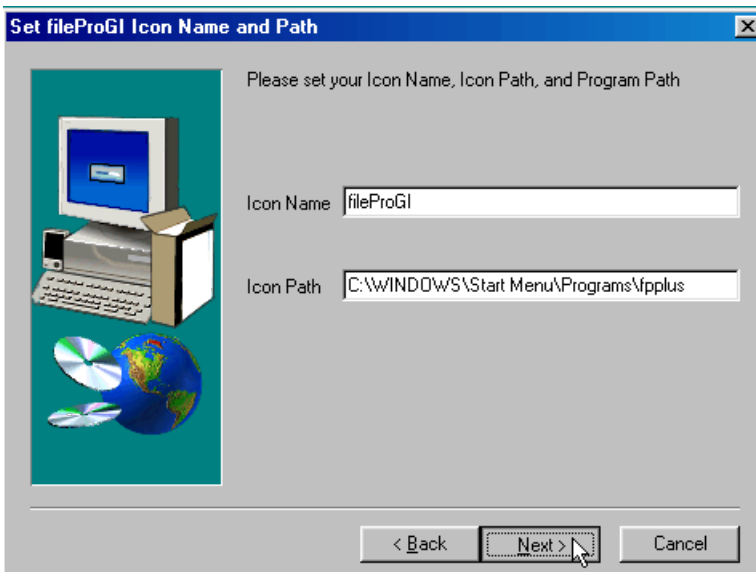
- Read and acknowledge the license agreement and click on the **Yes** button if you agree to the terms:



- Select the folder you want to install the fileProGI folder under. You can hit the Browse button to specify a different folder than the default. When you have selected the folder you want the **Next** Button:

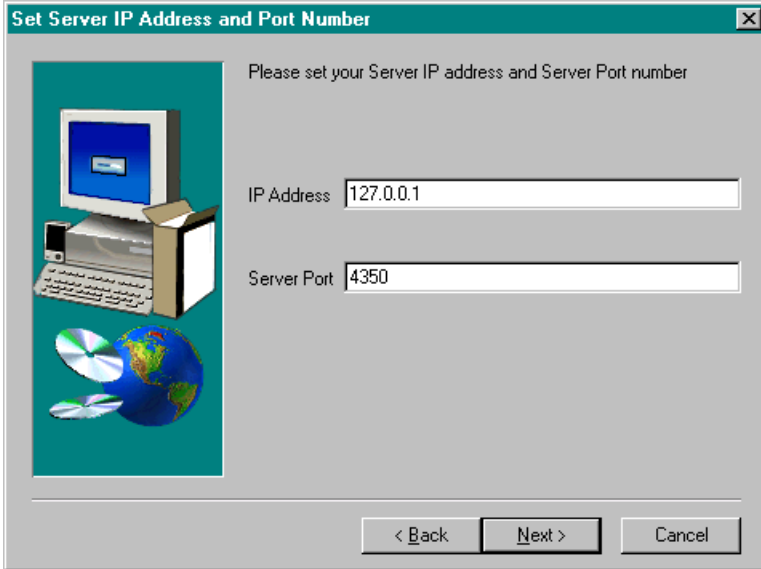


- Select the icon label you want and the location to store the icon and click on the **Next** button to continue:

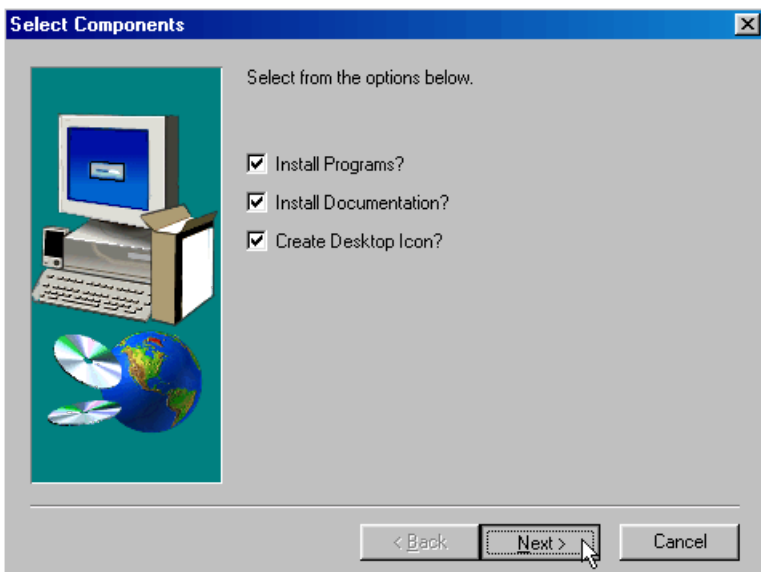


- Enter the IP Address and Server Port Number to use and then click on the **Next** button to continue.

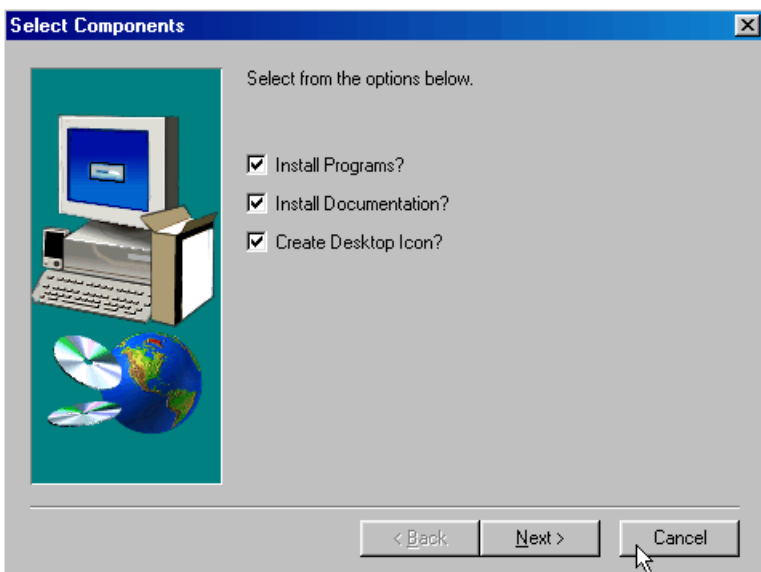
Note: If you are upgrading, the previous Server IP Address and Server Port Number will be provided.



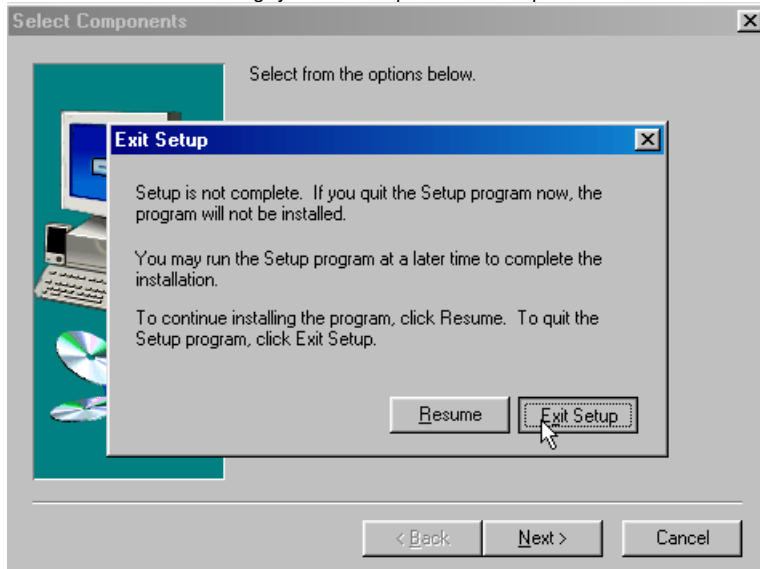
- Select the components you want to install and then click on the **Next** button to continue:



- If for any reason you want to cancel the installation at this point, click on the **Cancel** button:



You will be asked to acknowledge your desire to quit the installation process:



- If you elected to continue with the installation, the installation will proceed and an installation complete screen will appear.

Note: If you are upgrading fileProCl, you will be prompted before overwriting some files. If you are unsure as to whether to overwrite the files, the safest method is answer "**No**" to these prompts.

Launching fileProGI

- To launch fileProGI, simply click on the fileProGI icon that the installation process put on you windows desktop:



- The initial Log on screens will appear:



Log on to Gserver/filePro

- Before you log on for the first time, you must make sure that fileProGI is pointed to the right IP address and port number of the Gserver/filePro server. Click on the **Setup Configuration** button and set the connection options.



- Make sure that the IP address and port number match the Gserver's IP address and port #. The example shown is for a local Gserver installation and then click on the **OK** button. You will be returned to the **LOGON** screen.

Provide Configuration

Identify Server

Connection Type
 Network Dial-up Com Port

NetBIOS Name
or IP Address:

Network Port:

Phone Number:

Modem:



-
- If you decide not to Log on, click on the **Exit** button:



- If you want to log on to fileProGI, click on the **Log On** button
- The **User ID & Password** screen will appear. Enter your assigned **User ID** and **Password** and either hit the Enter key or click on **OK**



- The fileProGI Runtime Menu will then appear:



Note: If the fileProGI Runtime Menu does not appear, you probably have a wrong IPaddress /Port number or Gserver has not been launched. Launch Gserver (noting any errors) and try launching fileProGI again. If the problem persists, check your connectivity by pinging the applicable IP address or refer to troubleshooting procedures for Gserver.

Introduction To fileProODBC (not included in filePro Lite)

Welcome to the fileProODBC Manual. This resource has been developed to satisfy a wide range of audiences ranging from new filePro users to our loyal filePro filePro experts. This manual is intended for use in conjunction the fPmanual and so should be considered as an addendum. If you are new to filePro, refer the **How Do I ?** and **filePro Menu Options** topics in the fPmanual which serve as guides to get you started. For the experts, see **Advanced Concepts** and **Reference** topics in the fPmanual. By focusing on practical usage of the language, these guides are intended to familiarize you with filePro and fileProODBC at each level.

What is fileProODBC? fileProODBC allows you to access and maintain ODBC database records directly from filePro. This provides for quickly integrating filePro applications with databases developed with other application design tools such as Microsoft Access, SQL 2000 Server, Oracle, Sybase and many other ODBC compliant databases. The fileProODBC module provides for High-Level access to ODBC databases by simply using the " Define Files " option and Low-Level access using processing tables. Both methods are integrated into existing filePro design tools and each has inherent benefits that should be considered based on your requirements. The High-Level method is simpler to use but has some restrictions while the Low-Level method is intended for advanced users and is less restrictive.

Revision Date: 11/03/2003

System Requirements - fileProODBC

Windows NT 4.0, Windows 2000, Windows XP, Windows 2003, or later.

Minimum RAM 48 MB


Disk Space: 10 MB

The minimum RAM requirements for Windows will usually allow you to run filePro applications satisfactorily. Additional RAM may be required for Peer-to-Peer Network Servers to achieve acceptable performance for filePro if the machine designated as the filePro file Server is also used as a workstation and/or provides other services such as printing, FAX services, etc.

Foreword

Using Help

There are many **hyperlinks** in the fileProODBC Manual. They will stay on your screen until you press ESCAPE, press ENTER, or click the left mouse button. There are also many **hyperlink jumps**. These are secondary HELP screens that will appear on top of your primary window (depends on the Help version you are using). They have scroll bars and look very much like your primary HELP window. Read them and then close the **jump** window as you normally would close any secondary window. Do not lose your place on the primary topic by maximizing one of these secondary windows. It is also important to understand that these **jumps** are usually extra reference material and may be much more advanced in content than what you are studying. If this is true, just close the window and keep reading where you are. You can always find these resources later.

To go to the next sub-topic within a topic, use the >> button in the Help Menu bar. To go to previous topics, use the << button. If one of the buttons is not lit up, you are at one end or the other within that topic and should probably press Help Topics to choose the next (or previous)  topic.

Viewing:

The text quality and size is dependent on your system video configuration and various other factors. You can change the font size by selecting "Options" and then "Font" to change to a larger or smaller font.

Disclaimer

This documentation is distributed by fP Technologies Inc for your use with licensed copies of filePro and may not be distributed except as covered by the filePro license agreements. In using this documentation, you assume all risks arising from the use of this documentation. fP Technologies Inc or its suppliers are not liable for any damages (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or other loss) arising out of the use of or inability to use the documentation.

Installation

The installation contains replacement filePro executables that will replace your existing files if you are currently running a non-ODBC version of filePro. We recommend that you always copy or save your old files before installing any upgrade.

DLLs are also needed for the ODBC version of filePro. While there are numerous places where these could be placed, the typical location is either "C:\windows\system" or "C:\windows\system32". The install program will determine the applicable system directory and copy the required files to the right location. If the files already exist, the date of the files is compared to determine if files should be overwritten on your system. Even though the files we furnish may be newer than your files, the install program will prompt you to confirm "overwrites". We recommend that you always replace older versions of the files.

When installing fileProODBC, a new ICON is created in your Programs folder and on your desktop (unless you elect not to create the desktop ICON) for you to access fileProODBC.

Finally, we provide some sample ODBC filePro applications for you to test drive and to become familiar with fileProODBC. These will be copied to the specified \filepro directory on your system. A new reserved directory name of \fp\fputil\filepro is created by the install to store the sample database files and a cool index utility process. The filePro applications are copied to the specified \filepro folder. Refer to the "fileProODBC.hlp" file on how to apply ODBC to your applications.

Note:

Again, if you decide to overwrite your existing filePro programs, you should do one of the following before installing fileProODBC.

- Make a backup of your existing fp directory.
- Duplicate the fp directory tree into another directory (keeping the "fp" basename) and overwrite the programs there. Then, point PFPROG to the new directory when testing the ODBC version.

What Method Should I Use?

There are inherent benefits to using each method. Review the **Method Selection** chart to determine which method will better suit your requirements.

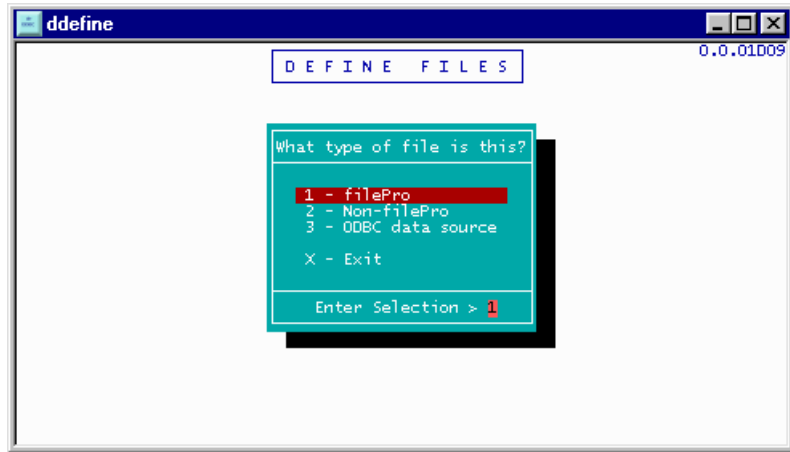
Method Selection	High Level	Low Level
Exchange data between filePro and ODBC data source files on an on-going basis. Method to select is dependent on programmer's level of experience with filePro. If you have used advanced programming functions such as arrays, understand use of subscripts and know a little about SQL, the Low-Level method should be used. Otherwise, begin with the High-Level method.		X
One-time exchange between filePro and ODBC data source or limited Query access to merge/compare data in your filePro files. High-Level method is very quickly implemented and will save you time.	X	
Integrate use of indexing with large ODBC databases that are updated by filePro as well as other applications developed in MS Access, SQL 2000 Server, etc.		X
Low-level is recommended since filePro's automatic indexes will not do the job for you. You will need to address the indexing requirements using SQL clauses such as ORDER BY and should create necessary table indexes on the ODBC data source as required.		X
Integrate use of indexing with ODBC databases of any size that are updated only by filePro but accessed by other applications such as MS Access, SQL 2000 Server, etc. for things like read-only queries, to produce charts and create reports.		
High-Level will work well here since you can take advantage of filePro's automatic indexes, " Define Screens " , " Define Edits " and other utilities and treat the ODBC data source just like a filePro file. Although this type of application can also be implemented using the Low-Level method, the High-Level method has the advantage here since you can more quickly implement this method and utilize all of filePro's advanced programming options to integrate access to the ODBC databases.	X	
Periodically exchange data between filePro and ODBC data sources for weekly or monthly reports where indexes are not needed.		
High-Level or Low-Level can be used and method selected is dependent on the size of the ODBC databases, how much updating activity there is on the ODBC data sources. If the ODBC databases are typically small and are not being updated when exchanging the data, e.g. nightly batch processing, use the High-Level method. If the data sources are large corporate databases or you need to exchange data while the source files are being updated by other than a filePro applications, use the Low-Level method.	X	X
Create new ODBC data source tables from filePro.		X
Use the Low-Level method since you cannot create new data source tables with the High-Level method but can only link to existing tables.		X

Using Define Files for ODBC (High Level method) (not included in filePro Lite)

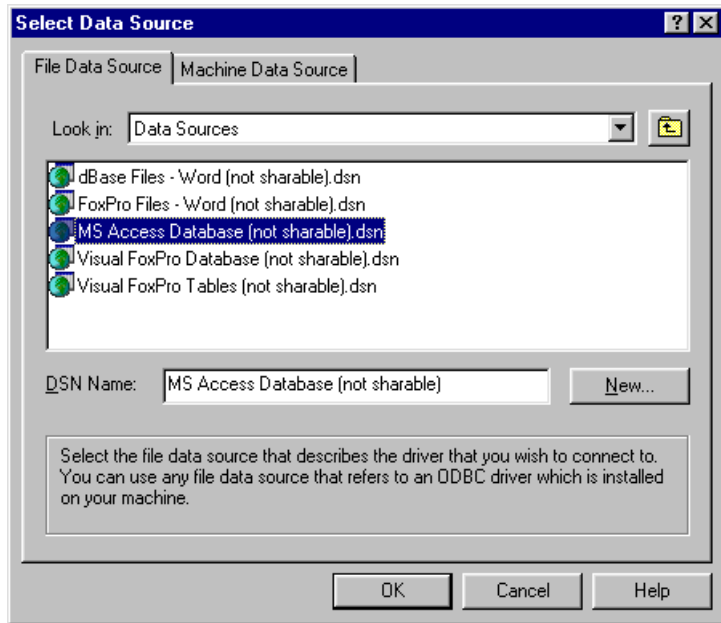
Use the " Define Files " option to create links to existing ODBC files. You cannot create a new ODBC file using define files but can only establish links to the data source with the " High Level " method. If you want to create an ODBC file, use the Low-Level method.

Using the " Define Files " option is the simplest method and should be considered by both new filePro developers and filePro experts as the first method. Take a look at the restrictions for the High-Level method and topic " What Method Should I Use ? " before selecting a method.

When using " Define Files " (ddefine), there is now a third choice, "ODBC data source".

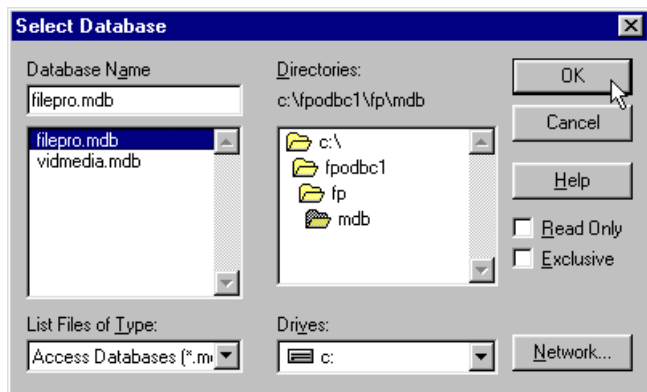


Upon selecting "ODBC data source", the standard Windows "select data source" dialog will appear:



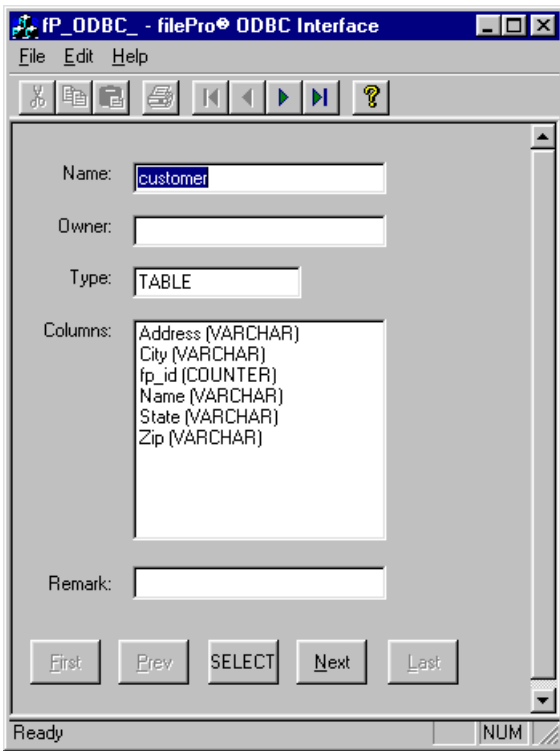
The first step is to select the type of source being used. (For example, "MS Access Database", or "SQL Server".)

Once the source type is selected, a type-specific dialog will appear, allowing you to select the actual source of the data. For example, if using an MS Access Database, you will be asked for the name of the .mdb file, using the standard Windows "open file" dialog:

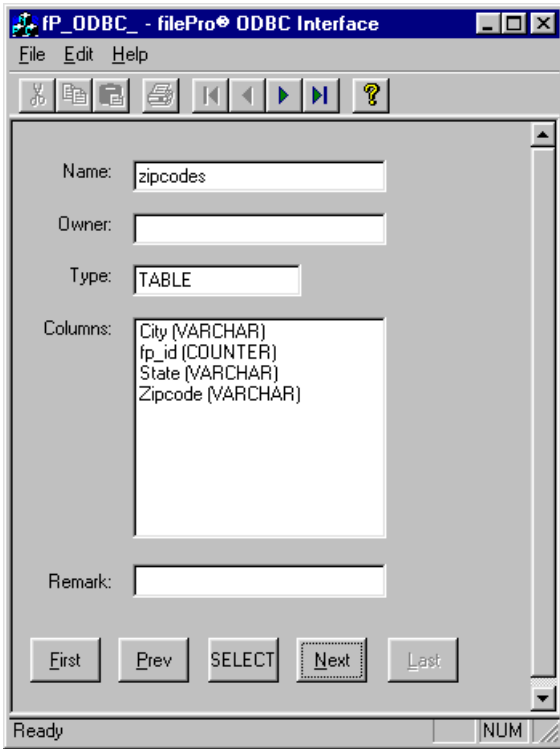


Other data source types may use other dialogs to select the data source.

The actual table within the data source must be selected. This is currently done with a Windows dialog that will either be redesigned, or pulled back into a filePro dialog.

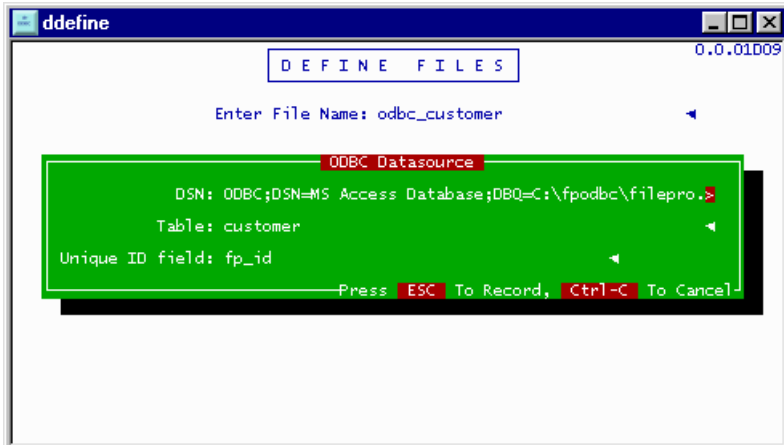


Click the First/Prev/Next/Last buttons to scroll through the available tables. Click SELECT when the proper table is shown.



At this point, the data source has been selected, and you are returned to ddefine's ODBC information screen:

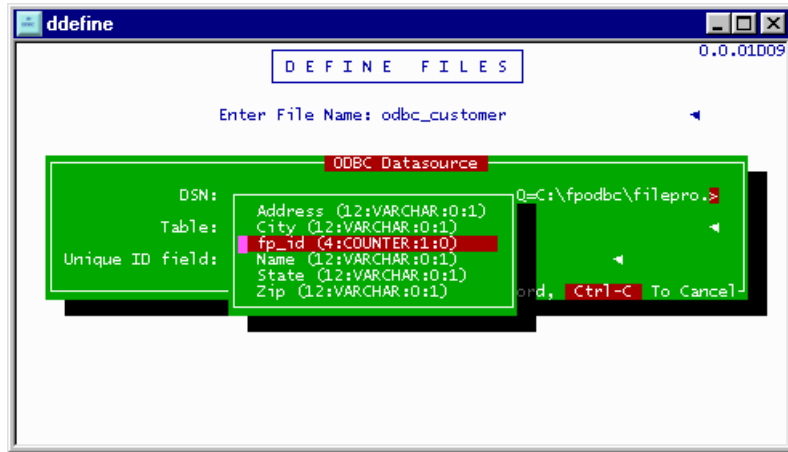
Note: Pressing F6 in the "Table" field will redisplay the table-select dialog, allowing you to select a different table from the list.



Note: Some combinations of Windows and video drivers cause parts of the screen to not be redrawn upon taking down the table-select dialog. This is being looked into. If necessary, press Alt+Enter twice to switch to full-screen text mode and back to the GUI desktop to force Windows to redraw the screen.

ODBC data sources have two parts, a "DSN" (short for "Data Source Name") and a table name. These are what you defined with the previous dialogs. filePro needs one more piece of information e.g. the name of the unique ID field. The best way to do this is to press F6 within the "Unique ID field" entry to bring up a list of fields within the selected table.

Note that the current version has extra information displayed beyond just the column name, mostly for debugging purposes:

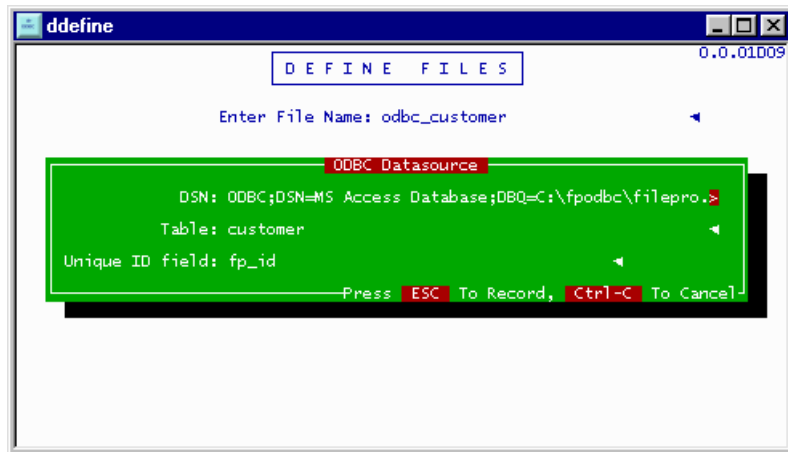


For the curious, the items within parens are:

- SQL data type, as a number.
- Data type, as a string returned from the ODBC database.
- The "autonumber" flag, as returned from the ODBC database.
- The "writable" flag, as returned from the ODBC database.

If you do not yet have a unique ID field specified, filePro will select any available autonumber field by default. It is important that you have a " autonumber " field available in the datasource since filePro will need this field to act as the pseudo @rn field. If there is no " autonumber " and you cannot assign one in the datasource, you should use the Low-Level method for ODBC files.

Once the unique ID field has been selected, save the ODBC datasource dialog to get to the standard ddefine field listing. Notice that filePro has already filled in the column names, lengths, and types:



The field names are not editable, as they are determined by the ODBC data source. The length and type are editable, and are initially filled in by filePro based on the information as returned from the ODBC table. While you can change the length and type to anything you want, you need to keep in mind the original ODBC data type. (For example, don't tell filePro to use an "MDY/" type if the ODBC type is "currency".) This allows you to determine, for example, which filePro date format to use when handling an ODBC date field, or to overlay a user-defined edit on top of a CHAR-type field.

filePro's ODBC objects (Low-Level method) (not included in filePro Lite)

To provide low-level access to ODBC data sources, filePro now includes ODBC objects. These are called ODBC_CONNECTION, and ODBC.

The ODBC_CONNECTION object (not included in filePro Lite)

The first part of creating an ODBC recordset is to establish a connection to an ODBC server. This is done by creating an ODBC_CONNECTION object:

```
handle = new ODBC_CONNECTION( [CONNECTION_STRING] )
```

where "CONNECTION_STRING" is the connection string for the server. If none is specified, or an empty value is provided, Windows will ask for the data source at runtime. Also, if insufficient information is provided in the connection string, such as not including the password for a password-protected dataset, Windows will ask for that information at runtime.

The return value is a handle to the new object. If the value is less than or equal to zero, the connection failed.

Once the connection has been established, this handle is used to create the ODBC recordset object.

Example connection strings:

```
Driver={Microsoft Access Driver (*.mdb)};DBQ=c:\mdb\filepro.mdb;
FileDSN=CustomerFileDSN;
```

Note that multiple recordsets can be created on a single connection.

To do: include a connection wizard.

To do: include a method for accessing the reason for failure.

The ODBC object (not included in filePro Lite)

The second part of creating an ODBC recordset is to create the ODBC object itself:

```
handle = new ODBC( connection_handle [ , query ] )
```

where "connection_handle" is the handle of the ODBC_CONNECTION object created earlier and "query" is the optional SQL query to be used to create the recordset.

The return value is a handle to the new object. If the value is less than or equal to zero, the recordset creation failed. (Note that, even if you specify an invalid query, the creation can still succeed. It will simply contain no data.)

If a query is not specified when the object is created, you must specify it later by using:

```
ODBC handle QUERY query [ DYNASET|SNAPSHOT ] [ BOOKMARKS ]
```

The optional DYNASET and SNAPSHOT flags force those modes to be used for the recordset. If neither is specified, the driver's default mode is used. The optional BOOKMARKS flag enables bookmark support, if supported by the driver. (Note that enabling bookmarks may slow down data retrieval, so don't use it if you aren't going to be using bookmarks on the recordset.)

A query must be executed prior to accessing any of the records within the recordset.

Note that you can replace the recordset within an ODBC object by executing another QUERY command. Doing so will close the existing recordset and create a new one.

Moving around within the recordset (not included in filePro Lite)

Once a recordset has been created, you can move around within it with the following methods:

```
ODBC handle GETFIRST
ODBC handle GETLAST
ODBC handle GETPREV
ODBC handle GETNEXT
```

These will move the current record to the first, last, previous, and next records in the set, respectively.

Note that you only have sequential access within recordsets. This is a limitation of ODBC.

Note that ODBC does not include a method of determining the number of records returned within a recordset. The only way to do so is to GETNEXT through the set, counting the records, until EOF is reached. If there are no records being modified on the table by another process, you can run a separate query with "SELECT COUNT(*)" using the same WHERE clause. (If the table is being modified by another process, there is no guarantee that both queries will return the same set of records.)

Note that with some ODBC servers, the GETLAST operation may take a long time on large recordsets. If you need to access the last record and scroll backwards through the recordset, it is recommended that you instead use an ORDER BY clause in descending order, and scroll forwards from the first record.

Bookmarks (not included in filePro Lite)

Although there is only sequential access through ODBC recordsets, it is possible (if the server supports it) to "bookmark" your current position and return to it later.

To retrieve a bookmark to the current position, you use the "BOOKMARK GET" method:

```
ODBC handle BOOKMARK GET bookmark
```

where "bookmark" is the variable in which to store the bookmark. This variable must be at least 16 bytes long, and of type "char".

If the bookmark cannot be made (for example, you didn't specify the BOOKMARKS flag on the query), the variable will be set to blank (ie: bookmark="" will test true.) In that case, @ODBCERROR[] will contain the error. If the bookmark succeeds, the bookmark variable will contain something other than all blanks. (ie: bookmark="" will test false.) The bookmark data is not in a human-readable format.

Once a bookmark has been made, you can return to that position within the same recordset at any time by using the "BOOKMARK SET" method:

```
ODBC handle BOOKMARK SET bookmark
```

where "bookmark" is the same variable used to GET the bookmark earlier.

Note that there is no guarantee that a bookmark will remain valid for anything other than the current recordset. Even executing a REQUERY may invalidate the bookmark.

Note that there can be multiple bookmarks per recordset.

Refreshing a recordset (not included in filePro Lite)

Not every ODBC data source can return a dynamic data set. Rather, they return a "snapshot" of the records at the time that the query was executed. (ie: if records are updated, added, or deleted while the recordset is open, you won't see those changes.) In order to get an updated snapshot, you need to re-execute the query.

Rather than requiring that your code keep track of the query used, you can simply "requery" the recordset using the same query as the current set:

```
ODBC handle REQUERY
```

You will be positioned back to the first record, if any.

Closing a recordset (not included in filePro Lite)

Once you are finished with a recordset, you can close it without deleting the ODBC object. (This allows future queries to be executed on the same connection, without the overhead of creating a new object.)

ODBC handle CLOSE

Once the recordset is closed, no more operations may be performed on it, aside from DELETE and QUERY .

New in filePro 5.8 is a 'timeout' function (not included in filePro Lite)

```
status = @ODBC.handle.TIMEOUT(timeout)
```

This sets the ODBC query timeout to "timeout" seconds. Only subsequent operations are affected.

As described by Microsoft:

> The default value for query timeouts is 15 seconds. Not all data sources support the ability to set a query timeout value. If you set a query timeout value of 0, no timeout occurs; the communication with the data source may stop responding.

The return value is the previous timeout value, or a null string -- "" -- if the data source does not return the old value.

Also, note that the data source may share the timeout value with all queries on the same connection, so if multiple ODBC handles are retrieved from the same ODBC_CONNECTION handle, setting the timeout on one may affect all ODBC handles on the same ODBC_CONNECTION.

```
xx = @ODBCERROR.CLEAR
```

This will clear the previous @ODBCERROR results

Accessing columns within the recordset (not included in filePro Lite)

Accessing the columns within a recordset is done with the new @ODBC [] system array, which is accessed using the syntax:

```
@ODBC.handle[subscript]
```

where " handle " is the handle to the ODBC recordset object, and " subscript " is the column number or name.

As with all system arrays, subscript zero returns the number of elements within the array, and accessing an out-of-range subscript returns an empty string.

If the number of elements is zero, then there are no columns available. This could be due to query returning no records, or positioning the current record to BOF, EOF, or a deleted record.

An enhancement from other system arrays is that " subscript " can be the name of the column, in addition to the numeric column number. For example:

```
@ODBC.handle["zipcode"]
```

This is especially important when doing " SELECT * " clauses on older ODBC servers, where the order of the columns cannot be guaranteed. (Or even on newer servers, where the database can have columns inserted within the existing structure, thereby changing the column number of the following ones.)

Note that there is currently only read-only access to the recordset.

Accessing column information (not included in filePro Lite)

There are " class members " within the @ODBC [] array to access information about the columns.

To return the names of the columns, use the " .NAME " member:

```
@ODBC.handle.NAME[column]
```

Note that expressions and aggregate functions have default names generated by ODBC. To explicitly specify a name for the column, you need to append " AS name " to the expression, such as " SELECT MAX(sales) AS MaxSales "

To return the types of the columns (as a server-specific name), use the " .TYPE " member:

```
@ODBC.handle.TYPE[column]
```

To return the types of the columns (as a server-independent number), use the " .TYPENUM " member:

```
@ODBC.handle.TYPENUM[column]
```

The following members can be returned when using the TYPENUM subscript.

```
SQL_GUID -11
SQL_WLONGVARCHAR -10
SQL_WVARCHAR -9
SQL_WCHAR -8
SQL_BIT -7
SQL_TINYINT -6
SQL_BIGINT -5
SQL_LONGVARIABLE -4
SQL_VARIABLE -3
SQL_BINARY -2
SQL_LONGVARCHAR -1
SQL_UNKNOWN_TYPE 0
SQL_CHAR 1
SQL_NUMERIC 2
SQL_DECIMAL 3
SQL_INTEGER 4
SQL_SMALLINT 5
SQL_FLOAT 6
SQL_REAL 7
SQL_DOUBLE 8
SQL_DATE 9
SQL_TIME 10
SQL_TIMESTAMP 11
```

SQL_VARIABLE 12 Note that these members can also use the field name as the subscript. (Yes, even the " .NAME " member can take a name as its subscript, as redundant as that may seem)

This page is under reconstruction - please check back later.

Adobe Flash Player is no longer supported.

This page is under reconstruction - please check back later.

Adobe Flash Player is no longer supported.

Restrictions

Unique ID - The field selected to the unique ID must be an integer type. The value in the field will be used as @RN within filePro. This is not yet enforced.

Use of filePro Automatic Indexes - Automatic indexes cannot be properly maintained when updating the ODBC files using other applications such as MS Access, SQL 2000 Server, etc. since the filePro indexes are not updated by these applications. Suggest using Demand indexes if they can be applied or use the Low-level implementation method.

Wide_Character Fields - There appears to be a bug in Microsoft's ODBC library related to fixed-length wide-character ODBC fields, where the last character is not returned. For example, in SQL Server if you define a field of type "nchar" and length 5, and the field contains "ABCDE", only "ABCD" will be returned by the ODBC library. Note that "nvarchar" (variable-length wide-character) fields do not appear to have this problem.

Currency Fields - filePro currently cannot save records from a Microsoft Access database that has a "currency" type field.

Column Limits - There is an undocumented(?) limit of 255 columns for Microsoft's MFC "CRecordset" class for accessing ODBC data sources. This limit is manifested as an "Assertion failure" error when using "Define Files", "Inquire/Update/Add", etc. when the table is more than 255 columns.

SQL Data Type & filePro Edits

The ODBC database data type is translated to a filePro edit type when using the High-Level method. The following chart identifies the filePro edit type that is applied for the corresponding ODBC data types.

SQL data type	Default filePro data type	Notes
GUID (-11)	*	
WLONGVARCHAR (-10)	*	
WVARCHAR (-9)	*	
WCHAR (-8)	*	
BIT (-7)	.0	
TINYINT (-6)	.0	
BIGINT (-5)	*	
LONGVARBINARY (-4)	*	
VARBINARY (-3)	*	
BINARY (-2)	*	
LONGVARCHAR (-1)	*	
UNKNOWN_TYPE (0)	*	
CHAR (1)	*	
NUMERIC (2)	.n	" n " is determined by the " scale " setting of the field.
DECIMAL (3)	.n	" n " is determined by the " scale " setting of the field.
INTEGER (4)	.0	
SMALLINT (5)	.0	
FLOAT (6)	F	
REAL (7)	F	
DOUBLE (8)	F	
DATE (9)	MDYY/	
TIME (10)	HMS	
TIMESTAMP (11)	*	You can change the filePro type to a date field, and filePro will use just the date portion of the field. Or, you can use a time field, and filePro will use just the time portion.
VARCHAR (12)	*	

Notes

The default length of the field (except for MDYY/ and HMS) is determined by the " precision " setting of the SQL field, except for NUMERIC and DECIMAL , which use " precision + 2 " for the length.

The above SQL data types are the internal representation format used by the SQL data engine. ODBC data sources may have different names for the field types, but they are all represented internally by one of the above types. For example, SQL Server calls its auto-increment field type " int identity " , and stores it as INTEGER . Microsoft Access also uses INTEGER , but calls it " COUNTER " .

Microsoft Access " currency " data type is stored as a NUMERIC type, precision 19, scale 4. However, there appears to be some problem with currency fields that prevents filePro from saving a record that has a currency-type field in it.

Technical Notes

READ-ONLY FIELDS & ODBC (not included in filePro Lite)

filePro recognizes fields marked as " read-only " from the ODBC data source, and will not allow you to modify them. If placed on the screen, filePro will automatically treat it as protected. You cannot assign to such fields in processing.

Microsoft SQL Server does not return a read-only status on read-only fields. Rather, every field is listed as " writable_unknown " , meaning it is not known whether the field is writable or not. Unfortunately, since every field is listed this way, filePro must treat " writable_unknown " as " writable " . Modifying read-only fields marked this way is not stopped by filePro, and will lead to SQL errors.

NULL SQL Values

ODBC returns NULL SQL values by using specific " pseudo-NULL " C values. Unfortunately, there is no way we currently know of to distinguish between a pseudo-NULL value and a field containing that same non-NULL value. For example, long-integer NULL values are returned as 1,246,576,928. If a long-integer field happens to contain 1,246,576,928, there is no way to distinguish that from the pseudo-NULL value. For now, filePro will not do anything special with such values, and NULL values will appear with their pseudo-NULL values.

PFLONGVARDOT=OLD

filePro used to accept a period in variable names. This is now not allowed (it never should have been allowed in the first place), in order to permit enhanced functionalities. To revert to the old behavior and allow periods in variable names, you can set PFLONGVARDOT=OLD. Note, however, that this will disable certain features, such as access to ODBC and biometrics, which require that periods not be allowed here.

Environment Variables

PFODBCCOMMITTYPE=*n*

Selects the open-commit-type to use for high-level ODBC data sources, where:

0 = "SELECT * FROM tablename" (default)

Very slow on some data sources with very large files, but uses nothing non-standard.

1 = "SELECT * FROM tablename WHERE id_field = nnn"

(Where "id_field" is the name of the ID field, and "nnn" is a valid ID.)

Usually faster, but may be slower on some systems, as filePro must first determine a valid ID to use.

2 = "SELECT TOP 1 FROM tablename"

Fastest version, but "TOP 1" is non-standard and not supported everywhere. Will cause ODBC failure on those DSNs that don't support it.

Version Ref: 1.0.14

PF LICFILE=*path*

Override the default license path for fileProODBC.

Sample:

Set PF LICFILE= %pfprog%\fp\license\licfp.dat (right of the equal sign can be any path you wish but must contain the licfp.dat file)

Default path is %pfprog%\fp\lib\licfp.dat

Version Ref: ODBC 1.0.01

Handling errors

Currently, there really isn't much error handling ability included.

If a query fails, for example if no records are selected due to a WHERE clause, then @ODBC.handle["0"] will return zero, indicating no data is available. (Note that this is also true should you scroll to BCF or EOF, or position to a deleted record.)

The system array @ODBCERROR[] will contain the text of the most recent ODBC failure. Subscript 1 will contain the human-readable text of the error, and subscript 2 will contain the state information from which you can programmatically extract the error information. For more information, see

http://msdn.microsoft.com/library/en-us/vcmfc98/html/_mfc_cdbexception.3a3a.m_strstaternativeorigin.asp

Appendix A - SQL Statements (not included in filePro Lite)

ALTER TABLE Statement

Modifies the design of a table after it has been created with the CREATE statement.

Syntax

```
ALTER TABLE table ADD COLUMN field type [( size )]  
ALTER COLUMN field type [( size )];  
ALTER TABLE DROP COLUMN field
```

The ALTER TABLE statement has these parts:

Part	Description
<i>Table</i>	The name of the table to be altered.
<i>Field</i>	The name of the field to be added to or deleted from <i>table</i> . Or, the name of the field to be altered in <i>table</i> .
<i>Type</i>	The data type of <i>field</i> .
<i>Size</i>	The field size in characters (Text and Binary fields only).

Remarks

Using the ALTER TABLE statement you can alter an existing table in several ways. You can:

- Use ADD COLUMN to add a new field to the table. You specify the field name, data type, and (for Text and Binary fields) an optional size. For example, the following statement adds a 20-character Text field called Remarks to the mytest1 table:
ALTER TABLE mytest1 ADD COLUMN Remarks TEXT(20)
- Use ALTER COLUMN to change the data type of an existing field. You specify the field name, the new data type, and an optional size for Text and Binary fields. For example, the following statement changes the data type of a field in the mytest1 table called " Customer " (originally defined as Integer) to a 10-character Text field:
ALTER TABLE mytest1 ALTER COLUMN Customer TEXT(10)
- Use DROP COLUMN to delete a field. You specify only the name of the field. For example, the following statement drops the column named " Remarks " from the table called " mytest1 " . Note that you can only delete one field at a time.
- ALTER TABLE mytest1 DROP COLUMN Remarks

CREATE TABLE Statement (not included in filePro Lite)

Description: Creates a new table.

Syntax

```
CREATE TABLE table ( field1 type [( size )] [, field2 type [( size )] [, ...] )
```

Parts - The CREATE TABLE statement has these parts:

Part	Description
<i>Table</i>	The name of the table to be created.
<i>field1</i> , <i>field2</i>	The name of field or fields to be created in the new table. You must create at least one field.
<i>Type</i>	The data type of <i>field</i> in the new table.
<i>Size</i>	The field size in characters (Text and Binary fields only).

Remarks

Use the CREATE TABLE statement to define a new table and its fields and field constraints. If NOT NULL is specified for a field, then new records are required to have valid data in that field. You can also use the CREATE INDEX statement to create a primary key or additional indexes on existing tables.

DELETE Statement (not included in filePro Lite)

Creates a DELETE query that removes records from one or more of the tables listed in the FROM clause that satisfy the WHERE clause.

Syntax

```
DELETE [ table .* ]  
FROM table  
WHERE criteria
```

Parts - The DELETE statement has these parts:

Part	Description
<i>table</i>	The optional name of the table from which records are deleted.
<i>table</i>	The name of the table from which records are deleted.
<i>criteria</i>	An expression that determines which records to delete.

Remarks

DELETE is especially useful when you want to delete many records.

To delete an entire table from the database, you can use the DROP statement. Keep in mind if you use DROP that the entire table structure is lost. In contrast, when you use DELETE, only the data is deleted. The table structure and all of the table properties, such as field attributes and indexes, remain intact.

A delete query deletes entire records, not just data in specific fields. If you want to delete values in a specific field, see Update Query to change the values to NULL.

Notes

After you remove records using a delete query, you cannot undo the operation. If you want to know which records were deleted, first examine the results of a SELECT query that uses the same criteria, and then run the delete query.

Maintain backup copies of your data at all times. If you delete the wrong records, you can retrieve them from your backup copies.

DROP Statement (not included in filePro Lite)

Deletes an existing table, procedure, or view from a database, or deletes an existing index from a table.

Syntax

```
DROP {TABLE table | INDEX index ON table | PROCEDURE procedure | VIEW view}
```

Parts - The DROP statement has these parts:

Part	Description
<i>table</i>	The name of the table to be deleted or the table from which an index is to be deleted.
<i>procedure</i>	The name of the procedure to be deleted.
<i>view</i>	The name of the view to be deleted.
<i>index</i>	The name of the index to be deleted from <i>table</i> .

Remarks

You must close the table before you can delete it or remove an index from it.

You can also use ALTER TABLE to delete an index from a table.

You can use CREATE TABLE to create a table and CREATE INDEX or ALTER TABLE to create an index. To modify a table, use ALTER TABLE.

INSERT INTO Statement (not included in filePro Lite)

Adds a record or multiple records to a table. This is referred to as an append query.

Syntax

Multiple-record append query:

```
INSERT INTO target [( field1 [, field2 [, ...]])] [(IN externaldatabase )  
  SELECT [ source .] field1 [, field2 [, ...]]  
  FROM tableexpression
```

Single-record append query:

```
INSERT INTO target [( field1 [, field2 [, ...]])]  
  VALUES ( value1 [, value2 [, ...]])
```

Parts - The INSERT INTO statement has these parts:

Part	Description
<i>target</i>	The name of the table or query to append records to.
<i>field1</i> , <i>field2</i>	Names of the fields to append data to, if following a <i>target</i> argument, or the names of fields to obtain data from, if following a <i>source</i> argument.
<i>externaldatabase</i>	The path to an external database. For a description of the path, see the IN clause.
<i>source</i>	The name of the table or query to copy records from.
<i>tableexpression</i>	The name of the table or tables from which records are inserted. This argument can be a single table name or a compound resulting from an INNER JOIN, LEFT JOIN, or RIGHT JOIN operation or a saved query.
<i>value1</i> , <i>value2</i>	The values to insert into the specific fields of the new record. Each value is inserted into the field that corresponds to the value's position in the list: <i>value1</i> is inserted into <i>field1</i> of the new record, <i>value2</i> into <i>field2</i> , and so on. You must separate values with a comma, and enclose text fields in quotation marks (' ').

Remarks

You can use the INSERT INTO statement to add a single record to a table using the single-record append query syntax as shown above. In this case, your code specifies the name and value for each field of the record. You must specify each of the fields of the record that a value is to be assigned to and a value for that field. When you do not specify each field, the default value or NULL is inserted for missing columns. Records are added to the end of the table.

You can also use INSERT INTO to append a set of records from another table or query by using the SELECT ... FROM clause as shown above in the multiple-record append query syntax. In this case, the SELECT clause specifies the fields to append to the specified *target* table.

INSERT INTO is optional but when included, precedes the SELECT statement.

If your destination table contains a PRIMARY key, make sure you append unique, non-Null values to the primary key field or fields; if you do not, the database engine will not append the records.

If you append records to a table with an AutoNumber field and you want to renumber the appended records, do not include the AutoNumber field in your query. Do include the AutoNumber field in the query if you want to retain the original values from the field.

Use the IN clause to append records to a table in another database.

To create a new table, use the SELECT INTO statement instead to create a table.

To find out which records will be appended before you run the append query, first execute and view the results of a SELECT query that uses the same selection criteria.

An append query copies records from one or more tables to another. The tables that contain the records you append are not affected by the append query.

Instead of appending existing records from another table, you can specify the value for each field in a single new record using the VALUES clause. If you omit the field list, the VALUES clause must include a value for every field in the table; otherwise, the INSERT operation will fail. Use an additional INSERT INTO statement with a VALUES clause for each additional record you want to create.

SELECT Statement (not included in filePro Lite)

Return information from the database as a set of records.

Syntax

```
SELECT [ predicate ] { * | table . * | [ table .] field1 [AS alias1] [, [ table .] field2 [AS alias2] [, ...]]]  
  FROM tableexpression [, ...] [(IN externaldatabase )  
  [WHERE... ]  
  [GROUP BY... ]  
  [HAVING... ]  
  [ORDER BY... ]
```

Parts - The SELECT statement has these parts:

Part	Description
<i>Predicate</i>	One of the following predicates: ALL, DISTINCT, DISTINCTROW. You use the predicate to restrict the number of records returned. If none is specified, the default is ALL.

*	Specifies that all fields from the specified table or tables are selected.
<i>Table</i>	The name of the table containing the fields from which records are selected.
<i>field1</i> , <i>field2</i>	The names of the fields containing the data you want to retrieve. If you include more than one field, they are retrieved in the order listed.
<i>alias1</i> , <i>alias2</i>	The names to use as column headers instead of the original column names in <i>table</i> .
<i>Tableexpression</i>	The name of the table or tables containing the data you want to retrieve.
<i>Externaldatabase</i>	The name of the database containing the tables in <i>tableexpression</i> if they are not in the current database.

Remarks

To perform this operation, the Microsoft® Jet database engine searches the specified table or tables, extracts the chosen columns, selects rows that meet the criterion, and sorts or groups the resulting rows into the order specified.

SELECT statements do not change data in the database.

SELECT is usually the first word in an SQL Statement.

The minimum for a SELECT statement is:

```
SELECT fields FROM table
```

You can use an asterisk (*) to select all fields in a table. The following example selects all of the fields in the mymedia table:

```
SELECT * FROM mymedia
```

If a field name is included in more than one table in the FROM clause, precede it with the table name and the . (dot) operator. In the following example, the Title field is in both the mymedia table and the mytest1 table. The SQL statement selects Title and Rated from the mymedia table and remarks from the mytest1 table:

```
SELECT mymedia.title, mymedia.rated, mytest1.remarks
```

```
FROM mymedia , mytest1
```

```
WHERE mymedia.title = mytest1.title
```

When a record set object is created, the table's field name is used as the Field object name in the **Recordset** object. If you want a different field name or a name is not implied by the expression used to generate the field, use the AS reserved word. The following example uses the title Notes to name the returned **Field** object in the resulting **Recordset** object:

```
SELECT mytest1.Remarks
```

```
AS Notes FROM mytest1
```

SELECT...INTO Statement (not included in filePro Lite)

Creates a new Table from an existing table.

Syntax

```
SELECT field1[, field2[, ...]] INTO newtable [IN externaldatabase]
FROM source
```

Parts - The SELECT...INTO statement has these parts:

Part	Description
<i>field1</i> , <i>field2</i>	The name of the fields to be copied into the new table.
<i>Newtable</i>	The name of the table to be created. If <i>newtable</i> is the same as the name of an existing table, a trappable error occurs.
<i>Externaldatabase</i>	The path to an external database. For a description of the path, see the IN clause.
<i>Source</i>	The name of the existing table from which records are selected. This can be single or multiple tables or a query.

Remarks

You can use make-table queries to archive records, make backup copies of your tables, or make copies to export to another database or to use as a basis for reports that display data for a particular time period. For example, you could produce a Monthly Sales by Region report by running the same make-table query each month.

Notes

You may want to define a primary key for the new table. When you create the table, the fields in the new table inherit the data type and field size of each field in the query's FROM source, but no other field or table properties are transferred.

To add data to an existing table, use the INSERT INTO statement instead to create an append query.

To find out which records will be selected before you run the make-table query, first examine the results of a SELECT statement that uses the same selection criteria.

Make sure that you do not use a table name that already exists.

UPDATE Statement (not included in filePro Lite)

Creates an update query that changes values in fields in a specified table based on specified criteria.

Syntax

```
UPDATE table
SET newvalue
WHERE criteria ;
```

Parts - The UPDATE statement has these parts:

Part	Description
<i>Table</i>	The name of the table containing the data you want to modify.
<i>Newvalue</i>	An expression that determines the value to be inserted into a particular field in the updated records.
<i>Criteria</i>	An expression that determines which records will be updated. Only records that satisfy the expression are updated.

Remarks

UPDATE is especially useful when you want to change many records. The following example will change the Format column values in mytest2 from " Tape " to " VHS " .

```
UPDATE mytest2 SET Format= VHS WHERE Format = Tape
```

Notes

UPDATE does not generate a result set. Also, after you update records using an update query, you cannot undo the operation. If you want to know which records were updated, first examine the results of a SELECT query that uses the same criteria, and then run the update query.

Maintain backup copies of your data at all times. If you update the wrong records, you can retrieve them from your backup copies.

XLSX MARKUP LANGUAGE

This new feature of filePro allows you to export data in an XLSX format to be opened directly by Microsoft Excel and others.

Follow this link to the full XLSX documentation. https://fptech.com/fptech/pdf/xlsx_docs.pdf

The documentation, xlsdoc.pdf, can also be found in your ~/app/fp/docs folder

XL_OPEN()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

```
e = XL_OPEN(file [, name])
```

Start building an XLSX output file.

Parameters -

file : Path to the file to create. If no full path is given the generated file will be placed in the PFTMP or equivalent directory.

name : The name for the default sheet that will be created. Defaults to Sheet1.

If the filename does not end in ".xlsx" it will be added on creation.

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

Note: Only one XLSX file can be created at a time.

XL_SAVE()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

```
e = XL_SAVE([password])
```

Save the current XLSX file.

Parameters -

password : If specified, encrypt the XLSX output file using Agile encryption (AES128).

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

Note: Encrypted XLSX files cannot be opened with most third party programs such as LibreOffice and OpenOffice. They are fully supported by Excel however. The documents are saved in an encrypted CFB file.

XL_ADDSHEET()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

```
handle = XL_ADDSHEET([name])
```

Add a new sheet to the XLSX document.

Parameters -

name : The name for the sheet to be created. Defaults to auto naming the sheet based on the Sheet1, Sheet2, ..., SheetN template.

Returns a handle to a new sheet object on success and "-1" on error.

XL_ERROR() can be called to return the last error.

XL_ADDCELL()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

```
e = XL_ADDCELL([data [, style [, sheet [, row [, col]]]])
```

Add a new cell to the XLSX document.

Parameters -

data : Data to be inserted into the document. A cell starting with '=' will be treated as a formula.

style : Handle to style to be used for this cell. Use blank to use the default style.

sheet : Handle to sheet to insert the cell on. Use blank, "0", or "-1" to use the default sheet.

row : Row to place the cell (0 indexed).

col : Column to place the cell (0 indexed).

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

Note: Using an empty or negative row/column value will cause the cell to be added using the auto counter in the sheet, incrementing the column value after the cell is added. Specifying a location will reposition the auto counter. Formulas can be used as part of the data as well by prefixing the string with '='.

XL_ADDCELL2()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

```
e = XL_ADDCELL2([data [, style [, sheet [, cell]]]])
```

Add a new cell to the XLSX document.

Parameters -

- data** : Data to be inserted into the document. A cell starting with '=' will be treated as a formula.
- style** : Handle to style to be used for this cell. Use blank to use the default style.
- sheet** : Handle to sheet to insert the cell on. Use blank, "0", or "-1" to use the default sheet.
- cell** : The Excel style cell to insert the cell. e.g. "A1" "D6" "F6".

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

Note: Using an empty cell number will cause the cell to be added using the auto counter in the sheet, incrementing the column value after the cell is added. Specifying a location will reposition the auto counter. Formulas can be used as part of the data as well by prefixing the string with '='.

XL_FORMAT()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

```
handle = XL_FORMAT(format)
```

Create a new format to use with the XLSX document.

Parameters -

format : Excel format string to use to format the a style. e.g.

```
"$ #,###,nnn.nn"
```

```
"% ##n.n"
```

```
"m/d/yyyy"
```

Returns a handle to a new format object on success and "-1" on error.

XL_ERROR() can be called to return the last error.

XL_COLWIDTH()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

e = XL_COLWIDTH(width, firstcol, lastcol [, sheet])

Change the default column width for a sheet between a range.

Parameters -

width : Width of the column(s). e.g. "24" "12.5", "11"
firstcol : Zero based column index or column letter to set from.
lastcol : Zero based column index or column letter to set to.
sheet : Handle to sheet to change the cell widths.

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

XL_FONT()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

```
handle = XL_FONT(font, [size [, attr [, color]])
```

Create a new font to use with the XLSX document.

Parameters -

font : Name of the font to use.

size : Point size of the font. e.g. "11" "8.42" "12", default "11.0"

attr : List of attributes to apply to this font, separated by commas.
e.g. "bold,italic"

Values:

"bold"

"italic"

"underline"

"strike"

"unlocked"

"hidden"

"wrap"

"shrink"

"fill"

"left"

"center"

"right"

"justify"

"top"

"bottom"

"vjustify"

"vcenter"

color : The RGB Hex value to set the font color.

e.g. "000000" "ADD8E6"

Returns a handle to a new font object on success and "-1" on error.

XL_ERROR() can be called to return the last error.

XL_BORDER()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

```
handle = XL_BORDER(borderstyle [, color])
```

Create a new border to use with the XLSX document.

Parameters -

borderstyle : The style to use with this border. Must be one of the

following values:

"thin"

"medium"

"dashed"

"dotted"

"thick"

"hair"

"medium_dashed"

"dash_dot"

"medium_dash_dot"

"dash_dot_dot"

"medium_dash_dot_dot"

"slant_dash_dot"

color : The RGB Hex value to set the border color.

e.g. "000000" "ADD8E6"

Returns a handle to a new border object on success and "-1" on error.

XL_ERROR() can be called to return the last error.

XL_FILL()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

```
handle = XL_FILL(bg [, fg [, fill]])
```

Create a new fill to use with the XLSX document.

Parameters -

bg : The RGB Hex value to set the background fill color.

e.g. "000000" "ADD8E6"

fg : The RGB Hex value to set the foreground fill color.

e.g. "000000" "ADD8E6"

fill : The fill pattern to use, defaults to "solid" fill. Value must be one of the following.

"solid"

"medium_gray"

"dark_gray"

"light_gray"

"dark_horizontal"

"dark_vertical"

"dark_down"

"dark_up"

"dark_grid"

"dark_trellis"

"light_horizontal"

"light_vertical"

"light_down"

"light_up"

"light_grid"

"light_trellis"

"gray_125"

"gray_0625"

Returns a handle to a new fill object on success and "-1" on error.

XL_ERROR() can be called to return the last error.

XL_ADD_DT()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

```
e = XL_ADD_DT(date, time [, style [, sheet [, row [, col]]]])
```

Combine two fields into a single spreadsheet datetime field and insert it as a new cell in the XLSX document.

Parameters -

date : filePro date field.

time : filePro time field.

style : Handle to style to be used for this cell. Use blank to use the default style.

sheet : Handle to sheet to insert the cell on. Use blank, "0", or "-1" to use the default sheet.

row : Row to place the cell (0 indexed).

col : Column to place the cell (0 indexed).

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

XL_ADD_DT2()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

```
e = XL_ADD_DT2(date, time [, style [, sheet [, cell]])
```

Combine two fields into a single spreadsheet datetime field and insert it as a new cell in the XLSX document.

Parameters -

date : filePro date field.

time : filePro time field.

style : Handle to style to be used for this cell. Use blank to use the default style.

sheet : Handle to sheet to insert the cell on. Use blank, "0", or "-1" to use the default sheet.

cell : The Excel style cell to insert the cell. e.g. "A1" "D6" "F6".

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

XL_CHART()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

```
handle = XL_CHART(type [, title [, xname [, yname [, row [, col [, stylenum  
    [, sheet [, xoff [, yoff [, xscale [, yscale]]]]]]]]]]))
```

Add a new chart to the XLSX document.

Parameters -

type : Type of chart to create. Must be one of the following values.

- "area"
- "area_stacked"
- "area_stacked_percent"
- "bar"
- "bar_stacked"
- "bar_stacked_percent"
- "column"
- "column_stacked"
- "column_stacked_percent"
- "doughnut"
- "line"
- "line_stacked"
- "line_stacked_percent"
- "pie"
- "scatter"
- "scatter_straight"
- "scatter_stright_markers"
- "scatter_smooth"
- "scatter_smooth_markers"
- "radar"
- "radar_with_markers"
- "radar_filled"

title : The title for this chart.

xname : The title for the x-axis.

yname : The title for the y-axis.

row : Row to place the cell (0 indexed).

col : Column to place the cell (0 indexed).

stylenum : Number of the built in Excel style to use. Must be between "1" and "48". The default style is 2. The value is one of the 48 built-in styles available on the "Design" tab in Excel 2007.

sheet : Handle to sheet to insert the chart on. Use blank, "0", or "-1" to use the default sheet.

xoff : X axis offset to place the chart, in pixels.

yoff : Y axis offset to place the chart, in pixels.

xscale : Scale the chart along the x axis. e.g. "1", "0.5" "2". Value cannot be negative.

yscale : Scale the chart along the x axis. e.g. "1", "0.5" "2". Value cannot be negative.

Returns a handle to a new chart object on success and "-1" on error.

XL_ERROR() can be called to return the last error.

Note: The chart functions do not use the auto counter found in the sheets

and instead will default to "0", "0" or "A1" when used for insertion.

XL_CHART2()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

```
handle = XL_CHART2(type [, title [, xname [, yname [, cell [, stylenum [, sheet  
    [, xoff [, yoff [, xscale [, yscale]]]]]]]]])
```

Add a new chart to the XLSX document.

Parameters -

type : Type of chart to create. Must be one of the following values.

- "area"
- "area_stacked"
- "area_stacked_percent"
- "bar"
- "bar_stacked"
- "bar_stacked_percent"
- "column"
- "column_stacked"
- "column_stacked_percent"
- "doughnut"
- "line"
- "line_stacked"
- "line_stacked_percent"
- "pie"
- "scatter"
- "scatter_straight"
- "scatter_stright_markers"
- "scatter_smooth"
- "scatter_smooth_markers"
- "radar"
- "radar_with_markers"
- "radar_filled"

title : The title for this chart.

xname : The title for the x-axis.

yname : The title for the y-axis.

cell : The Excel style cell to insert the cell. e.g. "A1" "D6" "F6".

stylenum : Number of the built in Excel style to use. Must be between "1" and "48". The default style is 2. The value is one of the 48 built-in styles available on the "Design" tab in Excel 2007.

sheet : Handle to sheet to insert the chart on. Use blank, "0", or "-1" to use the default sheet.

xoff : X axis offset to place the chart, in pixels.

yoff : Y axis offset to place the chart, in pixels.

xscale : Scale the chart along the x axis. e.g. "1", "0.5" "2". Value cannot be negative.

yscale : Scale the chart along the x axis. e.g. "1", "0.5" "2". Value cannot be negative.

Returns a handle to a new chart object on success and "-1" on error.

XL_ERROR() can be called to return the last error.

Note: The chart functions do not use the auto counter found in the sheets and instead will default to "0", "0" or "A1" when used for insertion.

XL_CHARTSHEET()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

```
handle = XL_CHARTSHEET(type [, title [, xname [, yname [, stylenum]]])
```

Add a new chartsheet to the XLSX document. A chartsheet is a full chart that occupies its own sheet and cannot contain any cells.

Parameters -

type : Type of chart to create. Must be one of the following values.

- "area"
- "area_stacked"
- "area_stacked_percent"
- "bar"
- "bar_stacked"
- "bar_stacked_percent"
- "column"
- "column_stacked"
- "column_stacked_percent"
- "doughnut"
- "line"
- "line_stacked"
- "line_stacked_percent"
- "pie"
- "scatter"
- "scatter_straight"
- "scatter_stright_markers"
- "scatter_smooth"
- "scatter_smooth_markers"
- "radar"
- "radar_with_markers"
- "radar_filled"

title : The title for this chart.

xname : The title for the x-axis.

yname : The title for the y-axis.

stylenum : Number of the built in Excel style to use. Must be between "1" and "48". The default style is 2. The value is one of the 48 built-in styles available on the "Design" tab in Excel 2007.

Returns a handle to a new chartsheet object on success and "-1" on error. XL_ERROR() can be called to return the last error.

XL_SERIES()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

```
e = XL_SERIES(chartnum, sheet, namerow, namecol, cfirstrow, cfirstcol, clastrow,  
             clastcol, vfirstrow, vfirstcol, vlastrow, vlastcol)
```

Add a series to a chart or chartsheet.

Parameters -

chartnum : Handle to a chart or chartsheet to add series.

sheet : Handle to sheet to get values from. Use blank, "0", or "-1" to use the default sheet.

namerow : Series name row (0 indexed).

namecol : Series name column (0 indexed).

cfirstrow : Categories first row (0 indexed).

cfirstcol : Categories first column (0 indexed).

clastrow : Categories last row (0 indexed).

clastcol : Categories last column (0 indexed).

vfirstrow : Values first row (0 indexed).

vfirstcol : Values first column (0 indexed).

vlastrow : Values last row (0 indexed).

vlastcol : Values last column (0 indexed).

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

XL_SERIES2()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

e = XL_SERIES2(chartnum, sheet, namecell, cfirst, clast, vfirst, vlast)

Add a series to a chart or chartsheet.

Parameters -

chartnum : Handle to a chart or chartsheet to add series.

sheet : Handle to sheet to get values from. Use blank, "0", or "-1" to use the default sheet.

namecell : Series name Excel style cell. e.g. "A1" "D6" "F6".

cfirst : Categories first Excel style cell. e.g. "A1" "D6" "F6".

clast : Categories last Excel style cell. e.g. "A1" "D6" "F6".

vfirst : Values first Excel style cell. e.g. "A1" "D6" "F6".

vlast : Values last Excel style cell. e.g. "A1" "D6" "F6".

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

XL_PROTECTSHEET()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

```
e = XL_PROTECTSHEET(sheet, password)
```

Add a password to restrict editing of a sheet.

Parameters -

sheet : Handle to sheet to protect. Use blank, "0", or "-1" to use the default sheet.

password : Password to use to protect this sheet.

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

XL_PROTECTCHARTSHEET()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

```
e = XL_PROTECTCHARTSHEET(cs, password)
```

Add a password to restrict editing of a chartsheet.

Parameters -

cs : Handle to chartsheet to protect.

password : Password to use to protect this sheet.

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

XL_ERROR()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

```
e = XL_ERROR()
```

Return the last error generated by the XLSX set of functions.

Returns the last error string generated by the XLSX engine.

XL_SETPOS()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

```
e = XL_SETPOS(row [, col [, sheet]])
```

Set the auto counter position for a sheet.

Parameters -

row : Row to move auto counter to (0 indexed).

col : Column to move auto counter to (0 indexed).

sheet : Handle of sheet to set. Use blank, "0", or "-1" to use the default sheet.

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

XL_SETPOS2()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

```
e = XL_SETPOS2(cell [, sheet])
```

Set the auto counter position for a sheet.

Parameters -

cell : Excel style cell to set the auto counter to. e.g. "A1" "D6".

sheet : Handle of sheet to set. Use blank, "0", or "-1" to use the default sheet.

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

XL_NEXTROW()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

```
e = XL_NEXTROW([sheet])
```

Move the auto counter down a row for a sheet.

Parameters -

sheet : Handle of sheet to set. Use blank, "0", or "-1" to use the default sheet.

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

XL_NEXTCOL()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

```
e = XL_NEXTCOL([sheet])
```

Move the auto counter one column to the right for a sheet.

Parameters -

sheet : Handle of sheet to set. Use blank, "0", or "-1" to use the default sheet.

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

XL_STYLE()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

```
handle = XL_STYLE([font [, fill [, fmt [, btop [, bbot [, bleft  
                [, bright]]]]]])
```

Add a new style to the XLSX document.

Parameters -

font : Handle to font object to use.
fill : Handle to fill object to use.
fmt : Handle to format object to use.
btop : Handle to border object to use for top border.
bbot : Handle to border object to use for bottom border.
bleft : Handle to border object to use for left border.
bright : Handle to border object to use for right border.

Returns a handle to a new style object on success and "-1" on error.

XL_ERROR() can be called to return the last error.

XL_IMAGE()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

```
e = XL_IMAGE(img [, row [, col [, sheet [, xoff [, yoff [, scalex [, scaley  
[, flag]]]]]]]])
```

Add a new image to the XLSX document.

Parameters -

img : Path to image file to use.

row : Row to insert the image on (0 indexed).

col : Column to insert the image on (0 indexed).

sheet : Handle of sheet to insert image. Use blank, "0", or "-1" to use the default sheet.

xoff : X-axis offset for the image, in pixels.

yoff : Y-axis offset for the image, in pixels.

scalex : Scale the image along the x-axis. e.g. "1", "0.5" "2". Value cannot be negative.

scaley : Scale the image along the y-axis. e.g. "1", "0.5" "2". Value cannot be negative.

flag : Option of how to position image.

- "0" - Default positioning.
- "1" - Move and size image with the cells.
- "2" - Move but don't size image with the cells.
- "3" - Don't move or size the image with the cells.
- "4" - Same as "1" but wait to apply hidden cells.

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

Note: The image functions only support PNG, JPEG, and BMP files.

XL_IMAGE2()

Version Ref: 6.1 (USP 6.0.02)

Syntax:

```
e = XL_IMAGE2(img [, cell [, sheet [, xoff [, yoff [, scalex [, scaley  
    [, flag]]]]]]]);
```

Add a new image to the XLSX document.

Parameters -

img : Path to image file to use.

cell : Excel style cell to insert the image. e.g. "A1" "D6" "F6".

sheet : Handle of sheet to insert image. Use blank, "0", or "-1" to use the default sheet.

xoff : X-axis offset for the image, in pixels.

yoff : Y-axis offset for the image, in pixels.

scalex : Scale the image along the x-axis. e.g. "1", "0.5" "2". Value cannot be negative.

scaley : Scale the image along the y-axis. e.g. "1", "0.5" "2". Value cannot be negative.

flag : Option of how to position image.

- "0" - Default positioning.
- "1" - Move and size image with the cells.
- "2" - Move but don't size image with the cells.
- "3" - Don't move or size the image with the cells.
- "4" - Same as "1" but wait to apply hidden cells.

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

Note: The image functions only support PNG, JPEG, and BMP files.

XL_LASTCMD()

Version Ref: Version 6.1.00 (USP 6.0.03)

Syntax:

```
e = XL_LASTCMD()
```

Get debug information about the last XLSX call.

Returns the last evaluated command parse string.

XL_MARGINS()

Version Ref: Version 6.1.00 (USP 6.0.03)

Syntax:

```
e = XL_MARGINS([left, [right, [top, [bottom, [sheet]]]])
```

Set the worksheet print margins.

Parameters -

- left : Left margin in inches, e.g. "0.5", "1", "0.75". A blank or negative value will use the default of "0.7".
- right : Right margin in inches, e.g. "0.5", "1", "0.75". A blank or negative value will use the default of "0.7".
- top : Top margin in inches, e.g. "0.5", "1", "0.75". A blank or negative value will use the default of "0.75".
- bottom : Bottom margin in inches, e.g. "0.5", "1", "0.75". A blank or negative value will use the default of "0.75".
- sheet : Handle of sheet to set the margins. Use blank, "0", or "-1" to use the default sheet.

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

XL_LANDSCAPE()

Version Ref: Version 6.1.00 (USP 6.0.03)

Syntax:

```
e = XL_LANDSCAPE([sheet])
```

Set the worksheet to print in landscape mode.

Parameters -

sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use the default sheet.

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

XL_PORTRAIT()

Version Ref: Version 6.1.00 (USP 6.0.03)

Syntax:

```
e = XL_PORTRAIT([sheet])
```

Set the worksheet to print in portrait mode.

Parameters -

sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use the default sheet.

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

XL_GRIDLINES()

Version Ref: Version 6.1.00 (USP 6.0.03)

Syntax:

```
e = XL_GRIDLINES(option [, sheet])
```

Set if the worksheet should display gridlines when printed.

Parameters -

option : Which Gridlines to print. Cannot be blank. Must be one of the following values.

"hide_all"

"show_all"

"show_screen"

"show_print"

sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use the default sheet.

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

XL_FITPAGES()

Version Ref: Version 6.1.00 (USP 6.0.03)

Syntax:

```
e = XL_FITPAGES([height, [width, [sheet]]])
```

Fit the printed area to a specific number of pages both vertically and horizontally.

Parameters -

height : Number of pages vertically. A value of "0" or blank will set the height as necessary.

width : Number of pages horizontally. A value of "0" or blank will set the height as necessary.

sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use the default sheet.

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

XL_FREEZEPANE()

Version Ref: Version 6.2 (USP 6.1.02)

Syntax:

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

Freeze part of an XLSX 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.

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.

XL_FREEZEPANE2()

Version Ref: Version 6.2 (USP 6.1.02)

Syntax:

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

Freeze part of an XLSX 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.

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.

XL_PAPERTYPE()

Version Ref: Version 6.1.00 (USP 6.0.03)

Syntax:

```
e = XL_PAPERTYPE(type [, sheet])
```

Set the paper format for the printed output of a worksheet.

Parameters -

type : The paper format to use with a printed worksheet. Must be one of the following values.

"default"

"letter"

"tabloid"

"ledger"

"legal"

"statement"

"executive"

"a3"

"a4"

"a5"

"b4"

"b5"

"folio"

"quarto"

"10x14"

"11x17"

"note"

"envelope"

"envelope_9"

"envelope_10"

"envelope_11"

"envelope_12"

"envelope_14"

"c"

"d"

"e"

"envelope_d1"

"envelope_c3"

"envelope_c4"

"envelope_c5"

"envelope_c6"

"envelope_c65"

"envelope_b4"

"envelope_b5"

"envelope_b6"

"monarch"

"fanfold"

"german_std_fanfold"

"german_legal_fanfold"

sheet : Handle of sheet to change type. Use blank, "0", or "-1" to use the default sheet.

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

XL_CENTERH()

Version Ref: Version 6.1.00 (USP 6.0.03)

Syntax:

e = XL_CENTERH([sheet])

Center the worksheet data horizontally between the margins on the printed page.

Parameters -

sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use the default sheet.

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

XL_CENTERV()

Version Ref: Version 6.1.00 (USP 6.0.03)

Syntax:

```
e = XL_CENTERV([sheet])
```

Center the worksheet data vertically between the margins on the printed page.

Parameters -

sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use the default sheet.

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

XL_PRINTACROSS()

Version Ref: Version 6.1.00 (USP 6.0.03)

Syntax:

```
e = XL_PRINTACROSS([sheet])
```

Change the default print direction to across then down.

Parameters -

sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use the default sheet.

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

XL_SETHEADER()

Version Ref: Version 6.1.00 (USP 6.0.03)

Syntax:

```
e = XL_SETHEADER(string [, margin, [limage, [cimage, [rimage, [sheet]]]])
```

Set the printed page header.

Parameters -

- string** : The header/footer definition string. See below for format options. Cannot be blank.
- margin** : The margin in inches to use for the header/footer. A blank, "0", or negative value will use the default margin of "0.3".
- limage** : Full path to an image to use in place of the left image placeholder.
- cimage** : Full path to an image to use in place of the center image placeholder.
- rimage** : Full path to an image to use in place of the right image placeholder.
- sheet** : Handle of sheet to set header/footer. Use blank, "0", or "-1" to use the default sheet.

Format Options -

Control	Category	Description
&L	Justification	Left
&C		Center
&R		Right
&P	Information	Page number
&N		Total number of pages
&D		Date
&T		Time
&F		File Name
&A		Worksheet name
&Z		Workbook path
&fontsize	Font	Font size
&"font.style"		Font name and style
&U		Single underline
&E		Double underline
&S		Strikethrough
&X		Superscript
&Y		Subscript
&[Picture]		Image placeholder
&G		Same as &[Picture]
&&		Literal ampersand &

Text in headers and footers can be justified to the left, center and right by prefixing the text with the control characters &L, &C and &R.

For example, "&LHello, World!", "&CHello, World!", "&RHello, World!"

For simple text, if the justification is not specified the text will be center aligned. However, you must prefix the text with &C if you use any other formatting.

You are limited to 3 images in a header/footer.

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

Note: The image types supported are PNG, JPEG, and BMP files. There is a hard limit of 255 characters in a header/footer string, including control characters. Strings longer than this will not be written to the document.

XL_SETFOOTER()

Version Ref: Version 6.1.00 (USP 6.0.03)

Syntax:

```
e = XL_SETFOOTER(string [, margin, [limage, [cimage, [rimage, [sheet]]]])
```

Set the printed page footer.

Parameters -

- string** : The header/footer definition string. See below for format options. Cannot be blank.
- margin** : The margin in inches to use for the header/footer. A blank, "0", or negative value will use the default margin of "0.3".
- limage** : Full path to an image to use in place of the left image placeholder.
- cimage** : Full path to an image to use in place of the center image placeholder.
- rimage** : Full path to an image to use in place of the right image placeholder.
- sheet** : Handle of sheet to set header/footer. Use blank, "0", or "-1" to use the default sheet.

Format Options -

Control	Category	Description
&L	Justification	Left
&C		Center
&R		Right
&P	Information	Page number
&N		Total number of pages
&D		Date
&T		Time
&F		File Name
&A		Worksheet name
&Z		Workbook path
&fontsize	Font	Font size
&"font.style"		Font name and style
&U		Single underline
&E		Double underline
&S		Strikethrough
&X		Superscript
&Y		Subscript
&[Picture]		Image placeholder
&G		Same as &[Picture]
&&		Literal ampersand &

Text in headers and footers can be justified to the left, center and right by prefixing the text with the control characters &L, &C and &R.

For example, "&LHello, World!", "&CHello, World!", "&RHello, World!"

For simple text, if the justification is not specified the text will be center aligned. However, you must prefix the text with &C if you use any other formatting.

You are limited to 3 images in a header/footer.

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

Note: The image types supported are PNG, JPEG, and BMP files. There is a hard limit of 255 characters in a header/footer string, including control characters. Strings longer than this will not be written to the document.

XL_SETBACKGROUND()

Version Ref: Version 6.1.00 (USP 6.0.03)

Syntax:

```
e = XL_SETBACKGROUND(image [, sheet])
```

Set the background image for a worksheet.

Parameters -

image : Full path to an image to use as the sheet background.

sheet : Handle of sheet to set background image. Use blank, "0", or "-1" to use the default sheet.

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

Note: The image types supported are PNG, JPEG, and BMP files.

XL_SPLITPANE()

Version Ref: Version 6.2 (USP 6.1.02)

Syntax:

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

Divide a worksheet into horizontal or vertical regions.

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.

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.

XL_HIDEZEROS()

Version Ref: Version 6.1.00 (USP 6.0.03)

Syntax:

```
e = XL_HIDEZEROS([sheet])
```

Hide zero values in worksheet cells.

Parameters -

sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use the default sheet.

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

XL_SHOWROWCOL()

Version Ref: Version 6.1.00 (USP 6.0.03)

Syntax:

```
e = XL_SHOWROWCOL([sheet])
```

Show row and column headers on the printed page.

Parameters -

sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use the default sheet.

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

Standard Computer Terms

Application - A solution you create with filePro to solve a business problem. Examples of applications are: accounting, payroll, inventory control, medical office system etc. Applications usually consist of more than one filePro file joined by lookups and other functions to make the files more useful.

ASCII - American Society for Computer Information Interchange.

ASCII character set - A set of characters, 1 character per unique byte, i.e., 256 characters can be represented by one byte (8 bits in all the various combinations of 1's and 0's.).

Backup - Safety copy of data and/or programs.

Batch File - A file (DOS) whose text can be read by DOS as instructions and then executed. Batch files can be called from filePro menu options or run from the command line.

Batch Processing - Processing groups of selected records such as: posting data from invoice file to a summary file, archiving and purging inactive accounts, performing mass recalculations.

Binary - two states, off and on, 1 and 0.

bit - 1 Binary digit, a bit can either be on or off, either a 1 or a 0.

byte - 8 bits in a group. There are 256 combinations of 1's and 0's in a byte.

character - 1 byte. 256 characters can be defined with 1 byte.

Database - A collection of related information organized in files, records and fields. Separate files may share common information for the purpose of connecting records between files (see filePro term "lookup").

Disk - Long term storage system for programs and data include hard disks (magnetic), optical disks (laser). The capacity in characters of these devices is expressed in gigabytes (or megabytes [older drives]). Disks are also called drives (hard drive, compact disk drive). Nobody seems to know why.

Flat ASCII files are (by convention) comprised of only the first 128 characters that one byte can provide, i.e., the combinations of 1's and 0's that the first 7 bits can make up. Of these, only the printable or human readable characters are normally found in a flat ASCII file (combinations 32 to 128). However, some of the lower combinations, such as carriage returns and line-feeds are also present.

Adding the 8th bit produces characters 129 through 255. These are sometimes called the high ASCII set (above 128). They are "off the keyboard," meaning you can not generate them from the keyboard. FilePro is capable of storing these high characters, but there is almost no reason to ever do so in a text database. The upper ASCII characters are displayed as symbols, pictures, etc. In fact, different "symbol sets" can be overlaid on the higher ASCII set to produce different language characters, and special characters. If you see one of these funny characters in your data... there is most likely a problem and your data has been corrupted somehow. Occasionally though, you will see some graphic character(s) representing carriage returns, line-feeds and combinations (new lines) when you pull a structured file into a "non-filePro" file format. This is normal, and these characters do not mean the data is corrupted.

gigabyte - 1024 megabytes.

GUI - The abbreviation for Graphical User Interface. The term came into existence because the first interactive user interfaces to computers were not graphical; they were text-and-keyboard oriented and usually consisted of commands you had to remember and computer responses that were brief.

kilobyte - 1024 bytes

megabyte - 1024 kilobytes

ODBC - Abbreviation for Open Database Connectivity. ODBC is Microsoft's strategic interface for accessing data in a heterogeneous environment of relational and non-relational database management systems.

RAM - Random Access Memory. This is the memory in which the computer stores programs and data temporarily while you use them. It retrieves these programs and data from your disk storage.

filePro Terms

Action - On processing tables, operations such as lookups, string manipulation, restarts, requests for input from the user, messages, screen switching and math formulas.

Action line - The "Then:" line on a processing table or a line on a filePro menu to direct filePro what action to take.

Alien File - A non-filePro file from which data can be read or updated.

Alias - A name assigned to a field or lookup on a processing table so that the field or lookup can be referred to by another name. This is useful for lookups for shortening the lookup name e.g. lookup trp = temporary_file where trp is the alias.

Associated Fields - A "Real Field" that has a group identifier e.g. "A)", "A1)", etc. to provide a means of associating related data.

Box Functions - Simple operations for defining outlined boxes on a screen or report format.

Browse - Allows filePro to display as many as 18 records on the screen at one time. The user can then select a record to work on.

Browse Lookup - Allows the user to specify a browse to be performed on a lookup file or other files. Fields from the records selected are presented in a window on the current data entry screen.

Choice - A menu option.

Codes - Abbreviations used to stand for certain processes. See Print codes

Condition - The "If:" line in a processing table that allows you to test for values. If the condition listed ("If") line is "TRUE", then the action described on Action ("Then:") line is performed. If the condition is not met e.g. is "FALSE", the action is not performed. If the "If:" line is left blank, the condition is "TRUE" by default.

Cross Reference - A report that lists all lookup statements, fields and line information for a processing table. Refer to "Define Processing".

Date Expansion - Typing in a single character "/" in a filePro date field will expand to today's date.

Default - A specific value established by a program or process if the value is left blank or not entered.

Default Report - A report generated by filePro when leaving the "Define File" option of the filePro Main Menu.

Default Responses - A group of responses to prompts in filePro that application uses automatically.

Dummy Fields - In-memory variables that are used for temporarily storing, processing, displaying, and/or printing data. These can be either "SHORT" or "LONG" variable names. "A"- "Z", "AA"- "ZZ" can be used for "SHORT" variable names. Up to 64 alphanumeric characters can be used "LONG" variable names.

Edits - Specifies the type of data that a field may contain to improve accuracy for data entry or to save time. FilePro includes system edits to ensure proper dates, time, etc. and allows the programmer to create user edits.

Field - A field is a set of related characters: a category or type of information. Fields are the blanks you fill in when entering data. Examples of fields are: customer number, last name, first name, address, city, state, ZIP.

File - A set of related records. Examples of files are: customer information, products and invoices.

fPClient - The client side of the filePro Server/Client software that allows you to implement a GUI version of filePro e.g. fileProGI.

fPServer - The server side of the filePro Server/Client software that provides for a GUI version of filePro.

Fuzzy Search - Enables you to search for a record in filePro and retrieve "inexact" matches to your search value.

Index - A table used for rapid access to records within a file. Indexes are also used to merge file information using lookups.

Lookup - A filePro function that provides for merging file data.

Menu - A list of options you can use. Menus for complex applications may have sub-menus to organize the application. Examples of menus include Accounting Options, Payroll options, Inventory Control options and other options to allow quick access and control parameters required to properly execute your programs.

Pop-up Screen - A screen from another file that is displayed as a window in the current file. Pop-up Screens allow maintenance to other files without exiting a current screen.

Processing - A set of filePro instructions to tell filePro how to handle data entry, data manipulation and data output. A sequence of these instructions is called a processing table.

Real Fields - Fields where the values are permanently stored to disk. These fields are established with the "Define Files" option of the filePro Plus main menu, and are retrieved using a field number which is assigned when defining the file.

Record - A set of related fields. Examples of records are: information about a single customer, information about a single item, etc. Records are grouped into a file in filePro so that information is stored for more than one customer and more than one item.

A_AVG()

Version Ref: 6.1 (USP6.1.01)

Syntax:

```
value = A_AVG(array [, array2 [, array3 [, ... [, arrayN]]]])  
Find the average 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.

A_MAX()

Version Ref: 6.1 (USP6.1.01)

Syntax:

```
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.

A_MIN()

Version Ref: 6.1 (USP6.1.01)

Syntax:

```
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.

A_TOT()

Version Ref: 6.1 (USP6.1.01)

Syntax:

```
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.

ABS()

Syntax:

```
Then: a = ABS (n)
```

Version Ref: 1.1

Description:

Returns the absolute value of a number.

Examples:

```
Then: aa="-6"
```

```
Then: ab = abs (aa)
```

The contents of variable "ab" is "6".

ACCESS()

Syntax:

Then: N=ACCESS (filename, type)

Version Ref: 4.8

Description:

ACCESS allows you to check whether a file has a particular type of access privilege.

Type is any combination of R=Read, W=Write, and X=Execute permissions.

Return Value

0 if you have all the specified access permissions to the file.

Otherwise, a negative value is returned. This value is the negative of the system error number.

Examples:

Then: pw= access(/etc/passwd,rw)

Note: When running in a setuid environment (as with filePro), ACCESS uses the real user id to determine access permission and not the effective user id. SCO UNIX includes the effective user id, but this function is not available for all *NIX systems. Refer to your *NIX documentation for details.

ACOS()

Syntax:

$xx = \text{ACOS}(n)$

Version Ref: 4.8

Description:

Trig Function to provide angle given in radians.

Examples:

Then: $\text{radian_value} = \text{ACOS}(90)$

ACOSH

Syntax:

$xx = \text{ACOSH}(n)$

Version Ref: 5.0

Description:

Hyperbolic Trig Function to provide angle given in radians.

Examples:

Then: $\text{radian_value_hyperbolic} = \text{ACOSH}(90)$

ADDMONTH()

Syntax:

Then: xx = ADDMONTH(date,months)

Then: xx = ADDMONTH(date)

If "months" is not specified, one month is added.

Return value is a date field, the same edit type and length as date.

NOTE: If the resulting date would be past the end of the month, the last day of the month is returned.

The number of months to add can be negative to go backwards.

Version Ref: 4.8

Description:

Adds a specified number of months to a date.

Examples:

Show the date two months from now.

```
Then: msgbox "Two months from today is " & addmonth(@td,"2")
```

Warn two months before the due date. Assuming "startdate" is the starting date, and "duration" is the number of days.

```
If: @td gt addmonth(startdate + duration,"-2")
```

```
Then: msgbox "Less than two months before due date."
```

ARCHIVE (ver 6.0.00)

Lookup command to create 'archive' instead of 'copy' to archive (copy) records and maintain all system variables. @CD, @CB, etc.

ASC()

Syntax:

Then: a = ASC(n)

Then: a = ASC(exp)

To return any character other than the first, use MID with ASC:

Then: asc(mid(f, "s", "1"))

"mid" is the MID command

"f" is the field number or letter(s)

"s" is the character to convert (the starting position)

"1" is the number of characters to convert (only one is allowed)

ASC can be used on all processing tables.

Version Ref: 3.x

Description:

Converts character to ASCII code.

Examples:

In a non-filePro Plus file, you wanted to perform one operation if the first character of the non-filePro Plus field 7 is a NULL (ASCII 0), and something else if it is CTRL A (ASCII 1). Write the tests as follows:

If: asc(7) eq "0"

Then: [do one thing]

If: asc(7) eq "1"

Then: [do something else]

ASIN()

Syntax:

Then: $xx = \text{ASIN}(n)$

Version Ref: 4.8

Description:

Trig Function to provide angle given in radians.

Examples:

Then: $\text{radian_value} = \text{ASIN}(90)$

ASINH()

Syntax:

Then: $xx = \text{ASINH}(n)$

Version Ref: 5.0

Description:

Hyperbolic Trig Function to provide angle given in radians.

Examples:

Then: $\text{radian_value_hyperbolic} = \text{ASINH}(90)$

ATAN()

Syntax:

Then: $\text{angle} = \text{ATAN}(n)$

Version Ref: 4.8

Description:

Trig Function to provide angle given in radians.

Then: $\text{result} = \text{ATAN}(y,x)$

Returns the arctangent of slope (y/x) in radians.

ATANH()

Syntax:

Then: $\text{angle} = \text{ATANH}(n)$

Version Ref: 5.0

Description:

Hyperbolic Trig Function to provide angle given in radians.

Then: $\text{result} = \text{ATANH}(y,x)$

Returns the hyperbolic arctangent of slope (y/x) in radians.

AVG()

Syntax:

Then: a=AVG(n)

Then: a=AVG(exp)

Finds average at subtotal and grand total levels.

Version Ref: 3.x

Description:

A system function used in output processing to find averages.

Examples:

To obtain average sales per sales representative if "Total Sales" is field 22, use the following syntax:

Then: A=AVG(22)

To see the average, put field A on a subtotal and or total line on the report format.

Restrictions:

May not be used on INPUT processing. AVG must always be to the right of the equal sign. Averages are calculated at the subtotal and total breaks. The averaging operation is done only if the processing element containing the AVG function is encountered.

BACKGROUND

Syntax:

Then: BACKGROUND ON

Then: BACKGROUND OFF

Description:

Turn on/off ability to enter background via '!G'.

Note: You cannot turn on if PFBACKGROUND if PFBACKGROUND=OFF. (BACKGROUND ON will be ignored.)

BASE()

Syntax:

```
Then: xx=BASE(expr,inbase,outbase)
Then: xx=BASE(expr,inbase)
```

where "expr" is the number specified in "inbase"

"outbase" is the output base.

If outbase is not specified, 10 is used.

Return value is the result of converting expr from inbase to outbase.

NOTES : Inbase and outbase must be between 2 and 36, inclusive. Only the integer portion of all numbers is used.

Version Ref: 4.5

Description:

Converts between different bases.

Examples:

A simple decimal-to-hexadecimal converter.

```
Then: input popup no(8,.0) "Enter a number: "
If: no ne ""
Then: msgbox no <"decimal is" < base(no, "10", "16") <"hexadecimal."
```

BEEP

Syntax:

Then: beep

Version Ref: 3.x

Description:

Beep makes a PC or terminal speaker beep once.

Examples:

Make a sound alerting the user to some condition, i.e., overshooting a credit limit.

```
If: 3 gt cl
```

```
Then: beep;show"Customer is over credit limit."
```

Make the beeper to go off at the grand-total point of a report and warn the operator that the report was almost finished.

```
@wgt If:
```

```
Then: beep
```


BLOB

Syntax:

Then: BLOB xx delete
Then: BLOB xx export
Then: BLOB xx import
if: BLOB
if: NOT BLOB

Version Ref: 5.0

Description:

Used to manage Binary Large Objects (BLOBS), which include such items as sound clips, pictures or word processor documents. The items are stored within a filePro record as a variable length field based on the size of the object. The objects are retrieved using external programs to display, edit, print, or to otherwise manipulate them.

Note:

Length and type must be defined as (16,BLOB).

Examples:

BLOB commands:

BLOB field IMPORT filename

Imports contents of "filename" into the specified BLOB field.

BLOB field EXPORT filename

Exports the specified BLOB field into "filename".

BLOB field DELETE

Deletes the specified BLOB field.

BLOB Conditions:

If: BLOB

If: NOT BLOB

Allows you to determine whether the most recent BLOB command succeeded or not.

BOM()

Syntax:

Then: xx=BOM(date_expr)

Version Ref: 4.5

Description:

Returns the beginning of a month for any date expression. If the date expression is left blank, returns the beginning of the current month. date_expr means that the field has to be of a "date" edit type. It cannot be uncast.

Examples:

Then: payroll_month_began = BOM(payroll_date)

Returns the 1st day of the month for the payroll_date variable.

Then: current_payroll_began = BOM()

Returns the 1st day of the current month.

BOQ()

Syntax:

Then: xx=BOQ(date_expr)

Version Ref: 4.5

Description:

Returns the beginning of a quarter for any date expression. If the date expression is left blank, returns the beginning of the current quarter.

Examples:

Then: payroll_qtr_began = BOQ(payroll_date)

Returns the 1st day of the quarter for the payroll_date variable.

Then: current_payroll_qtr = BOQ()

Returns the 1st day of the current quarter.

BOY()

Syntax:

Then: xx=BOY(date_expr)

Version Ref: 4.5

Description:

Returns the beginning of a year for any date expression. If the date expression is left blank, returns the beginning of the current year.

Examples:

Then: payroll_year_began = BOY(payroll_date)

Returns the 1st day of the year for the payroll_date variable.

Then: current_year_began = BOY()

Returns the 1st day of the current year.

BREAK

Syntax:

Then: BREAK ON

Then: BREAK OFF

Version Ref: 3.x

Description:

BREAK turns the <BREAK> key OFF and ON during processing. Use BREAK OFF when you don't want the user to break out of the current operation.

IMPORTANT: Don't add the BREAK statements to your tables until the tables have been debugged. Otherwise, on DOS systems, you'll have to reboot your computer to stop your processing. On UNIX/XENIX systems, you'll have to use the emergency BREAK key, either <CTRL><^> or <CTRL><7>, to get out of the operation. Exiting this way may turn off the ECHO function, the cursor, <CTRL><D> and <BACKSPACE>.

Examples:

If your program posts financial data from a sales file to a client file, you don't want the user to be able to cancel the operation before all the data from a particular record is correctly posted:

Then: lookup client k=1 i=a -nx

If: not client

Then: goto subr

Then: break off; sales (10)=2; sales (11)=3

Then: write; break on

Note: BREAK cannot be used on automatic processing tables.

Release 5.0.14: If BREAK OFF is executed in a CALL/CHAIN process, it did not remain off upon returning prior to release 5.0.14. It now remains off upon returning. Also, it is restored to ON if you go to a new record.

BUSYBOX

```
BUSYBOX  
BUSYBOX "my message"  
BUSYBOX("10","10" )  
BUSYBOX("10","10") "my message"
```

Example: `BUSYBOX("10","10") "Calling your mother."`

This is for times when filePro is processing under the hood and is a way to let the user know filePro is doing something and not ready for use. Think of it when scanning millions of records with a browse lookup with drop processing. A user might think the process is frozen or crashed if nothing is on the screen showing that it is working. The important thing about this new command would be that the instant filepro is waiting again for any keystroke, the show working display must come down automatically as there is no way for the programmer to know when to take it down in a browse lookup situation.

DO NOT use this command within any loop as it will execute over and over. Put it in your code once prior to when your process is about to busy for an extended function.

CALL

Syntax:

Then: CALL "name"

Then: CALL "path-to-prc-table"

Then: CALL NOAUTO

Name = processing table in the current filePro file

path = full path to processing table in any filePro file

Version Ref: 4.0 (not included in filePro Lite)

4.8 Enhancement - Added CALL path

5.0 Enhancement - Added CALL NOAUTO

Version Ref: 6.0.02 (not included in filePro Lite)

Will now accept flags -RW, etc. same as CLERK to pass data/parameters to the CALLED table.

Description:

CALL calls another processing table as a subroutine. When the called processing is done, control returns to the original processing table to the next statement after the CALL statement.

CALL "path" allows you to call a processing table in another filePro file or even in a library of processing tables by identifying the specific path e.g. "c:\filepro\fpcust\test.prc".

CALL NOAUTO ignores the dummy fields defined in automatic processing. This allows you to re-use field labels as defined in the CALLED processing table.

CALL is only one level deep; you cannot call another processing table from a called table (or CHAIN from a called table). The called processing table uses the token area reserved for printing a form from Inquire, Update, Add.

Dummy fields

Called processing tables have their own set of dummy fields. It cannot use dummy fields defined in the original (calling) processing unless the calling processing is automatic processing. (Dummy fields defined in automatic processing are available to all other types of processing.) When control returns to the calling processing, the original dummy fields and values are restored.

This function can be useful in modularizing code. Single tables can be "called" into multiple applications. In other words, it is easier to develop a "library" of tables that perform various functions, and CALL these functions, as needed, from into your other applications.

Note: The CHAIN command cannot be used in called processing tables.

Examples:

The key to using CALL effectively, is understanding how CALL tables work with dummy variables. The easiest way to learn this functionality is by setting some variables and tracking their values from the INPUT table to the CALL table and back again.

The only rule of thumb is this: All variables are cleared upon entry to a CALL table EXCEPT global variables, which have been defined, on the current AUTOMATIC table. Variables defined in this manner will pass their values between the CALL and INPUT table, all others will only keep their values in normal filePro fashion.

The following code will demonstrate this clearly.

The AUTOMATIC table can be critical to running sophisticated CALL tables.

```
Then: cc(3,,g)
```

The global variables defined on the AUTOMATIC table will have their values passed to any CALLED table. If these variables are assigned values on a CALL table, the values will be available to the INPUT table. No other variables, (even global variables defined on the INPUT table) will have their values passed between INPUT and CALL tables.

The FIRST time this code is run by pressing the T key, the variables aa, bb and cc will all be empty. After that, successive presses of the T key will show that the CALL table has passed a new value back to the INPUT table. These values can be tested in various ways, one place for this test might be @entsel as shown in the code on lines 7 to 10.

```
Then: end
```

```
@keyT If:
```

```
Then: show "@aa=" { aa { ", bb=" { bb { ", cc=" { cc
```

```
Then: aa(3)="111" ; bb(3,,g)="222" ; cc="333"
```

```
Then: show "@On the input table aa=" { aa { ", bb=" { bb { ", cc=" { cc
```

```
Then: call "fred"
```

```
    If: 'this end is superfluous, it will never be executed
```

```
Then: end
```

```
@entsel If: aa eq "123"
```

```
Then: show "@The call table will not pass this variable back to INPUT."
```

```
If: bb eq "123"
```

```
Then: show "@The call table will not pass this variable back to INPUT."
```

```
If: cc eq "123"
```

```
Then: show "@GLOBAL variables defined on the AUTO table will be passed!"
```

```
Then: end
```

The CALL table itself in this example does nothing but assign values to three different variables, a regular variable, a global variable, and a global variable defined on the AUTOMATIC table.

```
Then: show "@Begin call table, aa=" { aa { ", bb=" { bb { ", cc=" { cc
```

```
Then: show "@ Now set aa, bb and cc equal to 123"
```

Then: aa="123" ; bb="123" ; cc="123"
Then: end

CEIL()
FLOOR()
INT()
ROUND() (Added Version 6.0.00)

Syntax:

Then: `xx=CEIL(num_expr)`

"num_expr" is the given number.

"xx" is the resulting value.

Return value is a field of the same edit type and length as num_expr.

NOTE: If the given number is already an integer, the original number is returned.

Version Ref: 4.5

Description:

Performs the ceiling function, which given any number, returns the next greater integer.

Examples:

Show the difference between FLOOR(), CEIL(), INT(), and converting to an integer:

Try the example with the numbers: -5.6 -5.5 -5.4 5.4 5.5 5.6 and note the differences among the functions.

Then: `input popup xx(10,.5) "Enter a number: "`

`if: xx = ""`

`Then: end`

`Then: yy(10,.0) = xx`

`Then: msgbox xx & "\nFLOOR:" < FLOOR(xx) & "\nCEIL:" < CEIL(xx)& "\nINT:" < INT(xx) & "\nrounding:" < yy`

ROUND(value [, place])

Valid 'place' values are: ... -3, -3, -1, 0, 1, 2, 3 ...

examples:

`aa=round("123.456") 'aa eq 123`

`aa=round("123.456","1") 'aa eq 123.5`

`aa=round("123.456",-1") 'aa eq 120`

CHAIN

Syntax:

Then: chain "name"

Then: chain "path-to-prc-table"

name=processing table in the current filePro file

path = full path to processing table in any filePro file

Version Ref: 3.x (modified in 4.8 to add **CHAIN path**) (not included in filePro Lite)

Description:

CHAIN lets you load a new processing table from within a processing table. CHAIN is designed to let you develop applications for, or transfer applications to, systems with very limited memory. Use CHAIN only if your processing table is too large to fit into memory all at once. You can list a CHAIN statement on an automatic processing table for conditional access of a particular table. The chosen table will be executed when you go into update mode. Keep in mind that CHAIN does not return processing to the original table when the new table is finished. If you want to return to the original table, you must CHAIN back.

CHAIN can be used on input processing tables and on automatic processing tables for Inquire, Update, Add operations only. Loading new processing tables takes time. Avoid chaining whenever possible to keep from slowing down the user. When chaining to other tables, remember that any files you've been using for lookups remain open, unless you close them with CLOSE.

The CHAIN command cannot be used from a called processing table.

CHDIR

Syntax:

Then: CHDIR directory

where "directory" is the name of the directory.

Note: Under MS-DOS, specifying a drive will also make that drive current.

Under MS-DOS, you will be left in the specified directory upon exiting filePro. Under Unix systems, you will be left in your original directory upon exit from filePro.

Version Ref: 4.5

Description:

Changes the current directory.

Examples:

Run a program from a required directory, and then return to the previous directory.

Then: xx = GETCWD()

Then: CHDIR "/appl/otherapp" ; system "run-app"

Then: CHDIR xx

Without the CHDIR command, the following processing would work under MS-

DOS, but not Unix:

Then: SYSTEM "cd \appl\otherapp" ; system "run-app"

And the following would work under Unix, but not MS-DOS:

Then: SYSTEM "cd /appl/otherapp ; run-app"

CHR()

Syntax:

Then: a=CHR(n)

Then: a=CHR(exp)

where "n" is the decimal ASCII code (a literal) you want converted to its character equivalent, and "exp" is an expression.

Version Ref: 3.1

Description:

Converts ASCII code to character.

Examples:

To export ASCII code 7, bell, as part of a field:

Then: aa=chr("7"){5}

CLEAR

Syntax:

Then: CLEAR array

Where "array" is the array name.

Version Ref: 4.1

Description:

Sets each element of an array to blank; this includes alias arrays.

CLEARB

Syntax:

Then: CLEARB

Version Ref: 4.1

Description:

CLEARB removes a browse lookup window remaining on the screen. Since you can create a browse lookup window that remains on the screen indefinitely, using the "show=keep" or "show=only" parameters in the browse lookup statement, the CLEARB command gives you a way to remove it.

CLEARP

Syntax:

Then: CLEARP

Version Ref: 4.1

Description:

This command removes a popup and cleans up the screen. It is a good idea to always use this command after a popup.

Examples:

Then: end

@keyV If:

Then: lookup cust = customer k=2 i=a -nx

If: not cust

Then: errorbox "Customer Not On File!" ; end

Then: popup cust,"bal" ; show"@@" ; clearp ; end

The above code puts screen "bal" from the "customer" file on the screen and keeps it there until the user presses ENTER (the show"@@" does this). Once ENTER is pressed, the CLEARP command clears the popup from the screen.

NOTES: The ERRORBOX in the above code clears itself from the screen when the user presses ENTER. This is the nature of how ERRORBOX works, but not so with POPUP.

The quotes are necessary around the screen name "bal", but there are no quotes needed around the filename (or filename alias) in the POPUP command. If a single character screen name is used, no quotes are required. You can substitute a screen name that is contained in a variable by enclosing the variable in parentheses.

If:

Then: aa="test"

If:

Then: popup -, (aa) ; show"@@" ; clearp ; end

The above code will display screen "test" in the current file.

CLOSE

Syntax:

Then: close
Then: close filename

Use CLOSE by itself to close all files except the current file.

Use CLOSE filename to close an individual file, where "filename" is a lookup file name, or an export file name. CLOSE can be used on all processing tables.

Version Ref: 3.x (not included in filePro Lite)

Description:

Use CLOSE when doing lookups to avoid exceeding your operating system's open-file limitations.

NOTES: Different operating systems have varying abilities with regard to how many files can be opened at one time by one user. If you reach this limit (older XENIX systems, MS-DOS are usually 20 files open per user, with 3 files being reserved for use by the o/s itself; Unix systems usually are set to 60 to 100+ files open simultaneously by one user), then you may have to CLOSE some files that filePro does not need at a particular moment to accomplish the task at hand. When done, other files can be closed and previous ones opened again. In general, however, filePro will automatically handle the closing and opening of files for you.

Every individual file is counted toward the operating system limits. That is, in filePro's case, the "map", the "indexes", the "key" and "data" files (if there is anything in the data file which is usually not used any more), among others. These can mount up very quickly if you are doing a lot of lookups to files with lots of indexes. Still, the maximum number of open files is a hard limit to reach these days. You will more than likely not need these CLOSE functions.

Version 5.8.03.24 and higher was raised to 512 open files.

Technical Notes:

Open files will include a minimum of three files used by the operating system for screen and keyboard operation.

Restrictions:

Remember that you cannot access fields from a file once it's closed.

CLOSE()

Syntax:

Then: aa=CLOSE(handle)

"handle" is the file handle returned by OPEN() or CREATE().

Return value - 0 if successful; Negative if failed.

Note: Once a file is closed, it can no longer be accessed through that handle. If you need to access the file again, you must call OPEN() to get a new handle.

Version Ref: 3.x (not included in filePro Lite)

Description:

Closes an opened file.

CLOSEDIR()

Syntax:

Then: N=CLOSEDIR()

Version Ref: 4.8 (not included in filePro Lite)

Description:

Closes the OPENDIR(), NEXTDIR() session.

CLS

Syntax:

Then: CLS
Then: CLS(s)
Then: CLS(s,n)
CLS Clear entire screen.
CLS(s) Clear screen from line "s".
CLS(s,n) Clear screen from line "s" for "n" lines.

Without parameters, the command clears the entire 24-line Inquire, Update, Add screen, and lines 3-24 of the Request Output screen. The syntax is:

Then: cls

To clear from a particular line to the bottom of the screen, use this syntax:

Then: cls(s)

where "s" is an expression that designates the number of the line from which to start clearing. The value "s" is really an expression. In other words, the program can use the value of a literal (a real number from 1 to 24), a filePro field (real or dummy that contain a number from 1 to 24), or any expression that resolves to a number from 1 to 24 for a starting point. (Use quotation marks for Literals.)

To clear only particular lines, give the starting line and the number of lines to clear:

Then: cls(s,n)

where "s" is the starting line and "n" is the number of lines to clear from the starting line. The "n" value is also an expression whose value should be limited by how many lines are available to clear from the line designated by the "s" value.

Version Ref: 3.x

Description:

CLS is used to clear the screen during Inquire, Update, Add and Request Output operations

Examples:

To clear lines 5 through 14:

Then: cls("5","10")

Suppose field "aa" has a value of 5. You could clear the screen from line 5 to line 14 as follows:

Then: cls(aa,"10")

NOTE: As with all filePro expressions, quotation marks are required for Literals. The parameters s and n are expressions. If you use literal values, they must have quotes (as always).

COMPARE()

Syntax:

Then: xx=COMPARE(expr1,expr2)

"expr1" and "expr2" are the two values to compare.

Return values:

-1 if expr1 is less than expr2.

0 if expr1 equals expr2.

1 if expr1 is greater than expr2.

Version Ref: 4.5

Description:

Compares two values, with case sensitivity.

Examples:

Ask for a password and require the case to be correct.

If:

Then: inputpw popup pw "Enter password: "

If: compare (pw,password) ne "0"

Then: errorbox "Incorrect password" ; end

COPY/COPYIN

Syntax:

Then: COPY filename
Then: COPY lookupname TO lookupname
Then: COPYIN filename
Then: COPY lookup1 TO lookup2

v6.2 (6.1.02 USP)

Then: COPY array
Then: COPYIN array
Then: COPY array TO lookup
Then: COPY lookup TO array
Then: COPY array1 TO array2

Once you have used a lookup statement to open a record in another file (or your current file), any of the first three of these commands can be used. The COPY command copies the entire current record to the looked-up record. (The maps of each file must match each other exactly with respect to field lengths and edit types. The destination map can be bigger than the source map as long as the fields match between files from the beginning of each file to the point where the destination map adds more fields.) The fields are copied one for one so that the destination record becomes a duplicate of the current record (COPY) or the current record becomes a duplicate of the source record (COPYIN). If you have used lookup to open two files simultaneously, you can use the fourth syntax shown above to COPY a record from one lookup to the other or vice versa.

Note: It is prudent to do a WRITE immediately after COPY or COPYIN to ensure that the record has been completely copied and the transaction has been handed to the O/S. COPY has great usefulness when creating archive files since it will copy an entire record with only one command. You don't have to do it field by field.

NEW in Version 6.0 and higher: [ARCHIVE](#)

Automatic indexes are automatically updated, if need be, when COPY or COPYIN is used.

Version Ref: 3.x (modified in version 4.5 to add copy one lookup to another)

Description:

COPY works in conjunction with the LOOKUP command, to copy all fields in the current record to a record in a looked-up file, or COPY all fields from one lookup to another lookup. COPYIN works in conjunction with the LOOKUP command, to copy all fields in a looked-up record to the current record.

Restrictions:

COPY and COPYIN are not available on Automatic processing tables.

Examples:

Then: lookup arch = oldinvoices r=free -e
Then: copy arch ; write arch ; delete ; end

The processing above gets a free record in the archive file, copies the current record to the free record in the other file, writes the other file (just for good measure, really not needed), then deletes the current record from the current file. (The DELETE command is really only executed after the END statement of a processing table is executed, but it certainly does happen. This record will now only be found in the "oldinvoices" file.)

COPY()

v6.2 (6.1.02 USP) `n = COPY(array1, array2 [,src [,dest [,len]])`
- Copy data between arrays. Returns the number of elements copied from array1 to array2.

Syntax

`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.

NOTE: 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

Processing:

```
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 "")
```

COS()

Syntax:

Then: $\text{result} = \cos(\text{angle})$

Version Ref: 4.8

Description:

Trig Function to provide angles given in radians. Returns the COSINE of an angle.

Examples:

Then: $\text{radian_value} = \text{COS}(90)$

COSH()

Syntax:

Then: $\text{result}=\text{cosh}(\text{angle})$

Version Ref: 5.0

Description:

Trig Function to provide the hyperbolic angles given in radians. Returns the hyperbolic COSINE of an angle.

Examples:

Then: $\text{radian_value} = \text{COSH}(90)$

CREATE()

Syntax:

Then: handle = CREATE(filename [,mode])

"handle" is the file handle.

"filename" is the name of the file to create.

"mode" represents the permissions given to the file that gets created. For convenience, if the number starts with a zero, the rest of the number is treated as octal. The binary value corresponds to the 9 bits in the Linux directory listing "rwxrwxrwx". Note that Windows systems only use the top two bits of the value. The rest are ignored.

NOTE: For compatibility purposes with Unix systems, you may also specify the name of this command as CREAT e.g. CREAT(filename).

Version Ref: 4.5 (not included in filePro Lite)

Version Ref: 5.7.04 "mode" parameter was added

5.0.5 CREAT() added as alternative to CREATE()

Description:

Creates a new file, and opens it.

New function to get error code for ENCRYPT()/DECRYPT()

failure. Version Ref: 5.8 (not included in filePro Lite)

status = CRYPTERROR([format])

If "format" is omitted, or zero, then the value is returned as a numeric error number, or zero for "no error". If "format" is "1", then the value is returned as a string. Other values for "format" are undefined. (If ENCRYPT/DEcrypt fails, a null string -- "" -- is returned.)

CURSOR

Syntax:

Then: CURSOR ON

Then: CURSOR OFF

Then: CURSOR DEFAULT

Version Ref: 5.0

Description:

Forces the cursor on/off, or restores the default behavior.

Example:

```
CURSOR OFF
```

```
xx = LISTBOX(array)
```

```
CURSOR DEFAULT
```

NOTE:

filePro will not automatically restore the default cursor behavior. If you CURSOR OFF without ever executing CURSOR DEFAULT, the cursor will remain off until you exit *clerk or *report.

CURSORPATH

Syntax:

Then: CURSOR PATH ON

Then: CURSOR PATH OFF

Version Ref: 5.0.6

Description:

Allows you to turn off forced cursor path in fileProGl. Default is ON meaning that fileProGl enforces screen cursor path

Note: Same as MOUSE PATH

DACOS()

Syntax:

Then: result = DACOS(xx)

Version Ref: 4.8

Description:

Trig Function to provide Arccosine of an angle in degrees.

Examples:

Then: angle_in_degrees = DACOS(90)

DASIN)

Syntax:

Then: result = DASIN(xx)

Version Ref: 4.8

Description:

Trig Function to provide Arcsine of an angle in degrees.

Examples:

Then: angle_in_degrees = DASIN(90)

DATAN()

Syntax:

Then: result = DATAN(y,x)

Returns the arctangent of (y/x) in degrees.

Then: result = DATAN(n)

Trig Function to provide the arctangent of tangent value (n).

Version Ref: 4.8

Description:

Trig Function to provide Arctangent of an angle in degrees.

Examples:

Then: angle_in_degrees = DATAN(90)

Then: angle_in_degrees = DATAN(90,30)

DCOS()

Syntax:

Then: result = DCOS(xx)

Version Ref: 4.8

Description:

Trig Function to provide the cosine of an angle in degrees.

Examples:

Then: angle_in_degrees = DCOS(90)

DEBUG

Syntax:

```
Then: DEBUG ON
Then: DEBUG OFF
```

Version Ref: 3.x

Description:

Turns the runtime debugger on or off from within a processing table.

Examples:

One way to use the command is to set a flag. For example, if you type "Y" in a debug-on yes/ no field, the program turns it on:

```
@wef12 IF:
Then: input dt(1,yesno) "Debugger on?"
If: dt eq "Y"
Then: debug on
Then: end
```

IMPORTANT: The debugger will continue to operate until you turn it off. For example:

```
@wef13 IF:
Then: debug off; end
Test certain parts of a "live" system. in such a way that only you see the debugger, others work normally.
```

```
@wlf14 If: @id eq "root"
Then: debug on
Turn on the debugger if a value is not "right".
If: aa ne "right value"
Then: debug on
```

Version 6.0.00

debug now will accept long variables as break points

The scope of a longvar is different from a normal dummy field. Technically, longvar is not at a true global scope, and isn't available in the automatic processing table. Declaring it 'g' only will work across records, but not tables, declaring it GLOBAL will fix that, but it has to be matched with an

EXTERN in the other prc table.

There are also enhancements to debug:

```
DEBUG ON FIELD PA
DEBUG ON FIELD Invoice_Total
DEBUG ON LINE 1211
```

DECLARE

Syntax:

DECLARE LOCAL	The variable is visible only to the current prc table.
DECLARE GLOBAL	Makes the variable visible to other prc tables.
DECLARE EXTERN	Refers to a variable DECLARED GLOBAL in another table.
DECLARE	Same as DECLARE LOCAL.

Version Ref: 4.8 (not included in filePro Lite)

Description:

Declare global and local dummy variables with LONG names of up to 64 characters. Only applies to Define Processing. DECLARE variables are not supported on Screens or Output Formats. A declared variable can hold up to 32127 characters.

Caution: Avoid the use of reserved words such as "select", "max", etc. as variable names since these are functions and will not be properly interpreted as variable names. Although the processing will pass the syntax check, it will produce unwanted results.

Note: The meaning of GLOBAL when used with DECLARE is different than the meaning of the global ".g" flag used when specifying a field's type and length. The global ".g" tells filePro that the value of the field spans record to record. DECLARE GLOBAL tells filePro the field spans processing tables.

DECLARE variables can be used anywhere in processing that the classic 2-letter dummy fields could be used. If one table has DECLARE GLOBAL, and another table has the same name using DECLARE LOCAL, each table uses its own copy of that name.

Example:

```
Then: DECLARE GLOBAL LastName(20,*,g), FirstName(20,*,g)
Then: DECLARE LOCAL YTDSales(10,,2)
```

Added Version 6.0.01

Added new DECLARED function to check if an array or longvar is defined, meaning it is either declared LOCAL or GLOBAL or is declared EXTERN but has a matching GLOBAL definition. x=DECLARED(var) Where var is either a longvar or an array. Where x is the return value. Returns 0 if the variable is not fully defined. Returns 1 if the variable is fully defined.

Added Version 6.1.01

Added the ability to assign directly to a longvar when declaring.

Example:

```
Then: DECLARE LastName(20,*,g)="John"
```

ENCRYPT / DECRYPT (not included in filePro Lite)

Syntax

result = ENCRYPT(data,method,key [,nonce])

result = DECRYPT(data,method,key [,nonce])

where

data	the data string to be encrypted.
method	the encryption method used e.g. Blowfish, RC2, AES, etc.
key	the character sequence used as the primary base to encrypt or decrypt the "data" field.
nonce	the encryption mode used by filePro. If nonce is not specified, filePro generates one.

DECODE

(not included in filePro Lite)

```
v5.7.0 str = DECODE(str,exp)
```

- Converts the text found in the string in exp by method str. The resulting str is returned. Method str must be either "ROT13" or "BASE64"

Used to restore ENCODED() data to it's original form

```
v6.2 (6.1.02 USP)
```

Enhanced: Enhanced: Method str can now also be "URL" for URL percent encoding.

Example

Processing:

```
Then: a = "guvf vf n grfg"  
Then: x = DECODE("ROT13",a) ' x contains "this is a test"  
Then: a = "dGhpcyBpcyBhIHRlc3Q="   
Then: x = DECODE("BASE64",a) ' x contains "this is a test"  
Then: a = "this%20is%20a%20test"  
Then: x = DECODE("URL",a) ' x contains "this is a test"
```

The ROT13 method is a simple letter substitution cipher, **a=n, b=o, c=p, d=q, ..., m=z, n=a, ..., y=l, z=m** All numbers and symbols remain unchanged.

The BASE64 method (also known as MIME) was developed to allow binary data to be transferred over media that are designed to deal with text data.

The URL method (also known as percent-encoding) is a method to encode data in a uniform resource identifier (URI) using only the US-ASCII characters legal within a URI.

DELETE

Syntax:

```
Then: DELETE
Then: DELETE filename
```

Version Ref: 3.x

Description:

The DELETE command is actually executed AFTER the END statement of the processing table. You could put a DELETE statement as the very first thing on a processing table, then execute hundreds of lookups, calculations, etc., and they would all happen. When the END statement (or the end of the processing table) is reached, the DELETE of the current record takes place.

IMPORTANT: To delete records in another file, you must first do a lookup to the particular record that you want to delete and then execute the "DELETE filename" command where filename is the name of the lookup.

```
Then: lookup arch=oldinvoices k=1 i=a -nx
If: not arch
Then: end
Then: delete arch ; end
```

Be very sure to use the looked-up file's name (or alias) along with the DELETE command or you will inadvertently delete the record you are standing on in the current file!

Technical Note:

Deleting a filePro record with the DELETE command does not reclaim the disk space used by the deleted record. Instead, the record is marked as available for use. In other words, a file comprised of 100 records taking up 100Kb of disk space will stay 100Kb in size even after 30 records are deleted from it. On Unix systems, deleted records are placed on a free-chain. This means available empty (deleted) records are marked with pointers dictating which record will be obtained next when a free record is called for by filePro. This free-chain appends the last deleted record to the end of the free-chain list so that it will be used first when obtaining a free record.

To actually reclaim the disk space taken up by deleted records, a special processing must be run (see compressing files). This processing is usually a third party program specially designed to perform this function. The same results can be obtained from within filePro by copying all the used records in a file to a second (or spin off file), and then renaming this file with the original name of the first file. (This can be easily done with fpCopy.)

Restrictions:

DELETE is available for input and output processing.

DIM

Related Commands

CLEAR
SET

Syntax:

Then: DIM array[n]

Defines an array of "n" fields.

Version Ref: 3.x

Description:

DIM (dimension) defines arrays for use in repetitive operations and with MENU (see the MENU command for details). You can also use DIM to match an array to a list of real, dummy, or lookup fields.

Mapping array fields to real fields is another way to give fields aliases. It is also a way to override the real fields' types and lengths with arrays, you can create fields with more than one length and edit within a single file.

IMPORTANT: In the below examples note that the brackets [] can be used only for the number of array elements where as parentheses () must be used for the length and edit of the array. DIM array[n](l,e)

Examples:

Then: DIM cats[5]

Dimensions (or builds) an array of 5 elements called "cats". The five array elements are referred to as cats["1"], cats["2"], cats["3"], cats["4"] and cats["5"]. They can also be referred to as cats[n] as long as n has a value between 1 and 5. For any size array (any number of elements), the subscript value [n] can NOT be less than or equal to 0 and can not be higher than the highest element number. If this happens, a fatal error will occur and the processing table will exit you from the program with the error message "array index out of range."

IMPORTANT: A non-literal number is used to actually dimension an array. In the example shown above, the number 5 in DIM cats[5] does NOT mean the contents of field 5. This is the only place throughout all of filePro where a 5 not surrounded by quotes does not mean the contents of field 5. Be very aware of this, as it will cause you great difficulty if you use this index subscript improperly. After an array is defined (the actual DIM statement itself), you should never use a non-quoted number as a subscript for that array. You should always use a literal or a dummy field. If you use a number not surrounded by quotes, filePro will use the value found in that field and substitute it for the array index, and it had better be within the boundaries of the array size or you will be dumped out of the processing table. There are rare, very rare times when you might want to use a real field as the subscript of an array. Just be aware of how this will work if you do ever do it.

The fact that the subscript (or index) of the array can be an expression is the feature which makes arrays so valuable. You can start a variable equal to "1" and use it as the index, do something with that element of the array, increment the variable by "1" and do something to or with the next element of the array. You can repeat this loop for as many elements as there are in the array, being careful not to go past the highest element number. Looping through arrays in this fashion is a powerful programming construct that you will use often to reduce the number of lines of code you must write.

Other DIM formats

Then: DIM array[n](l,e)

Array of "n" fields with length "l" & edit "e"

Examples:

Then: DIM procs[3](5,.2)

Array of 3 fields, making them all 5-character, two-decimal-place numbers.

Then: DIM array((l,e)(l,e)...))

Array with fields each having individual length "l" and individual edit type "e". Has as many terms as you specify.

Example:

Then: dim fred((8,.2) (1,yesno) (8,mdy/))

The example above creates a three-field array, the first of which is an eight-digit decimal number, the second of which is a one-character "yesno" field, and the third of which is an eight character date field.

NOTE: When you use this option, you may run out of space on a line before you come to the end of your assignments. You can continue from line to line, however, by breaking the line of assignments between sets of parentheses, and by making sure that you don't add the final outside parenthesis until you're done with the list:

Then: DIM ants((10,*)(5,sex)(8,mdy/)

Then: (12,*) (8,mdy/) (20,allup))

Then: DIM address((30,uplow)(30,uplow)(15,uplow)(2,state)(5,zip))

Then: DIM array[n]:m

Array of n fields starts with its field 1 at the file's field number "m".

Then: dim items[10]:44

The code shown above builds a ten-element array that starts at field 44 of the current file (the file in which the processing resides). This means that the array elements "overlay" or are congruent with fields 44 through 53. Referring to or assigning to field 45 is the very same thing as referring to or assigning to items["2"]. Field 47 is equivalent to items["4"], etc.

DIM array[n]:lookup(m)

Maps to (overlay) a looked-up file's fields.

Then: dim phones[4]:CLIENTS(3)

A 4-field array taken from a lookup named CLIENTS starting at field 3 in CLIENTS. The elements (fields) of this array would coincide with fields 3 through 6 of the CLIENTS file. Changing any of the fields in this array will change the coinciding field in the CLIENTS file.

The following statement would change field 5 in the CLIENTS file to this phone number.

Then: phone["3"]="(201) 427-3311"

Commands used with Arrays

clear array

Sets each element of "array" to blank.

Then: clear phones <- clears the array named "phones".

v6.1 (6.0.02 USP)

EXTERN and GLOBAL arrays

DIM GLOBAL name(size)

DIM EXTERN name

Only non-aliased arrays can be declared GLOBAL/EXTERN. Functions similar to GLOBAL/EXTERN longvars.

v6.1 (6.0.02 USP)

Added new array size function to get the size of an array. Can be used with GLOBAL, EXTERN, LOCAL, and SYSTEM arrays.

x=ARRAYSIZE(array)

Where array is the name of the array.

Where x is the returned size of the passed array.

v6.1 (6.1.01 USP)

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"]
```

DISPLAY

Syntax:

```
Then: DISPLAY
Then: DISPLAY exp
```

DISPLAY refreshes all fields on the current screen to their current value.

DISPLAY exp switches to screen "exp" and shows most current field values.

Version Ref: 1.x

Description:

DISPLAY is similar to SCREEN, except that instead of putting the user in update mode, DISPLAY simply shows effects of processing up to that point. DISPLAY is especially useful with "when entering field" and "when leaving field" processing.

DISPLAY, by itself, will refresh all the fields (real, dummy, and system maintained) to their current values; the screen backdrop itself will not be redrawn, but all field data on the screen will be refreshed to its current value. In other words, if no argument is given to the DISPLAY command, the program redisplay only the visible fields and leaves all other screen text and messages unchanged.

DISPLAY "exp", will do the same thing but cause the specified screen (whatever "exp" resolves to), to also be freshly displayed. In other words, the current values of all fields will not just pop in on the existing screen backdrop, the whole screen will be redrawn.

IMPORTANT: DISPLAY is a most necessary command for interactive processing. For instance, if you add two fields together in an @mf process, the operation will take place. If the result of the calculation (dummy field, or real field) is on the screen, the user will see no change in this field until you issue a DISPLAY statement to bring it to its current value.

Restrictions:

DISPLAY can be used on input processing tables only.

Examples:

```
@mf4 If:
Then: 4="OPERATOR" ; end
```

will appear to do nothing from the user's viewpoint...although it did happen, field 4 was changed.

```
@mf4 If:
Then: 4="OPERATOR" ; display ; end
```

accomplishes the desired result. As soon as the operation is finished, the user is made aware of it. If field 4 is on the screen, the user will see immediately that it is now equal to "OPERATOR".

To display the current screen:

```
Then: display
```

To switch to another screen:

```
Then: display s
```

where "s" is the screen number or letter.

To switch to a named screen:

```
Then: display "name"
```

Make sure you enclose the screen name in quotes if it is a literal name.

DLEN()

Syntax:

Then: a=DLEN(exp)

Version Ref: 4.1

Description:

Returns the display length of a string. This function is useful for aligning or centering data on the screen.

Examples:

Then: aa=3<4 ; show"@the display length of aa is" < dlen(aa)

Note: If you want the display length of a cast dummy variable, in other words, one that has been given a length and edit type, you must do a small trick to find the real display length. Otherwise, DLEN will return the predefined length of the field, regardless of what data is in the field. To fix this, put nulls around the field and take the DLEN of that expression.

Then: aa(5,.0)="12"3" ; show"@incorrect display length is" < dlen(aa)

Then: show"@correct display length is" < dlen(" {aa}")

Instead of displaying 5 as the display length, surrounding the variable with nulls gives a correct result.

Note: DLEN() had an undocumented limit of 255 characters on the input string prior to release 5.0.12. The input is now unlimited, and the output is limited to 4095.

DOEDIT()

Syntax:

Then: xx=DOEDIT(val_expr,edit_expr)

Then: xx=DOEDIT(val_expr,edit_expr,len_expr)

"val_expr" is the value to be passed through the edit.

"edit_expr" is the name of the edit to apply.

"len_expr" is the length of the resulting value.

If len_expr is not specified, the length of val_expr is used.

Return value is a field of edit type edit_expr and length len_expr, if specified. Otherwise, the length is the same as val_expr.

Note: If "edit_expr" is not found, then it is treated as a NOEDIT.

Version Ref: 4.5

Description:

Performs an edit on a value, and returns the result.

Examples:

Right justify and zero-fill field "partno", a (10,*) field.

The old method:

Then: xx(10,rj) = partno ; xy(10,rj0) = xx ; partno = xy

The new method:

Then: partno = doedit(doedit(partno,"rj"),"rj0")

Note how the two fields that were previously needed exclusively for this conversion, can be eliminated by using DOEDIT.

dokey

Tells filePro to execute the keystroke as if the user pressed it (similar to `PUSHKEY`). Only the first character is used, and it need not be the same as the current `@key` event. `DOKEY` does an implicit `END` after running the default behavior.

`DOKEY` does not interfere with `PUSHKEY`, and in fact, they can be used in conjunction.

Example:

```
@keyB if: ' to trap "B" and then load the "mybrowse" browse format
then: pushkey "f|mybrowse[ENTR][ENTR]" ; DOKEY "B"
```

The pushkey does a F-File, L-Load, mybrowse, Enter to select the browse, and then Enter to execute the browse.

Example:

```
@keyX
then: msgbox "Are you sure you want to exit?","", "YN"
if: @bk eq "Y"
then: DOKEY "X" "Note that DOKEY does an implicit END"
then: msgbox "Exit cancelled"
then: end
```

DOM()

Syntax:

Then: a=DOM(date_expr)

Version Ref: 4.1

Description:

Returns the day of month for a date expression.

Examples:

Then: da = DOM(aa)

Then: show "You were born on a " <DOM (bd,"2")

DOQ()

Syntax:

Then: a=DOQ(date_expr)

Version Ref: 4.1

Description:

Returns the day of the quarter for a date expression.

Examples:

Then: da = DOQ(aa)

Then: show "Your payroll period started on a " <DOQ (bd,"2")

DOW()

Syntax:

Then: a=DOW(exp1,exp2)

If exp2 is "1", returns the 3-letter abbreviation.

If exp2 is "2", returns full day name.

If exp2 is "0" or omitted, returns 1-7 where 1=Sun.

Version Ref: 4.1

Description:

Returns day of week for a date expression.

Examples:

Then: da = DOW(aa)

Then: show "You were born on a " < DOW (bd,"2")

DOY()

Syntax:

Then: a=DOY(date_expr)

Version Ref: 4.1

Description:

Returns the day of the year for a date expression.

Examples:

Then: da = DOY(aa)

Then: show "You were born on a " < DOY (bd,"2")

DROP

Syntax:

```
DROP
DROP ALL BEFORE
DROP ALL AFTER
DROP ALL
```

Version Ref: 4.1

Description:

Causes current record to be left out of browse lookup box. (Used with PRC option in lookup). DROP ALL Allows all remaining records in a file to be quickly dropped from a browse window.

Example:

```
If: balance lt "400.00"
Then: DROP; end
```


DSIN()

Syntax:

Then: $\text{result} = \text{DSIN}(xx)$

Version Ref: 4.8

Description:

Trig Function to provide the sine of an angle in degrees.

Examples:

Then: $\text{angle_in_degrees} = \text{DSIN}(90)$

DTAN()

Syntax:

Then: result = DTAN(xx)

Version Ref: 4.8

Description:

Trig Function to provide the tangent of an angle in degrees.

Examples:

Then: angle_in_degrees = DTAN(90)

DTOR()

Syntax:

Then: $\text{result} = \text{DTOR}(\text{angle})$

Version Ref: 4.8

Description:

Trig Function to convert degrees to radians.

Examples:

Then: $\text{angle_in_radians} = \text{DTOR}(90)$

EDIT()

Syntax:

Then: a=EDIT(n)

n is equal to a real, dummy or system maintained field.

Version Ref: 3.x

Description:

Returns the edit-type of a field.

Examples:

TT(10,EDIT(3))

Assigns dummy field TT with length 10 & the same edit type as field 3.

When you use EDIT with LEN, you can assign both the length and the edit type of one field to another.

Then: aa(len(3),edit(4))

Assigns the length of field 3 and the edit type of field 4 to dummy variable aa. If you ever change the length of field 3 or the edit type of field 4, aa's length and edit type will change as well.

Restrictions:

EDIT can be used on all processing tables on both condition and action lines.

ENCRYPT / DECRYPT (not included in filePro Lite)

Syntax

result = ENCRYPT(data,method,key [,nonce])

result = DECRYPT(data,method,key [,nonce])

where

data	the data string to be encrypted.
method	the encryption method used e.g. Blowfish, RC2, AES, etc.
key	the character sequence used as the primary base to encrypt or decrypt the "data" field.
nonce	the encryption mode used by filePro. If nonce is not specified, filePro generates one.

ENCODE

(not included in filePro Lite)

v5.7.0 `str = ENCODE(str,exp)`

- Converts the text found in the string in `exp` by method `str`. The resulting `str` is returned. Method `str` must be either "ROT13" or "BASE64"

v6.2 (6.1.02 USP)

Enhanced: Enhanced: Method `str` can now also be "URL" for URL percent encoding.

Example

Processing:

```
Then: a = "this is a test"
Then: x = ENCODE("ROT13",a) ' x contains "guvf vf n grfg"
Then: x = ENCODE("BASE64",a) ' x contains "dGhpcyBpcyBhIHRlc3Q="
Then: x = ENCODE("URL",a) ' x contains "this%20is%20a%20test"
```

The ROT13 method is a simple letter substitution cipher, **a=n, b=o, c=p, d=q, ..., m=z, n=a, ..., y=l, z=m** All numbers and symbols remain unchanged.

The BASE64 method (also known as MIME) was developed to allow binary data to be transferred over media that are designed to deal with text data.

The URL method (also known as percent-encoding) is a method to encode data in a uniform resource identifier (URI) using only the US-ASCII characters legal within a URI.

END

Syntax:

Then: end

Version Ref: 3.x

Description:

END means "stop processing here; do not continue down the table." It's not needed at the end of the processing table - processing ends automatically when it runs out of elements.

Note : Use RETURN instead of END to quit a subroutine started with the GOSUB command.

EOM()

Syntax:

Then: xx=EOM(date_expr)

Version Ref: 4.5

Description:

Returns the end of a month for any date expression. If the date expression is left blank, returns the end of the current month. date_expr means that the field has to be of a "date" edit type. It cannot be uncast.

Examples:

Then: payroll_month_ends = EOM(payroll_date)

Returns the last day of the month for the payroll_date variable.

Then: current_month_ends = EOM()

Returns the last day of the current month.

EOQ()

Syntax:

Then: `xx=EOQ(date_expr)`

Version Ref: 4.5

Description:

Returns the end of a quarter for any date expression. If the date expression is left blank, returns the end of the current quarter.

Examples:

Then: `payroll_qtr_ends=EOQ(payroll_date)`

Returns the last day of the quarter for the `payroll_date` variable.

Then: `current_quarter_ends=EOQ()`

Returns the last day of the current quarter.

EOY()

Syntax:

Then: `xx=EOY(date_expr)`

Version Ref: 4.5

Description:

Returns the end of a year for any date expression. If the date expression is left blank, returns the end of the current year.

Examples:

Then: `payroll_year_ends=EOY(payroll_date)`

Returns the last day of the year for the payroll_date variable.

Then: `current_year_ends=EOY()`

Returns the last day of the current year.

ERRNAME()

Syntax:

Then: `xx = ERRNAME(num_expr)`

where "num_expr" is the system error number.

Return value is a text field, the length is the same as the length of the error message text.

Note: System error messages are highly system dependent, and you should not rely on the same error having the same text description on different machines. This function's purpose is to allow some feedback to the user in case of an error.

Version Ref: 4.5

Description:

Returns the name of the specified system error.

Examples:

Then: `h = open("/tmp/file.dat")`

If: `h lt "0"`

Then: `errorbox "Cannot open file. Error is " < ERRNAME(h)`

ERRORBOX

Syntax:

Then: ERRORBOX(row,col) message,prompt,keylist

Version Ref: 4.1

Description:

The ERRORBOX processing command lets you display an error message in a popup window on the screen. It will display a string "message" in a popup window at a designated position on the screen until the user presses ENTER or a key from a specified list. The key pressed by the user can be captured and acted upon appropriately.

Row and Column

If row and column are supplied, the upper left corner of the window appears at the row and column coordinates. If row and column are not supplied, the window is centered on the screen. The upper-left corner of a screen is row,col ("1","1").

Width and Height

The longest line of text determines the width of the window. The number of text lines determines the height of the window.

Message

The message to appear in the window is a string expression. You can add additional lines to this message by placing a "\n" at the point where you want a new line to begin. All of the SHOW codes are available to ERRORBOX. For example:

```
Then: errorbox "Text line 1\nText line 2"
```

Prompt

Prompt is the prompt you want to appear telling the user what keys can be used to remove the window (and/or take certain actions) to continue. The prompt appears in the lower-right corner of the window. If prompt is not specified, then the default prompt of "Press ENTER" is used.

Keylist

Keylist is the list of keys, in addition to ENTER, that will remove the window. These keys can be trapped and acted upon with further processing. If keylist is not specified, the ENTER key is the default for removing the window.

NOTES:

ENTER is the special key defined as ENTR. ERRORBOX will display whatever is set as the keylabel for this key.

After the message box is executed, @BK is set to the keystroke entered.

The arguments to ERRORBOX (row/col, the message, the prompt and the keylist) can be variables. If they are variables (or real fields), you must NOT put quotes around them. If they are literals, such as the keylist below (SLN), then they must be in quotes. Case is not significant (except if the key is a shifted key such as "@", then a 2 will not work, only a shifted 2 "@" will work).

ERRORBOX is exactly the same thing as MSGBOX except for the colors which are assigned to it. The colors for ERRORBOX are ERRORNORMAL and ERRORINVERSE. This environmental variables allow you to give a consistent look and feel to your programming by showing "errors" in one set of colors and "messages" (or navigational questions) in another set of colors.

Examples:

```
If: 4 ne "0"
Then: end
Then: o="NO money on this record. Save it, Delete it, or Fix it?"
Then: m="(s/d/f)=>"
Then: errorbox("10","5") o,m,"SDF"
If: @bk eq "S"
Then: end
If: @bk eq "D"
Then: delete ; end
If: <== the F key in keylist will default here, so will ENTER.
Then: restart
If: exists(fn) gt "0"..
Then: errorbox "File already exists!\nContinue?",
"Press \rY\r or \rN\r", "YN"
If: @BK eq "Y"
Then: gosub mkfile
```

ESCAPE

Syntax:

Then : ESCAPE

Version Ref: 3.x

Description:

ESCAPE, in effect, presses <ESC> for the user, and saves the data on the current record. ESCAPE is used only with when-processing.

Examples:

```
@wefn If:  
Then: escape
```

```
@mf6 If: aa le "5"  
Then: escape
```

NOTE: ESCAPE can be only be used with @when processing.

EXISTS()

Syntax:

Then: EXISTS(filename)

Return Values:

"1" if the file exists,

"0" if the file does not exist

a negative number if a system error occurs

Note: "fn" can be a file name as well as a directory path.

Version Ref: 4.1

Description:

Checks for the existence of a file on the system, where filename is an expression that resolves to either a relative or full pathname.

This command is valuable if you need to make sure a file already exists before you do a particular task, otherwise the task may go into an error condition. This function allows you to avoid such situations. It is also valuable when used as a configuration tool. For example, by placing certain files in a user's environment, you can run processing based on their existence.

Examples:

Then: fn="/"&dp&"/"&4

If: exists(fn) le "0"

Then: msgbox "File does not exist." ; goto loop

EXIT

Syntax:

Then: EXIT num_expr

"num_expr" is the program's exit value.

NOTE: The exit value must be between 0 and 255, inclusive. Standard practice is to use zero as a success indicator, and other values to indicate failure.

Version Ref: 4.5

Description:

Terminates running of the filePro program and sets the exit value.

EXIT records the data up to the point in the table at which it is encountered and takes the user completely out of Inquire, Update, Add or Request Output.

When used in Request Output, all records that have been processed before the EXIT occurs are printed.

Note: The current record is written out first as is.

Retrieving the exit value

EXIT is used in conjunction with a batch files or script files.

MS-DOS, the programs exit value can be checked by using the "IF ERRORLEVEL" batch command.

Unix, you can use "if dclerk ..." to check for success or failures, or you can check "\$?" for specific values.

Indicate & test failure.

Example:

Assume the following code is on a report that is at the head of a long group of reports. You only want to run the rest of the reports if this one runs correctly. The menu script might look something like this. Remember - the filePro code shown is really on "report1" and returns to the environment a good or bad status. The "if \$?..." line is how you test whether this report failed or not. If it fails, i.e., if the exit value is not 0, the "exit" command (this time the Unix shell's version of exit) takes you completely out of the script and the following reports do not get executed. Otherwise, the test falls through, and the reports run.

```
/appl/fp/dreport firstfile -f report1 -a ...
```

```
Then: m(1,0)="1"
```

```
Then: lookup control r=m -nx
```

```
If: not control
```

```
Then: errorbox "Cannot read control file!" ; exit "127"
```

```
Then: exit "0"
```

```
f $? != "0" && exit
```

```
/appl/fp/dreport otherfile -f otherreport2
```

```
/appl/fp/report thatfile -f otherreport3
```

```
/appl/fp/dreport samefilefile -f otherreport4
```

EXP()

Syntax:

Then: result = EXP(n)

Version Ref: 4.8

Description:

Exponent function (e^n)

Examples:

Then: na="4.6051702"

XX=exp(na)

Returns the natural antilog value of "100"

EXP10()

Syntax:

Then: result = EXP10(n)

Version Ref: 4.8

Description:

Base 10 exponent (10^n)

Examples:

When na="2.00000"

XX=exp10(na)

Returns the base10 antilog value of "100"

EXPORT (not included in filePro Lite)

Syntax:

Then: export type filename options
filename can be an expression by assigning it as an alias as in:
Then: export type aaa=(exp) options

Version Ref: 4.8 (enhanced) (not included in filePro Lite)

Description:

EXPORT creates spin-off files to be used with other programs. EXPORT can create two types of files, ASCII files or application specific files:

Structured ASCII (Fixed length ASCII):

Then: EXPORT ASCII name -X [-A]

Delimited ASCII:

Then: EXPORT ASCII name R=F O=C [-A]

where "r", "f", "o", "c" are delimiters for the ASCII file and optional flag [-A] allows you to append to the end of a file. If the [-A] flag is used and the file does not exist, filePro will create it.

"R" is the record indicator.
"F" is the field separator.
"O" is opening field delimiter. <- Not often used.
"C" is closing delimiter. <- Not often used.
-A means to append the existing file.

Example:

Then: EXPORT ASCII test=test.txt R=J f=^ -A

The above would append a file named "test.txt" in your current directory containing a "linefeed" character at the end of each record with a "^" between each data field. Notice the different use of "^" symbol. In the first case it is used as a control character to send a "linefeed" and in the second case it is considered as a literal "^" since it is preceeded with a "\" backslash. Also notice that the "O" Opening and "C" Closing delimiters are not used in the above example. These are also optional and seldom used.

IMPORTANT: You may use a few special codes with the delimited ASCII format as delimiters or field separators:

\r for carriage return
\f for form feed
\n for new line
\t for tab

You can also use punctuation characters, ASCII codes, and control codes as delimiters and separators. Use the caret (^) followed by a letter to indicate a "CTRL" character e.g. ^J, ^L, ^M, etc. Refer to the "Character Table" in "Define Processing" help for "CTRL" characters. In cases where you want to use the caret (^) as a literal, remember to preceed it with a backslash e.g. \^.

Note: -A flag is only available for the "EXPORT ASCII" function.

Specialized Export Formats:

Then: EXPORT DIF name
Then: EXPORT MULTI name [R=n | C=n]
Then: EXPORT WORD name
Then: EXPORT WordPerfect name

Then: name(1)=aa ; name(2)=ab ; ... ; name(n)=zz

Using EXPORT requires a minimum of two statements: the first defines the name of the spin-off file (sometimes called a merge file); the second (and additional statements, if necessary), tells the program which fields in the exported file are to accept the data.

Each "EXPORT" definition must be alone on its "Then" line. Use at least three characters for the filename, and don't start with a number.

Each field assignment specifies the data to be put into the fields of the exported file.

Then: mergename(n)=exp

where "mergename" is the filename (or alias of the exported file), "n" is the specific number designating the desired field in the exported file, and "exp" is the supplied data. Note that you can assign results of text and math formulas to merge fields and that you can use literals. For example:

Then: mergename(2)="Show me";mergename(4)=3;mergename(8)=aa+bb

You can, of course, put more than one assignment per line (separate them with semicolons), and use more than one line.

Structured ASCII (Fixed length ASCII)

Then: EXPORT ASCII name -X

Creates an ASCII file called "name" in structured format.

This is a very useful export format. The entire field is placed in the export file. No field separators are required because each field takes up exactly its length in characters whether there is data in the field or not. The record delimiter is a new line by default. This format is becoming more and more of a standard means for exchanging files between systems and applications.

NOTE: Structured ASCII files can be immediately read by filePro. All that is needed is the layout of the field structure, i.e., the map of the file. This list of fieldnames, lengths, and data types is simply entered into filePro as an "alien" file under Define Files. The structured ASCII file is set as the "Data File Name", and it can then be instantly read and written to by filePro programs. Indexes can be built on the alien file as well. (There are only two important differences between alien files and regular filePro files. Regular filePro files have a 20 byte header at the beginning of each record that holds system maintained information such as record creation date, created by, etc. Secondly, an alien file can not "reclaim" deleted record space by adding such records to a "freechain" of available records [a multi-user function].)

IMPORTANT: Make sure to consider the record delimiter when using an alien file. This is usually one or two characters that are not mentioned in the file layout (or map). If your records look like scrambled eggs when you go into Inquire, Update and Add on an alien file, try adding a one-character field to the end of the map. DO NOT RESTRUCTURE THE FILE WHEN PROMPTED! Try viewing the file again in IUA. If it still doesn't look right, try changing the one character field at the very end of the map to a length of 2 characters. Again, DO NOT RESTRUCTURE THE FILE WHEN PROMPTED! This should clear things up and the file will look okay. This last field does not ever have to be shown on the screen or used anywhere. It is a place holder and simply must be in the map to allow it to overlay the data correctly.

The specialized export formats, can be used by various application programs. The DIF format is sometimes available to spreadsheet programs. The MULTI format is used by Multiplan an archaic spreadsheet program. (This export format allows you to use the optional row equals or column equals parameters to designate where exported data will begin on a spreadsheet.) The WORD format is used by WordStar an

archaic word processing program. The WordPerfect format is used by the WordPerfect word processing program. (HINT: The WordPerfect format can be easily read by Microsoft Word. Do not use the WORD format for anything but WordStar.)

ASCII files: EXPORT can create ASCII files in either structured or delimited form. Structured ASCII files set aside a specific number of characters for each field in every record. Fields in the exported file retain their specified length regardless of whether the supplied data for these fields fills that length or not. Each record in a structured ASCII files is separated by a newline. Delimited ASCII files set a specific character as a field separator (delimiter), and, the fields in the export file shrink or expand to fit the length of the data within each field. Delimited files may also employ opening and closing record indicators, however, these special delimiters are not frequently used these days.

Application specific files: EXPORT can create several application specific files. These formats can be used to merge data from filePro into various spreadsheet and word processing applications.

Technical Notes:

Overriding extensions

EXPORT adds default extensions to the merge file name, depending on the parameter. For example, if your merge name is "letters" and your parameter is either ASCII or WORD, the spin off file's name is "letters.wp". Other extensions are ".dif" for DIF and ".sl" for MULTI.

For example:

```
Then: export ASCII aaa=/tmp/rawdata f=r, r=\n
Then: aaa(1)=2 ; aaa(2)=" " ; aaa(3)="P" ; aa(4)=14 ; end
```

filePro will append a ".wp" to this type of an export if you do not already have a suffix on the export filename. The code above will create a file called "/tmp/rawdata.wp". To change this behavior, just add the suffix you would rather have: ".txt", ".doc", etc. To override any of the default extensions ("dif," "sl," or "wp"), set the exported filename equal to the name you want with the new extension. Use a complete pathname if desired:

```
Then: export EXPORTTYPE alias=filename.newext [options]
```

IMPORTANT: Remember that on DOS 3.1 systems an extension can only be comprised of 3 letters.

There is also an environment variable, PFADDWP, that turns the behavior of adding ".wp" to exported filenames off and on. The default is ON.

5.0.14 Change

EXPORT ASCII/WORD would always export the same number of fields, regardless of whether the fields were assigned to on each record, even if they were only referenced in a comment. Now, filePro will only export the number of fields as the highest-reference field actually assigned.

For example:

```
If:
Then: out[1] = 1 ; out[2] = 4
If: xx = "y"
Then: out[3] = 3 ; out[4] = 4
If:
Then: ' out[5] = 5
```

filePro would previously always exported 5 fields. Now, if x="y" is true, it will export 4 fields, and if false will export 2 fields.

To revert back to the old behavior, set PFEXPORTALL=ON.

Overriding Default Directories and Paths

On DOS systems, filePro puts the exported spin off file in the current directory.

On UNIX/XENIX systems, it puts the file in ".../fpmerge" directory (as governed by the PFDIR environment variable and the /etc/default/fppath file). There must be an "fpmerge" directory in the same directory as your "filePro" directory.)

You can place the exported file in any directory (for which filePro has write permission) by setting the exported filename equal to the desired pathname:

```
Then: export exporttype alias=dir\mergename.ext [options]
```

Use the separator mark (\ or /) appropriate to your operating system.

Restrictions:

An "IMPORT filename" definition must be alone on its "Then" line.

Only one EXPORT or IMPORT statement with the same merge file name is allowed per processing table. In other words, you can't have the statement, "EXPORT MULTI List" followed in a few elements by "IMPORT WORD List."

Examples:

ASCII Files

Fixed Length ASCII:

```
Then: export ASCII filename -X
```

When you use the "-X" flag, the program creates one long string per record, maintaining the defined lengths of the filePro fields. In other words, it does not strip blanks on either side of any data. For example, your first data field is 10 characters long, the second is 15 characters long and the third is a 7 character numeric field. The exported string would look like this (without the quotes of course):

```
"John Smith 123.45r"
```

The record delimiter (carriage return or line feed), depends on the operating system.

Delimited ASCII:

```
Then: export ASCII aaa=/tmp/merge.txt f=r, r=\n
```

When you specify field and record delimiters, the program creates one long string per record, by marking each exported field with the designated field separator. It is important to note that BLANK space to the right of any supplied data will be removed. For example, your first data field is 10 characters long, the second is 15 characters long, the third is a 12 character field, and the fourth is a 7 character numeric field. The exported string would look like this (without the quotes of course):

```
"John,Smith,policeman,123.45r"
```

If you make an assignment to an exported field in a delimited ASCII format and the data assigned is null (i.e., the field or expression being assigned is blank), the exported string will still surround the designated exported field with the field separator. For example if the above example was assigned in the following way:

```
Then: aaa(1)=2;aaa(2)=3;aaa(3)=4;aaa(4)=19
```

and field 4 on this particular filePro record was blank, the exported string would look like:

```
"John,Smith,,123.45r"
```

Microsoft Excel Import Files

Although Microsoft Excel will import various formats, a "Comma Separated, Quote Delimited" format is more commonly used. Use the filePro **EXPORT word** option to create this type of file.

Simple:

```
Then: export Word cust
Then: cust(1)=1 ; cust(2)=3 ; cust(3)=4<6
```

Elaborate:

```
Then: 'The following will create a merge file of name/addresses
Then: 'that will automatically close up blank lines when merging
Then: FNAME:1
Then: LNAME:2
Then: COMPANY:3
Then: ADDRESS:4
Then: CITY:5
Then: STATE:6
Then: ZIP:7
Then: COUNTRY:13
Then: export Word merge=C:\wpmerge.wp
Then: dim array[4](50);clear array
Then: i(1..0)="1"
    If: FNAME ne "" or LNAME ne ""
Then: array[i] = "" { FNAME < LNAME;i=i+1"
    If: COMPANY ne ""
Then: array[i] = "" { COMPANY;i=i+1"
    If: ADDRESS1 ne ""
Then: array[i] = "" { ADDRESS1;i=i+1"
    If: CITY ne "" or STATE ne "" or ZIP ne "" or COUNTRY ne ""
Then: array[i] = "" { CITY < STATE < COUNTRY < ZIP;i=i+1"
Then: merge(1)=array["1"]
Then: merge(2)=array["2"]
Then: merge(3)=array["3"]
Then: merge(4)=array["4"]
Then: end
```

Word Processing and Spreadsheets Files

WordPerfect:

Simple:

```
Then: export WordPerfect cust
Then: cust(1)=1 ; cust(2)=3 ; cust(3)=4<6
```

Elaborate:

```
Then: 'The following will create a merge file of name/addresses
Then: 'that will automatically close up blank lines when merging
Then: FNAME:1
Then: LNAME:2
Then: COMPANY:3
Then: ADDRESS:4
Then: CITY:5
Then: STATE:6
Then: ZIP:7
Then: COUNTRY:13
Then: export WordPerfect merge=C:\wpmerge.wp
Then: dim array[4](50);clear array
Then: i(1..0)="1"
    If: FNAME ne "" or LNAME ne ""
Then: array[i] = "" { FNAME < LNAME;i=i+1"
    If: COMPANY ne ""
Then: array[i] = "" { COMPANY;i=i+1"
    If: ADDRESS1 ne ""
Then: array[i] = "" { ADDRESS1;i=i+1"
    If: CITY ne "" or STATE ne "" or ZIP ne "" or COUNTRY ne ""
Then: array[i] = "" { CITY < STATE < COUNTRY < ZIP;i=i+1"
Then: merge(1)=array["1"]
Then: merge(2)=array["2"]
Then: merge(3)=array["3"]
Then: merge(4)=array["4"]
Then: end
```

Note : Later versions of WordPerfect adopted the use of standard merge file formats. If you are using WordPerfect 6.0 or later, try using EXPORT WORD to create a standard CSV format and select the "ASCII Delimited" option in WordPerfect's merge options.

WORD (WordStar not Microsoft Word):

Here is a typical WORD record export:

```
"Smith", "John", "123.45", "01/06/1986" <cr/lf>
```

WordStar uses quotation marks as field delimiters, commas as field separators, and carriage return line feed codes as record separators.

HINT: Microsoft Word can make use of a delimited ASCII file as a merge document.

MULTI:

Below are two EXPORT MULTI statements. Data from fields 22 and 23 in the source filePro file will be arrayed across the spreadsheet starting on row 3. In this example, the exported filename (the one to be merged with the spreadsheet) is called "merge."

```
Then: export multi merge r=3
Then: merge(1)=22; merge(2)=23
```

To place the data in columns, starting with column 2:

```
Then: export multi merge c=2
Then: merge(1)=22 ; merge(2)=23
```

DIF:

Then: export dif merge

Then: merge(1)=22; merge(2)=23

FIELDEDIT()

Syntax:

Then: xx = FIELDEDIT(lookupname,fieldno)

"lookupname" is the name of the lookup file. Use a dash "-" to represent the current file.

"fieldno" is an expression designating the field number for which you want the edit type.

Return value is a text field containing the name of the target field's edit type.

Version Ref: 4.5

Description:

Returns the name of the edit for the specified field in a lookup file.

Examples:

Retrieves the edit type of a lookup field and places it in variable aa.

```
Then: lookup inv=arinvoice k=2 i=a -nx
```

```
Then: aa=fieldedit(inv,"5")
```

Assigns an edit to a field based on the edit type of a lookup field.

```
Then: lookup inv=arinvoice k=2 i=a -nx
```

```
Then: aa(10,fieldedit(inv,"5"))
```

The following code uses other filePro field functions to display the "map" of a lookup file. It displays the field names, lengths, and edit types in a listbox that looks similar to an actual map printout. (Assumes the lookup name is "file".)

```
Then: dim map(100)(60,*)
```

```
Then: no(3,,0) = "1"
```

```
If: numfield(file) eq "0"
```

```
Then: errorbox "No fields" ; end
```

```
loop If: no gt numfield(file) or no gt "99"
```

```
Then: goto done
```

```
Then: map[no] = no & " - " &
```

```
doedit(fieldname(file,no),"*","45") & " " &
```

```
doedit(fieldlen(file,no),".0","3") <
```

```
fieldedit(file,no)
```

```
Then: no = no + "1" ; goto loop
```

```
done If:
```

```
Then: aa = listbox(map,"1",no-"1")
```

Note the use of the DOEDIT() function while building the map entry. This guarantees that all entries will be of the same length, and the columns will be aligned.

FIELDLEN()

Syntax:

Then: xx=FIELDLEN(lookupname,fieldno)

"lookupname" is the name of the lookup file. Use a dash "-" to represent the current file.
"fieldno" is an expression designating the field number for which you want the length.

Return value is a numeric field containing the length of the target field.

Version Ref: 4.5

Description:

Returns the length of the specified field in a lookup file.

Examples:

Retrieves the length of a lookup field and places it in variable aa.

Then: lookup inv=arinvoice k=2 i=a -nx

Then: aa=fieldlen(inv,"5")

Assigns a length to a field based on the length of a lookup field.

Then: lookup inv=arinvoice k=2 i=a -nx

Then: aa(10,fieldlen(inv,"5"))

The following code uses other filePro field functions to display the "map" of a lookup file. It displays the field names, lengths, and edit types in a listbox that looks similar to an actual map printout. (Assumes the lookup name is "file".)

Then: dim map(100)(60,*)

Then: no(3,.0) = "1"

If: numfield(file) eq "0"

Then: errorbox "No fields" ; end

loop If: no gt numfield(file) or no gt "99"

Then: goto done

Then: map[no] = no & " - " &

doedit(fieldname(file,no),"*","45") & " " &

doedit(fieldlen(file,no),"0","3") <

fieldedit(file,no)

Then: no = no + "1" ; goto loop

done If:

Then: aa = listbox(map,"1",no-"1")

Note the use of the **DOEDIT()** function while building the map entry. This guarantees that all entries will be of the same length, and the columns will be aligned.

FIELDNAME()

Syntax:

Then: xx=FIELDNAME(lookupname,fieldno)

"lookupname" is the name of the lookup file. Use a dash "-" to represent the current file.
"fieldno" is an expression designating the field number for which you want the name.

Return value is a text field containing the name of the target field.

Version Ref: 4.5

Description:

Returns the name of the specified field in a lookup file.

Examples:

Retrieves the name of a lookup field and places it in variable aa.

Then: lookup inv=arinvoice k=2 i=a -nx

Then: aa=fieldname(inv,"5")

The following code uses other filePro field functions to display the "map" of a lookup file. It displays the field names, lengths, and edit types in a listbox that looks similar to an actual map printout. (Assumes the lookup name is "file".)

Then: dim map(100)(60,*)

Then: no(3,,0) = "1"

If: numfield(file) eq "0"

Then: errorbox "No fields" ; end

loop If: no gt numfield(file) or no gt "99"

Then: goto done

Then: map[no] = no & " - " &

doedit(fieldname(file,no),"*", "45") & " " &

doedit(fieldlen(file,no),"0", "3") <

fieldedit(file,no)

Then: no = no + "1" ; goto loop

done If:

Then: aa = listbox(map,"1",no-"1")

Note the use of the DOEDIT() function while building the map entry. This guarantees that all entries will be of the same length, and the columns will be aligned.

Version 5.6

FIELDNUM() was added to return the field number of the FIELDNAME of a lookup

FIELDVAL()

Syntax:

Then: `xx=FIELDVAL(lookupname,fieldno)`

"lookupname" is the name of the lookup file. Use a dash "-" to represent the current file.
"fieldno" is an expression designating the field number for which you want the name.

Return value is the contents of the specified field.

Version Ref: 4.5

Description:

Returns the contents (value) of the specified field in a lookup file.

Example:

Retrieves the contents of a lookup field and places it in variable aa.

Then: `lookup inv=arinvoice k=2 i=a -nx`

Then: `aa=fieldval (inv,"5")`

FILENAME()

Syntax:

Then: aa = FILENAME(handle)

Return value

The filename of the specified handle.

Version Ref: 5.0

Description:

Returns the filename of the specified handle.

The handle must be a value returned from OPEN() or CREATE(). Invalid handles return null.

Example:

```
handle = CREATE("/tmp/tempfile");
```

```
...
```

```
name = FILENAME(handle)
```

NOTE:

This is useful in generic CALL routines where the filename isn't known. In a text file, it is possible that the value returned by FILESIZE() is not the same number of bytes that can be read from the file.

FILESIZE()

Syntax:

Then: aa=FILESIZE(handle)

"handle" is the file handle returned by OPEN() or CREATE().

Return value

The size, in bytes, of the file.

Note: In a text file, it is possible that the value returned by FILESIZE() is not the same number of bytes that can be read from the file.

Version Ref: 4.5

Description:

Returns the number of bytes in a file.

FLOOR()

Syntax:

Then: `xx=FLOOR(num_expr)`

"num_expr" is the given number.

"xx" is the resulting value.

Return value is a field of the same edit type and length as num_expr.

NOTE: If the given number is already an integer, the original number is returned.

Version Ref: 4.5

Description:

Performs the FLOOR function, which given any number, returns the next lesser integer.

Examples:

Show the difference between FLOOR(), CEIL(), INT(), and converting to an integer:

Try the example with the numbers: -5.6 -5.5 -5.4 5.4 5.5 5.6 and note the differences among the functions.

Then: `input popup xx(10,,5) "Enter a number: "`

If: `xx = ""`

Then: `end`

Then: `yy(10,,0) = xx`

Then: `msgbox xx & "\nFLOOR:" < FLOOR(xx) & "\nCEIL:" < CEIL(xx)& "\nINT:" < INT(xx) & "\nrounding" < yy`

FLUSHKEY

Syntax:

Then: FLUSHKEY

Description:

Empties the keyboard buffer before returning control to the executing program. This includes any keystrokes sent via the PUSHKEY command that have not already been executed.

FOR

Version Ref: 6.1 (USP6.1.01)

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

- 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.

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.

Example - For Loop

Processing:

```
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
```

WHILE

Version Ref: 6.1 (USP6.1.01)

WHILE cnd DO label

- 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.

Example - While Loop

Processing:

```
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|LOOP UNTIL

Version Ref: 6.1 (USP6.1.01)

LOOP label WHILE cnd

LOOP label UNTIL cnd

- 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.

Example - Loop While

Processing:

```
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
```

FORM

Syntax:

Then: FORM formname
Then: FORM exp

FORM can also be used on the condition line to test whether the form was actually found:

If: not form
Then: [do something]

Version Ref: 4.8

Version 6.0.02

FORM now will allow you to pass a path and filename to use instead of looking in the local directory
Syntax: form "library/zipform"

Description:

Prints a form from within processing. (AUTOMATIC and INPUT processing only.)

Examples:

FORM lets you print forms conditionally; print one form instead of another; and print more than one copy of a particular form. If you have a set of output formats named SALES1, SALES2, SALES3 and so on, and you want to print different forms depending on the conditions. There are at least two ways to do this. First is to set up a series of conditional elements: if field 2, format number, contains "1," then print SALES1, if field 2 contains "2," then print SALES2, etc. The second way is to use an expression, and generate the format name from the contents of field 2:

```
Then: form "SALES" { 2
```

If field 2 contains "3," then SALES3 is printed.

To print more than one copy of a form, you'd write a loop:

```
Then: input aa(2,.0) "Print how many copies? "  
loop  If: aa gt "0"  
Then: form "label"; aa=aa- "1"; goto loop
```

Any processing associated with the form (by virtue of having the same name as the form) is not done. Dummy fields in the input processing tables can be put on this form. For example, if field A on the input processing table is a date field, and field A on the output processing table is a YESNO field, the field that appears on the printed form will be a date and not a YESNO.

Hint: If you have a need to create a form quickly, you can copy a screen by using the "Extended Functions - Import Text File" feature in "Define Output". Keep in mind that you will have to remove the non-text portions of the screen and change "%" field indicators to either exclamation points or asterisks to properly print a form version of your screen.

v6.1 (6.0.03 USP)

FORM WITHPROC "formname"

Added additional command switch to FORM and FORMM commands to allow the associated processing table to run while in input processing. Note: You cannot call the WITHPROC variant from within another form UNLESS the calling form is a processing only form.

FORMERROR()

syntax: xx=FORMERROR()

returns: errno from last FORM or FORMM command.

e.g. 2=file not found, 13=permission error

FORMM

Syntax:

Then: FORMM formname

Then: formm exp

where "name" is the name of the form to be printed and "exp" is any expression that evaluates to a form name.

Version Ref: 4.0

Version 6.0.02

FORM now will allow you to pass a path and filename to use instead of looking in the local directory

Syntax: formm "library/zipform"

Description:

(UNIX and network DOS) - Same as the FORM command, except that it does not close the print spooler. With the spooler open, any other forms sent to the spooler (with FORMM) will be grouped with the original. Use FORM with the last form to close the spooler. All the forms will be printed together, in the order in which they were sent to the spooler. FORMM is available in Inquire, Update, Add only.

Examples:

You could have a multi-page report, with each page a different form. When printing the report, use FORMM for all the pages (in order) up to the last page; for the last page, use FORM or exit Inquire, Update, Add to close the spooler. All the pages of the report will be printed consecutively, in order.

v6.1 (6.0.03 USP)

FORMM WITHPROC "formname"

Added additional command switch to FORM and FORMM commands to allow the associated processing table to run while in input processing. Note: You cannot call the WITHPROC variant from within another form UNLESS the calling form is a processing only form.

FPSTAT()

Version Ref: Version 6.2 (USP 6.1.02)

s = FPSTAT(lookup) - Return map information and basic access attributes for a given filePro lookup.

Parameters:

lookup: A lookup to a filePro file to retrieve 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.

FRAC()

Syntax:

Then: $a = \text{FRAC}(n)$

Version Ref: 4.1

Description:

Returns the fractional portion of a number. (Compliment of the INT function.)

Examples:

Then: $fr = \text{FRAC}(no)$

FREESPACE()

Syntax:

Then: XX=freespace()

Then: XX=freespace(drive)

Returns the available free space (in bytes) of the drive on which the main file's map is located.

For UNIX versions, FREESPACE() will use the partition on which the main file's map is located.

Version Ref: 4.8

Description:

FREESPACE() returns the free space (in bytes) of the drive letter [DOS] or filesystem [Unix] specified.

Examples:

Then: XX=FREE SPACE("C")

Returns the amount of free space left on the C drive.

FTP_OPEN

Version 6.0.00

handle = ftp_open(remote, username, password [,timeout])

Returns a handle to an FTP connection

remote: URL of the server to connect to, e.g. ftp://172.16.2.1/ (must end in a trailing slash)

username: The username to connect with.

password: The password for the connection.

timeout: Optional - sets the length in seconds before the connection is terminated on a pending connection. Default 0, infinite timeout.

example:

```
declare local handle(8,.0);
```

```
handle = ftp_open("ftp://172.16.2.1/","logan","mypassword","10")
```

```
status = ftp_chdir(handle, directory) Changes the current FTP directory for certain commands.
```

```
ftp_rename, ftp_mkdir, ftp_rmdir, ftp_delete, ftp_list, ftp_pwd
```

handle: Handle to the FTP connection.

directory: Directory to switch to.

Returns "0" or a negated libcurl error.

example:

```
status = ftp_chdir(handle, "directory")
```

```
status = ftp_put(handle, local, remote [, show progress])
```

Send a file to a FTP server.

handle: Handle to an FTP connection.local: Local file to send.

remote: URL to upload to (Must be URL, cannot use relative paths).

show progress: "0" hide progress bar, "1" show progress bar, default "1".

Returns "0" or a negated libcurl error.

example:

```
status = ftp_put(handle, "/send/file.txt","ftp://server/path/to/file.txt", "1")
```

```
status = ftp_get(handle, local, remote [, show progress])
```

Get a file from a FTP server.

handle: Handle to an FTP connection.

local: Local file to save to.

remote: URL to get from (Must be URL, cannot use relative paths).

show progress: "0" hide progress bar, "1" show progress bar, default "1".

Returns "0" or a negated libcurl error.

example:

```
status = ftp_get(handle, "/save/file.txt","ftp://server/path/to/file.txt", "1")
```

```
status = ftp_rename(handle, from, to)
```

Rename a file on a FTP server.

handle: Handle to an FTP connection.

from: File to rename.

to: New file name.

Returns "0" or a negated libcurl error.

example:

```
status = ftp_chdir(handle, "mydirectory")
```

```
status = ftp_rename(handle, "original.txt","renamed.txt")
```

```
status = ftp_mkdir(handle, directory)
```

Create a directory on a FTP server.

handle: Handle to an FTP connection.

directory: Path to create new directory, can be relative or absolute.

Returns "0" or a negated libcurl error.

example:

```
status = ftp_chdir(handle, "mydirectory")
```

```
status = ftp_mkdir(handle, "new directory")
```

```
status = ftp_rmdir(handle, directory)
```

Remove a directory on a FTP server.

handle: Handle to an FTP connection.

directory: Directory to remove.

Returns "0" or a negated libcurl error.

example:

```
status = ftp_chdir(handle, "mydirectory")
```

```
status = ftp_rmdir(handle, "todelete")
```

```
status = ftp_delete(handle, remote)
```

Delete a file on a FTP server.

handle: Handle to an FTP connection.

remote: File to delete.

Returns "0" or a negated libcurl error.

example:

```
status = ftp_chdir(handle, "mydirectory")
```

```
status = ftp_delete(handle, "todelete")
```

```
status = ftp_close(handle)
```

Closes an open FTP handle.

handle: Handle to an FTP connection.

Returns "0" or a negated libcurl error

error = ftp_error(code)

Translates a FTP error into a human readable error string.

code: The numeric error code returned by a FTP method.

Returns the associated error string.

```
num = ftp_list(handle, array)
```

Gets a listing of all files and directories in the current FTP directory.

handle: Handle to an FTP connection.

array: A non-aliased array to return the list of files through.

Returns the number of elements in the returned array, or a negated libcurl error.

example:

```
dim array(); ' 0 size arrays are now allowed by filePro
num=ftp_list(handle, array)
if: num lt "0" ' num should contain the number of elements in array
then: errorbox ftp_error(num); xx=ftp_close(handle); end
then: xx=selectbox(array)
```

path = ftp_pwd(handle)

Get the PWD of the FTP handle.

handle: Handle to an FTP connection.

Returns the PWD of the ftp handle.

GET16() GET32() PUT16() PUT32()

Gets or Writes data to a file.

Syntax

```
value = GET16( buffer [ ,offset [ ,byteorder ] ] )
value = GET32( buffer [ ,offset [ ,byteorder ] ] )
binval = PUT16( value [ ,byteorder ] )
binval = PUT32( value [ ,byteorder ] )
```

Version Ref: 5.6

The GETnn functions retrieve a binary value from a buffer, and the PUTnn functions convert the value into binary. The offset parameter specifies the zero-relative offset within the buffer where the value resides. The byteorder parameter specifies the byte order of the binary value, with "L" meaning little-endian, "B" meaning big-endian, and the default being the native order of the current system.

For example, to get the 32-bit little-endian value at offset 8 within the MyBuffer variable, you would use:

```
value = GET32(MyBuffer,"8","L")
```

To convert the RouterHandle variable into a 16-bit little-endian value, you would use:

```
value = PUT16(RouterHandle,"L")
```

Note that 8-bit values can already be read/written using the existing ASC and CHR functions.

These functions are used more easily interpret the contents, where READ()/WRITE() could access the data, but interpreting the contents would require a bit of work to extract/build the binary data.

For example, you want to get information about a .BMP file. Within the file is a BITMAPINFOHEADER structure, defined in C as:

```
typedef struct tagBITMAPINFOHEADER
{
    DWORD biSize;
    LONG biWidth;
    LONG biHeight;
    WORD biPlanes;
    WORD biBitCount;
    DWORD biCompression;
    DWORD biSizeImage;
    LONG biXPelsPerMeter;
    LONG biYPelsPerMeter;
    DWORD biClrUsed;
} BITMAPINFOHEADER;
```

After you read the information with:

```
xx = read(handle,MyBuffer,"40")
```

You want to extract the information from it:

```
biSize = GET32(MyBuffer)
biWidth = GET32(MyBuffer,"4")
biHeight = GET32(MyBuffer,"8")
biPlanes = GET16(MyBuffer,"12")
biBitCount = GET16(MyBuffer,"14")
... and so on ...
```

If you wanted to write out the information, you could build the buffer using the PUT16/PUT32 functions:

```
MyBuffer = PUT32(biSize) & PUT32(biWidth) & PUT32(biHeight) & PUT16(biPlanes) & PUT16(biBitCount) & ...etc...
```

It may also be the case that the file is always in a specific endian byte order, regardless of the platform you are running filePro on. (For example, binary data sent via TCP/IP is almost always sent in big-endian order, even on little-endian platforms.) In that case, you would pass an explicit "L" or "B" endian parameter to force the byte order.

GETCWD()

Syntax:

Then: `xx=GETCWD()`

Return value is a text field, which is the length of the current directory name.

Note: Under MS-DOS, the path entries are separated with forward slashes "/" rather than backslashes "\".

Version Ref: 4.5

Description:

Returns the current directory name.

Examples:

To write an ASCII file to the current directory:

Then: `xx=getcwd(); aa=xx & "/mailist"`

Then: `export ASCII test=(aa)`

GETENV()

Syntax:

Then: v=getenv("name")

Version Ref: 4.1

Description:

Returns value of environment variable "name", or variables stored in the filePro config file (fp/lib/config).

Examples:

Then: aa=getenv("LOGNAME")

Then: hm=getenv(aa)

The GETENV function returns the value of a variable stored in the fp\lib\config file or the system environment. However, when you start Inquire, Update, Add or Request Output, these values are copied to a temporary memory, where GETENV will look for them. Therefore, any changes made to these variables by other users while you are in Inquire, Update, Add will not be picked up by GETENV.

If you execute GETENV on the variables PFPROG, PFDIR or PFDATA, and the variable has not been given a value, GETENV will look for the value in the corresponding line in the fppath file.

IMPORTANT: An environment variable in the system environment will override the same variable in the configuration file.

GETENV now checks the fppath file for PFPROG, PFDATA, and PFDIR

GETLOCKS()

Version Ref: Version 6.2 (USP 6.1.02)

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
```

GETNEXT

Syntax

Each "getnext" lookup operation has five basic parts:

- a lookup statement
- a test for the end of the file
- field assignments and operations
- the getnext or getprev instruction
- a loop back to the end-of-file test

The syntax is like this:

Then: lookup filename . . .

subr

If: not filename

Then: [end or go elsewhere]

Then: n=filename(m); . . .

Then: getnext filename; goto subr

or:

Then: getprev filename; goto subr

Version Ref: 3.x

Description:

LOOKUP's "getnext" (get next) and "getprev" (get previous) commands make it easier to write repetitive lookups. They let you look through the lookup file repeatedly without retyping lookup statements or writing loops back to the lookup statement.

You might want to add a second test to check that the lookup key and lookup field still match - if you only want Smiths, for instance, you don't want the program to continue on to the Todds once the Smiths run out. The syntax would be something like this:

If: filename(m) ne [key field]

Then: end or go elsewhere

GETNONCE() (not included in filePro Lite)

After a call to ENCRYPT(), you can call GETNONCE() to get the "nonce" that was used.

Syntax

```
xx = GETNONCE ( )
```

"nonce" is the encryption mode used by filePro.

If "nonce" is not specified, filePro generates one.

Many consider it important to use a unique nonce with each piece of data encrypted. It does not reduce the cryptographic security (make the decryption any easier) of the encrypted data if the nonce is known unless the same nonce is reused with different data. An example of this might be encryption of data in medical records. You might use a nonce built from the patient ID and social security number. Then, a unique nonce could be generated (and regenerated) in processing from data which would not change stored in the record. If some of the nonce data might possibly change it wouldn't be to hard to build a routine to retrieve the encrypted field, decrypt it with the old data, re-encrypt it with the new data, and store it.

GETPID()

Returns the process ID of the current process.

Syntax

```
xx = GETPID()  
xx = GETPPID()
```

Return value

A text field, which is the current process ID.

Version Ref: 5.6

Note: GETPPID() returns the process ID of the parent process under*NIX. On Windows, this information is not available, and an empty string is returned.

GETPREV

Syntax

Each "getprev" lookup operation has five basic parts:

- > a lookup statement
- > a test for the end of the file
- > field assignments and operations
- > the getnext or getprev instruction
- > a loop back to the end-of-file test

The syntax is like this:

Then: lookup filename . . .

subr

If: not filename

Then: [end or go elsewhere]

Then: n=filename(m); . . .

Then: getnext filename; goto subr

or:

Then: getprev filename; goto subr

Version Ref: 3.x

Description:

LOOKUP's "getnext" (get next) and "getprev" (get previous) commands make it easier to write repetitive lookups. They let you look through the lookup file repeatedly without retyping lookup statements or writing loops back to the lookup statement.

You might want to add a second test to check that the lookup key and lookup field still match - if you only want Smiths, for instance, you don't want the program to continue on to the Todds once the Smiths run out. The syntax would be something like this:

If: filename(m) ne [key field]

Then: end or go elsewhere

GETQUAL()

Version Ref: Version 6.2 (USP 6.1.02)

s = GETQUAL(fname) - Return a colon delimited list of all qualifiers for the file given by "fname"
n = GETQUAL(array, fname) - Return 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: s=GETQUAL("invoices") ' s will contain "2022 :2023 :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

GOSUB

Syntax:

```
Then: GOSUB "label"  
If: ...  
Then: ...  
label If: ' subroutine  
Then: ...  
If: ...  
Then: ' subroutine  
Then: RETURN
```

Version Ref: 3.x

Description:

"label" is a subroutine of code, terminated with a RETURN, that GOSUB calls. RETURN ends the subroutine started at "label" by GOSUB, and starts processing up again at the command after GOSUB.

Processing will immediately branch to "label" and start executing from that point until a RETURN is encountered, at which time processing returns to the command immediately after the GOSUB. If there is a ";" after this GOSUB label, the command after the ";" is executed next. Otherwise, processing will pick up again at the "if" line immediately below "GOSUB label".

IMPORTANT: You must end all subroutines with a RETURN. Do not send processing to a subroutine with GOSUB and have it encounter an END statement before it encounters a RETURN. This may work in many cases but is not a good practice and can cause erratic results.

Note: Maximum number of nested GOSUBs (GOSUBs that are called by other GOSUBs) is 16 per table.

Command used with GOSUB:

GOSUB OF / GOTO OF

Syntax:

Then: GOSUB (num_expr) OF label1, label2, ...
Then: GOTO (num_expr) OF label1, label2, ...

"num_expr" is an index into the list of labels (starting at 1).
"label1,label2, ..." are the labels where processing branches to.

Only the integral portion of num_expr is used. Any fractional portion is ignored. The number is not rounded.

If "num_exper" is less than 1, or greater than the number of labels, the gosub/goto is not executed, and processing falls through to the next statement.

NOTE: The parentheses around (num_expr) are required.

Version Ref: 4.5

Description:

Gosub/goto one of a list of labels.

Examples:

Handle different types of records, assuming field "rectype" contains "A", "B", or "C".

(Note that "A" has an ASCII value of 65.)

Then: GOSUB (asc(rectype) - "64") OF RecTypA, RecTypB, RecTypC

GOTO

Syntax:

```
Then: GOTO "label"  
If: ... ' this code will be skipped  
Then: ... ' this code will be skipped  
label If: ' label; processing continues here  
Then: ...  
If: ...  
Then: END
```

Version Ref: 3.x

Description:

GOTO means "go to another processing element." GOTO must be followed by a label.

group(path/filename) (ver 5.8.03)

Group() returns a string containing "owner:group" on the file.

In Windows, there is no concept of group but the function still exists. It returns the gid and uid associated with the file (if it exists) otherwise, it returns "0:0"

see also [mode\(\)](#)

GUI

Syntax:

if: GUI

if: NOT GUI

Version Ref: 5.0

Description:

Allows you to test if you are running under a Graphical User Interface (GUI) environment or not.

GUI2WEB (Version 6.0.00)

Syntax:

if: GUI2

if: NOT GUI2

Syntax:

if: WEB

if: NOT WEB

Description:

Allows you to test if you are running under a fileProWEB User Interface environment or not.

Version 6.0.02

x=@GUI.PAUSE()

Pauses automatic screen updating while in Gl/Web.

x=@GUI.RESUME()

Resumes automatic screen updating while in Gl/Web.

These functions are useful for hiding the output of some system commands while running processing. Output can sometimes cause undesired effects on a filePro screen or prevent some commands from updating the screen correctly. This should correct some text being displayed while outside of filePro's messaging.

Example:

Then: x=@GUI.PAUSE()

Then: SYSTEM command

Then: x=@GUI.RESUME()

HARDCOPY

Syntax:

Then: hardcopy

Prints the current screen. Works in conjunction with PFHCFE (when set to ON this environment variable adds a form feed after the screen print.)

Version Ref: 3.x

Description:

HARDCOPY lets you print a hardcopy from a processing table. It prints the current record, functioning like "H - Hardcopy" on the Inquire, Update, Add option line.

Restrictions:

May only be used on Input processing tables.

Examples:

Then: input popup q(1,yn) "Do you want a hardcopy? (y/n) "

If: q eq "y"

Then: hardcopy

Note : When printing is directed to a spooler, the screen may not print immediately but will print after exiting "Inquire/Update/Add" mode. You may be able to override this with spooler options/controls provided by your operating system or by turning spooling "off" when there is a need for immediate printing. You can also use the "form" command (provided you have created a form with the desired fields) as an alternative to using the "hardcopy" command. See FORM for additional details.

HASH (Version 5.7.3)

```
result = HASH( hash_type, data [, result_type [, length ]] )
```

Where:

hash_type is one of the available hash methods:

```
"whirlpool"  
"sha512"  
"sha384"  
"sha256"  
"sha224"  
"sha1"  
"ripemd320"  
"ripemd256"  
"ripemd160"  
"ripemd128"  
"tiger"  
"md5"  
"md4"  
"md2"
```

Case is not significant in the name.

Data is the actual data to hash

Result_type specified how the hash should be returned.

0 - (default) Return the hash as-is

1 - Return the hash in hex

2 - Return the hash in BASE64 encoding

Length is the length of the input data. The default is to use the entire length of 'data'

Example:

```
hash("sha1","Hello","1") returns
```

```
"F7FF9EBB7BB2ED9B70935A5D785E0CC5D9D0ABF0"
```

```
hash("sha1","Hello","2") returns
```

```
"9/+ei3uy4Jtwk1pdeF4MxdnQq/A="
```

New "hash-based message authentication code" (HMAC) hash function added to Version 5.7.04:

```
result = HMAC_HASH( method, data, key [, method2 [, length]] )
```

where "method", "data", "method2", and "length" correspond to the same parameters in HASH(), and "key" is the HMAC key.

HELP

Syntax:

Then: HELP name

Display a help screen of section "name".

"name" is found in a file called "help" (Unix) in the current filePro file's directory.

"name" is found in a file called "HELP" (DOS) in the current filePro file's directory. This is a plain text file which holds help for screens, fields, and "names".

Version Ref: 3.x

Description:

Displays a help screen.

Examples:

Say that you want a certain help screen to appear automatically whenever the user moves the cursor into the tax rate field (field 8) and tries to type whole numbers instead of a decimal number. First you'd write a piece of text in the help file named "taxes." Then, you'd write the following conditional processing statement:

```
@mf8 If: mid(8,"1","1") ne "."  
Then: help "taxes ; screen 1,8
```

As soon as the user moves out of field 8, the program checks for the period, puts up the help screen if it isn't there, and returns her to field 8 in update mode when she is done looking through the help pages. (HELP screens can be as many pages long as required).

HTML (not included in filePro Lite)

[See HTML Reference.](#)

HTMLERRNO()

Syntax:

Then: xx=HTMLERRNO()

Version Ref: 4.8 (not included in filePro Lite)

Description:

Returns an error code for the last HTML/JSFILE/JSON statement. Zero means the statement succeeded. "1" indicates that the specified document id is not an open document. A negative number "-1", "-2", etc. is the system error number.

IMPORT (not included in filePro Lite)

Syntax:

Then: import ASCII filename R=f F=f O=o C=c

Then: import DIF filename

Then: import WORD filename

DIF, WORD, ASCII are the type of file from which IMPORT will draw data.

filename is the source filename.

filename can be an expression by assigning it as an alias as in:

Then: import ASCII aaa=(exp) R=f F=f O=o C=c

Version Ref: 3.x

Description:

IMPORT opens a non-filePro file so that its data can be accessed by filePro. Typically, records from an imported file are copied into records of a filePro file. The data can be copied (on a field by field basis) into existing records or into free records.

IMPORT can work with three types of file formats: ASCII for reading text and word processing files, DIF for reading "dif" style files, and WORD for reading WordStar text files.

ASCII:

Then: import ASCII name R=f F=f O=o C=c

or:

Then: import ASCII name=pathname R=f F=f O=o C=c

where "name" is the name of the import file and "pathname" is a full or relative pathname. Use the pathname option when the merge file is not in the current directory.

"r" is record separator

"f" is field separator

"o" is opening field delimiter

"c" is closing field delimiter.

Example:

Then: IMPORT ASCII test=test.txt R=^J f=^

The above would import a file named "test.txt" in your current directory containing a "linefeed" character at the end of each record with a "^" between each data field. Notice the different use of "^" symbol. In the first case it is used as a control character to read a "linefeed" and in the second case it reads a literal "^" since it is preceeded with a "\ " backslash. Also notice that the "O" Opening and "C" Closing field delimiters are not used in the above example.

IMPORTING CSV formats - Opening and closing field delimiters are commonly used in "CSV" type files.

Example:

Typical CSV formatted file

"John","Adams","123 South Street","Anywhere","NY"

"Quincy","Smith",100 North Street", "Washington","PA"

"Debby", "Adams", "123 State Hwy 7", "Mount Vernon", "VA"

This data format is usually imported using the IMPORT WORD but can also be imported using IMPORT ASCII by identifying quotes as the opening and closing field delimiters as follows.

Then IMPORT ASCII tst=c:\temp\test.txt R="^J f=" O=" C="

IMPORTANT: You may use a few special codes with the IMPORT ASCII format to identify delimiters or field separators:

\r for carriage return

\f for form feed

\n for new line

\t for tab

You may also use punctuation characters, ASCII codes, and control codes if your import file contains these as delimiters and separators.

The CARET " ^ " symbol - This symbol requires special treatment since it indicates a "CTRL" character and is normally followed by a letter e.g. ^J, ^L, ^M, etc. Refer to the "Character Table" in "Define Processing" help for "CTRL" characters. In cases where you want to import the caret "^" symbol as a literal, precede it with a backslash e.g. \^.

DIF:

Then: import DIF name

or:

Then: import DIF name=pathname

where "name" is the name of the import file and "pathname" is a full or relative pathname. Use the pathname option when the merge file is not in the current directory.

WORD:

Then: import WORD name

or:

Then: import WORD name=pathname

where "name" is the name of the import file and "pathname" is a full or relative pathname. Use the pathname option when the merge file is not in the current directory.

Each "IMPORT filename" definition must be alone on its "Then" line.

Field assignments are written after the IMPORT command has been executed.

Use at least three characters for the IMPORT filename, and don't start with a number.

IMPORT is not available on Automatic or Input processing.

When using a pathname for the IMPORT filename, use the separator mark (\ or /) appropriate to your operating system.

Restrictions:

An "IMPORT filename" definition must be alone on its "Then" line.

Only one EXPORT or IMPORT statement with the same filename is allowed per processing table. In other words, you can't have the statement, "EXPORT MULTI List" followed in a few elements by "IMPORT WORD List."

INDEXOF()

Version Ref: 6.1 (USP6.1.01)

Syntax:

```
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"
```

INKEY

Syntax:

```
If: inkey = "c"  
Then: aa=inkey
```

Where "c" is a keyboard character or nothing (no key pressed) and where "aa" is the dummy field used to hold the value of the key the user pressed.

Version Ref: 3.x

Description:

Gets next keystroke if available.

INKEY is a system function that returns the value of the first character in the type-ahead buffer. Until a key has been pressed, the value of INKEY is null. On action lines INKEY must be assigned to a field in order to use the key pressed. On condition lines INKEY must be compared to a value; it cannot be used by itself. The condition, "If: inkey", won't work.

For another way of capturing user keystrokes for processing, see the WAITKEY command.

INKEY cannot detect and does not respond to the BREAK key.

IMPORTANT: The INKEY function is CPU intensive. It is **STRONGLY** suggested that you use the WAITKEY function in its place.

Examples:

Use INKEY to eliminate extra keys the user may have pressed on the way out of Inquire, Update, Add or Request Output. These extra keystrokes are sometimes accepted by menus, and send the user into unwanted situations. Use this syntax to eliminate extra keystrokes:

```
loop If: inkey ne ""  
Then: goto loop
```

In other words, keep resetting INKEY until it has no value (is null ""). When INKEY becomes null, the loop ends.

To continuously show the time at the top-right corner of the screen until the user presses a key:

```
loop If: inkey eq ""  
Then: show(1,70) @TM; goto loop
```

Use @SK with INKEY to test whether the user wants help:

```
If: inkey eq "?" or @sk eq "HELP"  
Then: help "field1"
```

INPUT

Syntax:

```
INPUT n "message"  
INPUT(r,c) n "message"
```

Displays message on line 23, waits for user to respond and puts response into dummy variable n. (The response holder MUST be a dummy variable.)
The "message" is an expression. Any part of it that is a literal string must be inside quotes.

Version Ref: 3.x

Description:

Prompts for user input.

```
INPUTPW(r,c) n "message"
```

Standard input statement.

```
Then: input b "message to user"
```

You can build a literal string, or use any field or expression as the message portion of an INPUT statement.

```
Then: input b "message" < n & m
```

or:

```
Then: mm="Does everything look okay? (y/n)"  
Then: input q mm
```

You may define the variable which holds the response at the same time an INPUT question is asked. (Like any other filePro variable, this dummy field should only be defined once (anywhere) on a table. Subsequent or previous uses of a defined variable should not be defined again.

```
Then: input q(1,yesno) "Do you want to continue?"
```

INPUT questions can be placed at a particular point on the screen:

```
Then: input(r,c) b "message"
```

where "r" is the row and "c" is the column. The row and column entries are expressions.

You must include a message in the INPUT statement. For a blank message, use:

```
Then: input n ""
```

Note: The response can only be captured in dummy fields. Transfer the received values to real fields if they are to be stored on the record.

INPUT POPUP

Syntax:

Then: INPUT POPUP (r,c) aa "message"

Then: INPUT POPUP(r,c) n "message" DEFAULT [value]

Version Ref: 4.1

5.0 (enhanced)

Description:

Functions the same as INPUT, but displays the input request message in a popup box at row "r" and column "c". If these coordinates are omitted, the popup box is centered on the screen.

If the row and column value specified position any part of the window beyond the limits of the screen, the window is automatically re-positioned to appear completely within the screen boundaries.

The location of the upper left corner of the popup window is determined by the row "r" and column "c" values. If row and column are omitted, the window is centered on the screen. The popup window can also be centered by using "-1" for the row or column. This is useful if you want to center the window vertically or horizontally but want to specify a row or column.

The width of the popup window is determined by the defined length of the input variable plus the length of any message text on the same line.

The height of the window is determined by the number of message lines.

The code "\\N" can be used to begin a new line of text within the popup window.

Examples:

lf: 'display the input popup box at row 5 and column 7

Then: input popup ("5","7") aa(8,mdy/) "A2-SYSTEMARCHIVE PROCEDURE\n-----\n\nEnter Archive Date: "

lf: 'center the input popup box horizontally and vertically

Then: input popup aa(8,mdy/) "A2-SYSTEMARCHIVE PROCEDURE\n-----\n\nEnter Archive Date: "

lf: 'center the input popup box vertically starting in column 1

Then: input popup ("-1","1") aa(8,mdy/) "A2-SYSTEMARCHIVE PROCEDURE\n-----\n\nEnter Archive Date: "

lf: 'center the input popup box horizontally in row 1

Then: input popup ("1","-1") aa(8,mdy/) "A2-SYSTEMARCHIVE PROCEDURE\n-----\n\nEnter Archive Date: "

5.0 INPUT POPUP(r,c) n "message" DEFAULT [value]

Defaults the value of the input. You can optionally specify the value, or use the current value of the input field.

Example:

INPUT POPUP State "Enter state: " DEFAULT "NY"

Asks for the state, with "NY" defaulted in the input field.

Note: Make sure that "n" is defined e.g. declare State(2,state)

INPUTPW

Syntax:

Then: INPUTPW(r,c) n "message"

Version Ref: 4.0

Description:

Same as INPUT, but a '#' is shown for each character entered by the user to hide the user input. It is often desirable to mask responses when prompting for passwords and other security sensitive data.

INPUTPW POPUP

Syntax:

Then: `inputpwpopup(r,c) pw "message"`

Version Ref: 4.1

Description:

Same as INPUT POPUP, but a '#' is shown for each character entered by the user (for security purposes).

Examples:

Then: `inputpwpopup ("9", "8") pw(8,*) "Enter Password "`

INSTR()

Syntax:

Then: a = INSTR(n,m,s)

Looks for field "m" in field "n" starting at position "s". The starting location in field "n" is returned as a number in field "a"; return value is "0" if field "m" is not found in field "n";

"a" is the return value

"n," "m," and "s" are expressions; "n" is the field you are searching

"m" is the field you are looking for

"s" (optional) is the character position in field "n" from which to start the search.

If the field is not found, INSTR returns a zero.

Examples:

Then: a = "The quick, brown fox jumped over the lazy dog."

Then: b = "fox"

Then: x = INSTR(a,b) ' x will contain 18.

Then: x = INSTR(a," ") ' x will contain 4.

Then: x = INSTR(a," ",20) ' x will contain 21.

Version Ref: 4.1

Description:

INSTR looks for the contents of one field within a second field, beginning at a specified character position. INSTR returns the character position in the second field where the first field is found. INSTR can be used in an expression.

Version 6.0.01

RINSTR, and INSTR now allows negative positions for working backwards.

RINSTR(n,m)

RINSTR(n,m,s)

Search for m in n.

Searches backwards from string length or position s.

INSTR(n,m,s)

"s" (optional) is the character position in field "n" from which to start the search.

NEW: If you use "-1" as the position it will scan backwards from the end of the string and return the position if m is found.

INT()

Syntax:

Then: $a = \text{INT}(n)$

Version Ref: 4.1

Description:

Returns the integer portion of a number. (Compliment of the FRAC function.)

Examples:

Then: $in = \text{INT}(no)$

ISFILE

v6.2 (6.1.02 USP)

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 systemerror on failure.

ISDIR

v6.2 (6.1.02 USP)

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 systemerror on failure.

ISLINK

v6.2 (6.1.02 USP)

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 systemerror 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.

ISFILE

v6.2 (6.1.02 USP)

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 systemerror on failure.

ISDIR

v6.2 (6.1.02 USP)

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 systemerror on failure.

ISLINK

v6.2 (6.1.02 USP)

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 systemerror 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.

ISFILE

v6.2 (6.1.02 USP)

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 systemerror on failure.

ISDIR

v6.2 (6.1.02 USP)

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 systemerror on failure.

ISLINK

v6.2 (6.1.02 USP)

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 systemerror 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.

ISLEAP()

Syntax:

Then: xx=ISLEAP(date)

Then: xx=ISLEAP(year)

"date" is a date within the year to test.

"year" is a numeric field containing the year itself.

Return value is 1 if it is a leap year, 0 otherwise.

Version Ref: 4.5

Description:

Determines if a given year is a leap year.

Examples:

Using a date field, determine if the date is in a leap year.

```
Then: input popup dt(10,mdyy) "Enter a date: "
```

```
  If: isleap(dt) gt "0"
```

```
Then: msgbox dt < "is during a leap year."
```

Using a number, determine if that year is a leap year.

```
Then: input popup yr(4,.0) "Enter a year: "
```

```
  If: isleap(yr) gt "0"
```

```
Then: msgbox yr < "is a leap year."
```

Note: A leap year is defined as any year divisible by 4, unless it ends in a "00", in which case it must also be divisible by 400. The year 2000 is a leap year.

IXCOMMENT()

Syntax:

Then: xx=IXCOMMENT(filename,indexletter)

Return Value

Returns the comment from the specified index.

If the index doesn't exist, or has no comment, a null value is returned.

Version Ref: 4.8

Description:

IXCOMMENT() returns the comment for the index specified.

IXSORT()

Syntax:

Then: XX=IXSORT(filename,indexletter[,sortlevel])

Return Value

In the format: field,length,ascend/decend,[subtotal])

If no sort level is specified, all sort levels are returned in the format for all possible choices: 2, 12,a,n:1, 15,a,n:::

Version Ref: 4.8

Description:

IXSORT() returns the sort information for the specified index.

Version Ref: 5.8.03

Enhanced to allow adding working path the filename

Then: IXSORT(filename, indexletter[,sortlevel[,path]])

JSFILE

Syntax:

JSFILE :CR filename

JSFILE [id] :CR filename

where id is an optional file handle, :CR CReates the file.

(NOTE: If [id] is not provided, it defaults to "0")

JSFILE [id] :TX text

Writes text line(s) containing text to the file specified by id.

JSFILE [id] :CR-

Closes the file specified by filename.

The [id] file handle is not required for a single file. To create multiple files, use a unique file handle for each [id].

Version Ref: 4.8 (not included in filePro Lite)

Description:

JSFILE Creates sequential ASCII files.

Example: Creates file1.txt and file2.txt that says hello and filename.

If:

Then: fa="file1.txt"; fb="file2.txt"

If:

Then: ta="hello"

If:

Then: JSFILE "1" :CR fa

If:

Then: JSFILE "2" :CR fb

If:

Then: JSFILE "1" :TX ta < fa; JSFILE "2" :TX "Hello" < fb

If:

Then: JSFILE "1" :CR- ; JSFILE "2" :CR-

IXSEL() - (Ver. 5.8.03)

This command will return the selection set for the automatic index.

The format is:

```
XX=IXSEL (filename,indexletter)
```

The return value is:

```
SelectorSentence:Group,Heading,Rel,Value:Group,Heading,Rel, Value...
```

Version Ref: 5.8.03

Enhanced to allow adding working path the filename

```
Then: IXSEL(filename, indexletter[,sortlevel[,path]])
```

LEN()

Syntax:

Then: LEN(f)

Where "f" is a dummy or real field.

Version Ref: 3.x

Description:

LEN returns the length of the field if one is assigned. In the case of a dummy field, if no length is assigned, LEN will return the length of its contents. LEN can be used on all processing tables, on both the condition and action lines.

Examples:

The following assigns dummy field aa to have a length of field 5 and the "*" edit type.

Then: aa(len(5),*)

Fill the dummy field bb with the length of field 5.

Then: bb=len(5)

If the length of field 5 is 20, the contents of dummy field bb will also be 20.

LISTBOX()

Syntax:

```
If: n=# of elements in box, w=width of lines in box
Then: dim name[n](w) as in dim name[4](12)
If:
Then: name["1"]="Line 1 of box" ; name["2"]="Line 2 of box"
Then: name["3"]="Line 3 of box"; ... ; name[n]="Line n of box"
Then: a=LISTBOX(name,first,last,row,col,height,width)
```

```
Then: aa=LISTBOX(array,first,last,row,col,height,width [,position])
```

Version Ref: 4.1

5.0 (Enhanced)

Description:

Displays a popup window containing a list of values from "name", an array defined with DIM.

Only "name" is required, all other parameters are optional. "first" and "last" refer to the range within the array to use (defaults to first and last elements of the array).

All parameters up to the last parameter used must be specified. Any unused parameters may be specified as a NULL string (" "). Examples of valid parameter lists follow:

```
Then: aa=LISTBOX(array,"", "", "", "", height,width)
Then: aa=LISTBOX(array,"", "", "", "", width)
Then: aa=LISTBOX(array,"", "", "", "", height,"")
```

Unused trailing parameters can be omitted. For example, the following two statements are equivalent:

```
Then: aa=LISTBOX(array,"", "", row,col)
Then: aa=LISTBOX(array,"", "", row,col,"", "")
```

Note Version 5.0 enhanced "LISTBOX" to take an optional 8th parameter, which is the position of the initial highlight.

If row/col/height/width are not given, the window is centered on the screen. If the window is larger than needed, it will be shrunk accordingly.

The user sees the listbox (as prescribed) on the screen and control is handed to the user. Moving the highlighted bar to the desired choice and pressing ENTER deposits the number of the listbox item in the specified variable to the left of the equal sign, and puts control back on the processing table. The listbox will be cleared automatically from the screen. Appropriate action can be taken based on the number chosen by the user.

Choices from the listbox are made by positioning the highlight bar to the desired element with the arrowkeys, or by pressing the first character of that element, then pressing <RETURN>. If the desired element is named "test!", pressing "t" selects it. Select the next element (named "test2") by pressing "t" again.

The <PgUp> and <PgDn> keys may be used to move between screen pages. Use <HOME> to move the highlight to the first element on each screen page.

Description:

Prompts the user to select a value from a list.

Examples:

```
Then: dim months[12](3)
Then: months["1"]="Jan";months["2"]="Feb";...months["12"]="Dec"
Then: a=LISTBOX(months,"", "", "", "", "3")
Then: show "You chose month#"<a<"which is"<months[a]
```

@keyH If:

```
Then: dim hbox[3](41)
Then: hbox["1"]=1) Print the full schedule for this date."
Then: hbox["2"]=2) Print the schedule for a specific Dr."
Then: hbox["3"]=3) Cancel hardcopy request. "
Then: sg=listbox(hbox)
If: @sk="SAVE"
Then: goto @keyH
If: @sk="BRKY"
Then: end
If: sg="1"
Then: goto doform1
If: sg="2"
Then: goto doform2
If: sg="3"
Then: end
```

doform1 If: *doform1

```
Then: beep
Then: msgbox "\n\rThis schedule is\r\n\rnowprinting. \r\n\r\n"
Then: sn="report pcdappdet -f alldoctors -v vall -id -a -u -bg -bs 2>/d
ev/null 1>/dev/null -r"<p&d
If: 'the /dev/null redirections to /dev/null are for Unix only
Then: system noredraw sn
Then: end
```

doform2 If: *doform2

```
Then: input popup sl "Enter Doctor \r1-8\r\n=> "
If: sl=""
Then: end
If: sl lt "1" or sl gt "8"
Then: goto doform2
Then: sm=doctnum[sl]
Then: sn="report pcdappdet -f singlecoct -v vsingle -ia -u -bg -bs 2
>/dev/null 1>/dev/null -r"<sm&p&d
If: 'the /dev/null redirections above are for Unix only
Then: beep
```


Lite or filePro Lite

filePro Lite is an abbreviated version of filePro that has enough features to create small basic programs. Advanced features are not available in Lite and certain functions may be limited.

LOCKED()

Syntax:

If: LOCKED(lookupname)

Version Ref: 4.8

Description:

Test if the specified lookup failed because the record was locked. (requires the new "-w" flag on lookup in addition to the "-p" flag)

LOG()

Syntax:

Then: result = LOG(n)

Version Ref: 4,8

Description:

Natural (base e) logarithm

Examples:

When na = "100"

XX=log(na)

Returns the natural log value "4.6051702"

LOG10()

Syntax:

Then: result = LOG10(n)

Version Ref: 4,8

Description:

Base 10 logarithm

Examples:

When na = "100"

XX=log10(na)

Returns the common log value "2.000000"

LOGTEXT

Syntax:

Then: LOGTEXT exp

Adds "exp" to file stored in environment variable LOGFILE.

Notes: Will do nothing if the LOGFILE variable is not set.

LOGAPPEND controls the "open" mode of the file for the session. The choices are overwrite or append for each session.

Version Ref: 4.5

Description:

Stores data in the destination file set by the environment variable LOGFILE.

Examples:

Then: LOGTEXT "Record "<@m< " updated by "<@ID< "on" <@TD<"at"<@TM<"."

Result: A line of text is shown in the destination file similar to the following

```
"* Record 3 updated by root on 08/07/99 at 12:30:22"
```

See Also

LOGFILE

PFLOGAPPEND

Creating a Lookup

There are three steps (or parts) to consider when creating any lookup:

1. Designating which file to use for the lookup.
2. Designating how to find the appropriate record in that file.
3. Designating what to do if the lookup fails, i.e., is not successful in finding the specified record.

A fourth consideration comes into play after the lookup has been run successfully. This step is not actually part of the lookup statement itself, but comes on processing lines after the LOOKUP line.

4. Designating the fields in the lookup file from which to take data, or those in which to put data, or both.

IMPORTANT: The lookup wizard can be used to prompt you step-by-step for the required LOOKUP parameters. This wizard is accessed by pressing the F5 key while the cursor is on a "then" line. (This key is for PCs and ANSI mode terminals. It will be different for character based terminals, on which it is normally CTRL-R.

IMPORTANT: If you use punctuation in filenames, and you want to do a lookup to such a file, you need to use an alias

Lookup Statements

Record-number, key-field and free-record lookups have different formats and a variety of different options.

For help defining the lookup file statement, use the "define lookup" option accessed with <F5 >.

For an assigned file name:

```
Then: lookup assign=filename r=f -e
```

Record-number options

```
Then: lookup filename r=f -e
```

where "filename" is the name of the file containing the desired records, "r" indicates record-number lookup, "f" is the field containing the record number in the current file (the value in field "f," is a record number from the other file), and "-e" is the default lookup failure choice, "report an error. "

Free-record options

```
Then: lookup filename r=free -e
```

Keep in mind that the lookup file doesn't have to contain empty records. LOOKUP expands the file as needed.

On UNIX/XENIX systems, you might want to add a "protect" flag (-P) to the statement. The -P flag prevents two users from accessing and changing the same lookup record at the same time. It does so by locking the lookup record as soon as it's accessed; the record isn't unlocked until all processing is done and the new data is saved or until a WRITE statement is done on that lookup.

Syntax:

```
Then: lookup filename r=free -ep
```

Keyfield options

Syntax:

```
Then: lookup filename k=m i=o -xe
```

where "filename" is the name of the lookup file, "k" is the keyfield flag and "m" is the reference field in the current file; "i" is the index flag and "o" can be index A-H, 0-9 in the lookup file; -x is the default match, "exact match," and e is the default lookup-failure choice, "report an error."

Indexes

Keep in mind that the index must be built in the lookup file on the field corresponding to the cross-reference field in the current file. For example, if the key field in the current file is zip code, sort the lookup file's index on its zip code field. For best results, make sure the index sort-key and the cross-reference field have the same edit type.

Match options

- X, exact match
- G, exact or next greater match
- L, exact or next lower match

If you pick the 'less than' or 'greater than' mode, the program looks for an exact match first. If it finds no match, it chooses the record with the value just less or greater than the one requested.

Lookup-failure options

The failure option flags tell the program what to do if the right information isn't found. The options are:

- B, fill the field with blanks
- N, do nothing (use for processing tests)
- E, report an error

The E flag is the only one of the three that returns a "Lookup Failed" error message. The N flag is used to test for lookup failures and resolve them while processing is running.

Note:

If you use the N flag without testing for failed lookups (by adding the statement "If: not filename"), you'll get another error message later, when you try to use the field for which the lookup failed. The B flag returns no error message, just fills the field with blanks. See "Connecting Files: Lookups and Posting" earlier in this chapter for more information.

Changing record positions

Use LOOKUP and a hyphen instead of a file name to change the record position in the current file. (Note that this is not the same as doing a lookup from and to the current file.) This option lets you keep the user in update mode while still letting him pick the record to update next. See example 2 below.

Syntax: Writing Field Assignments

Once the lookup files are specified, you must tell the program into which fields, exactly, the data are to be put. Specifying fields for lookups:

```
Then: a=filename(n); b=filename(m); . . .
```

where "a" and "b" are the fields in the current file into which the data go, "filename" is the name of the lookup file, and "n," "m," etc., are the numbers of real or system-maintained fields containing the data in the lookup file.

Specifying fields for posting:

```
Then: filename (n) =a; filename (m) =b; . . .
```

where "filename" is the name of the lookup file, "n," "m," etc. are the numbers of real fields into which the data go, and "a" and "b" are dummy, system-maintained or real fields in the current file from which the data come.

Example 1

In the COPY and INPUT examples, your records were simply appended to the end of Archive. With LOOKUP, however, you can archive to particular records. In this set of instructions, you archive records according to customer name.

Step 1

Access the "arch" output processing table in sample file, TInvoice. Change the record-number lookup in element 3 (or 1 if you didn't do the INPUT example) to a key-field lookup, as follows:

```
lookup archive k=1 i=b -ex
```

Save the processing table. If you tried the INPUT example, this is how the revised table should look (screen 13-23):

Screen (x): Copying to Particular Records

Step 2

Build Archive index B on field 1, customer. Also, create a few new invoices using the same customer numbers but different item numbers or quantities as in the invoices already archived. Note the balances due.

Step 3

Run the archival; access Request Output; the file name is tinvoice, the output format is "arch." Sort on field 1, if desired, and select all records. Enter an archive date.

Step 4

Check Archive via Inquire, Update, Add. There should be no additional records, but the balances-due should be different.

Example 2

Suppose that you want to let the user update a series of records without having to exit from update mode each time he wants a particular record. You could write the processing as follows (try it on TClient):

```
Then: cc=@rn; input aa(15,uplow) "What name do you want to
      update now? Press RETURN to quit"
Then: bb=aa { ""
If: aa eq ""
Then: exit
Then: lookup - k=bb i=a -ex
If: cc eq @rn
Then: getnext -
Then: screen 1; restart
```

This set of elements asks the user what name to find next, switches to the new record (if he asks for the same name, moving the user to the next instance of the name - cc eq @RN, getnext -). Once the record is found, the SCREEN command puts him in update mode and RESTART starts processing from the top of the table.

Restrictions

A lookup statement must be alone on its action line.

Any file name (including an assigned file name) used in a lookup statement must be at least three characters long and should start with a letter.

IMPORTANT: If you use punctuation in filenames, and you want to do a lookup to such a file, you need to use an alias

Example 3

```
If: lookup cust r=free -e
Then: cust(4)=3
```

Copy field 3 of the current (or main) file record into field 4 of the free record in "cust".

Example 4

```
Then: aa="1"
Then: lookup control r=aa -e
Then: zz=control(8)
```

Looks up file "control" using the value in field aa to get a particular record number in the "control" file. Copies the value in field 8 on that record in the "control" file to the dummy variable zz.

NOTE: As of Version 5.7.04, browse lookups are now supported in dreport/rreport

MAX()

Syntax:

Then: a=MAX(n)

Then: n=MAX(aa,ab, ...)

Version Ref: 4.1

Description:

Finds maximum value of field "n" on output, or returns the highest value in the list of fields given.

Examples:

For financial analysis, you want the average sales per sales representative, plus, for comparison, his or her maximum and minimum sales (see the "AVG" and "MIN" commands also). "Total Sales" is field 22. To find the maximum value, you'd put dummy field M on a subtotal line in the report format, and write the element as follows:

Then: M=MAX(22)

Find the maximum in a list of values.

Then: mx = MAX(ab,ac,ad,ae,5,12)

Restrictions:

For aggregate maximums, calculations are made at the subtotal and total breaks.

For comparative maximums, the fields do not have to be numeric, but they must all be comparable. For example, you may supply a list of dates or times, but not both. There must be at least two fields. If only one field is supplied, the aggregate form of the function is used.

FOR

Version Ref: 6.1 (USP6.1.01)

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

- 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.

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.

Example - For Loop

Processing:

```
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
```

WHILE

Version Ref: 6.1 (USP6.1.01)

WHILE cnd DO label

- 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.

Example - While Loop

Processing:

```
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|LOOP UNTIL

Version Ref: 6.1 (USP6.1.01)

LOOP label WHILE cnd

LOOP label UNTIL cnd

- 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.

Example - Loop While

Processing:

```
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
```


FOR

Version Ref: 6.1 (USP6.1.01)

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

- 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.

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.

Example - For Loop

Processing:

```
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
```

WHILE

Version Ref: 6.1 (USP6.1.01)

WHILE cnd DO label

- 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.

Example - While Loop

Processing:

```
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|LOOP UNTIL

Version Ref: 6.1 (USP6.1.01)

LOOP label WHILE cnd

LOOP label UNTIL cnd

- 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.

Example - Loop While

Processing:

```
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
```

FOR

Version Ref: 6.1 (USP6.1.01)

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

- 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.

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.

Example - For Loop

Processing:

```
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
```

WHILE

Version Ref: 6.1 (USP6.1.01)

WHILE cnd DO label

- 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.

Example - While Loop

Processing:

```
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|LOOP UNTIL

Version Ref: 6.1 (USP6.1.01)

LOOP label WHILE cnd

LOOP label UNTIL cnd

- 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.

Example - Loop While

Processing:

```
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
```

MDAY()

Syntax:

Then: `xx=MDAY(date)`

where "date" is a date within the month to test.

Return value is a numeric field containing the number of days in the specified month.

Version Ref: 4.5

Description:

Returns the number of days in a given month.

Examples:

An old rhyme

Then: `msgbox mday(doedit("090195","mdy")) < "days hath September."`

MEMO

Syntax:

Then: MEMO xx delete

Then: MEMO xx clear

Then: MEMO xx edit [(row, col, height, width [,startrow, startcol])] [WRAP | NOWRAP] [READONLY] [TITLE]

Then: MEMO xx export filename [APPEND]

Then: MEMO xx import filename

Then: MEMO xx show [(row, col, height, width [, startrow, startcol])] [WRAP | NOWRAP]

ff: MEMO

ff: NOT MEMO

Version Ref: 5.0

Version Ref: 6.0.00

Added maxsize to limit the number of allowable characters entered into a memo window

memo NNN edit (row,col,lines,width,startLine,startcol,maxSize)

Description:

Used to manage Binary Large Objects (BLOBS) which are plain-text. The items are stored within a filePro record as a variable length field based on the size of the object. The objects are retrieved with an internal filePro editor or by using an external program to display, edit, print, or to otherwise manipulate them.

Important Note: If you use MEMO SHOW you must have a corresponding MEMO CLEAR before updating or moving to a new record. Without a CLEAR you will leave old memo data on the screen and it will under/overlay with data from the new record or auto processing.

Note:

Length and type must be defined as (16,MEMO).

Memo Printing

Refer to "[Embedding Objects](#)" for memo printing.

MEMO commands:

Then: MEMO xx IMPORT filename

Imports contents of specified "filename" into MEMO field xx.

Then: MEMO xx EXPORT filename [APPEND]

Exports MEMO field xx to the specified filename.

APPEND is an optional parameter that allows you to append the exported memo to a file. This option was added in version 5.6

Then: MEMO xx DELETE

Deletes the specified MEMO field xx.

Then: MEMO xx SHOW [NOWRAP]

Shows the specified MEMO field xx.

NOWRAP is an optional parameter added in version 5.6 to prevent text from wrapping.

Then: MEMO xx CLEAR

Clears the specified MEMO field xx after a SHOW xx MEMO.

Then: MEMO field EDIT [(row, col, height, width [,startrow, startcol])] [WRAP|NOWRAP] [READONLY]

Brings up the internal filePro memo editor. All parameters are optional. Blank row/col means center vertical/horizontal. Blank startrow/startcol

means position at start of memo. The internal filePro editor will word wrap as the default. Word wrap can be turned with the optional NOWRAP parameter. READONLY will prevent the memo from being updated.

Then: MEMO field EDIT (row,col) TITLE "Window title text"

Then: MEMO field SHOW (row,col) TITLE "Window title text"

Adds a title to the memo window.

Note: Only fileProGI will display this title in version 5.0.6.

Examples:

@keym lf:

Then: MEMO 2 EDIT

The above example will bring up the internal filePro memo editor for field 2 in a centered window when the key "m" is pressed at "Enter Selection". Since no parameters are specified, a default window size of 10 rows by 35 columns wide is used.

@wlfmf if: mf gt "" 'a memo flag field is greater than a blank

then: MEMO 2 EDIT (" ", "15", "60", "2 ", "1")

In the above example, if dummy field "mf" is not equal to a blank you will get a memo editor window that is centered. The height is "15" rows and width is "60" columns. When the user enters the memo field, they will be placed in row 2, column 1 as specified by the last two parameters.

lf: @sn eq "ray" 'if the current screen name is ray

Then: MEMO 10 EDIT ("10", "10", "10", "40") NOWRAP

If you are on screen name "ray", the filePro memo editor opens up a window with an upper left-hand corner at 10,10 and places the current contents of memo (field 10) in it for editing. The window size is 10 rows by 40 columns. Notice that the starting row and column values are not specified in this example. If these values are not specified the starting row and column defaults to "row 1", "column 1". Also, word wrap is turned off with the optional parameter "NOWRAP". If this parameter is not used, the internal memo editor defaults to word wrapping e.g. "WRAP".

lf: @sn eq "ray"

Then: MEMO 10 SHOW

If you are on screen name "ray", the filePro memo editor opens up a window with an upper left hand corner at 10,10 and places the current contents of memo (field 10) in it for display only. The window size is 10 rows by 40 columns. Notice that the starting row and column values are not specified in this example. If these values are not specified the starting row and column defaults to "row 1", "column 1".

lf:

Then: MEMO nn CLEAR

Clears the memo field from the screen after a MEMO nn SHOW.

MEMO Condition

lf: MEMO

lf: NOT MEMO

Determines if the most recent MEMO command succeeded or not.

Note:

Length and type must be defined as (16,MEMO). Currently, MEMO and BLOB are aliases, so "if: NOT BLOB" command will return the same result as "if: NOT MEMO". (MEMO fields are simply plain-text BLOB fields.)

lf: MEMO xx CO "mytext"

Determine if MEMO field xx contains a string "mytext".

Extended Functions and Help for "Inquire/Update/Add"

The internal filePro editor has extended functions and help. When a memo field is displayed in "Inquire, Update, Add", you will see "Press [F8] for extended functions, [F10] for help". When pressing [F8], a screen is displayed as shown in figure 1.

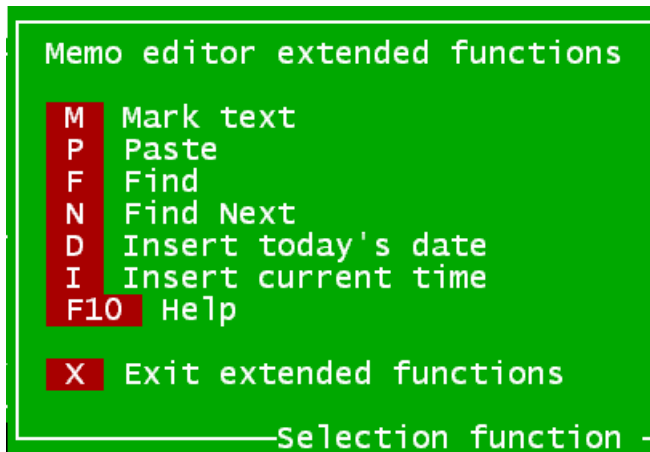


Figure 1 - Memo Editor Extended Functions

Although the extended functions are rather self-explanatory, it is important to note that the "Mark text" option allows you to select blocks of text that can be pasted to other records.

Mark Text - Allows you to use the cursor keys, PgUp, and PgDn to select text After selecting this option, another set of extended functions (as shown in figure 2) are presented when pressing the [F8] key.

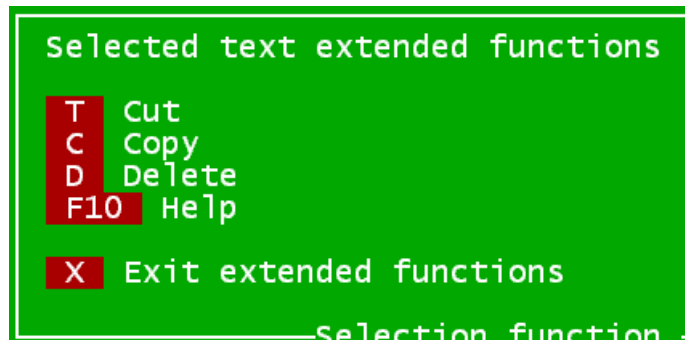


Figure 2 - Selected text extended functions.

This menu allows you to cut, copy, or delete the text you have selected. When selecting these options, the selected text is maintained in filePro's clipboard for pasting into the current record or another record. When pressing any of these extended function keys, you are returned to the memo editor screen and your selected text is stored in the clipboard.

You can press [F10] for help at any point while editing your memo fields. Figure 3 depicts the typical help for the "Memo Editor" functions.

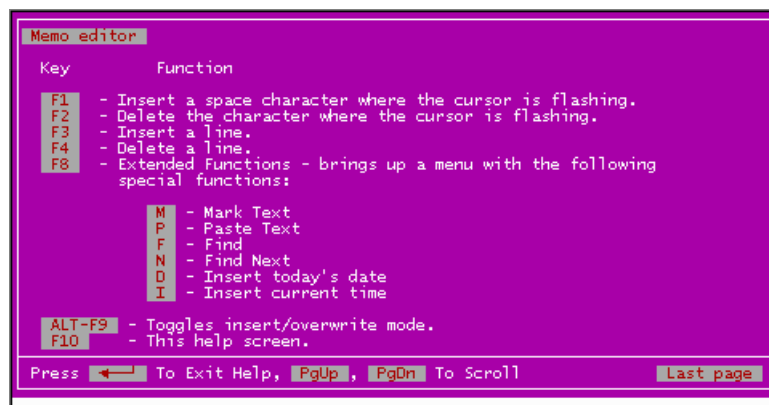


Figure 3 - Memo Editor Help

Note that the standard filePro keys also apply for the filePro "Memo Editor" functions. You can use [Alt] [F9] (DOS/Windows) or [Ctrl] [Z] (*NIX) to toggle INSERT/OVERTYPE mode while editing memo fields.

When pressing [F10] for help in the extended functions, a screen similar to figure 4 is displayed.

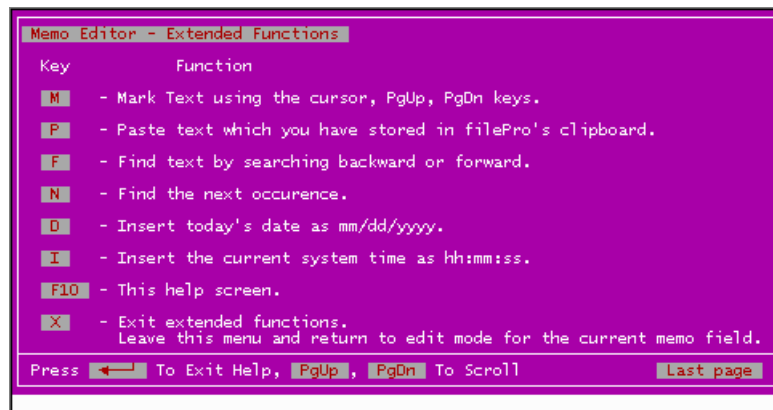


Figure 4 - Memo Editor Extended Functions Help

MENU

Syntax:

Then: MENU name label1,label2,...

Version Ref: 3.x

Description:

Puts up a menu that looks exactly like a standard filePro menu. It is composed of "name", an array defined with DIM.

There must be one more element in array "name" than choices on the menu. This extra element (the first element) holds the menu's title. All elements of array "name" are loaded in processing as in the following

```
name["1"]="The Menu's Title"
```

The first element of the array is always the menu's title, as show above.

The actual menu choices, descriptions, and actions are loaded into this array based on their physical (p)osition on the displayed menu. The syntax for loading the rest of the elements of the menu array is as follows.

```
name[p]="#:description"
```

p is the position on the menu, must resolve to "1" through "24".

is the menu-choice character, can be 0-9, A-W,Y,Z, case-insensitive, or any punctuation mark on the keyboard. X is reserved by filePro as an exit key for the menu.)

: is the mandatory separator between the choice character and the description

description is the text for the menu choice

The label list directs processing after the user has selected a menu item. The label is where processing does an "implied" GOTO for each choice on the menu. Every menu choice is related to this label list by its numerical position on the menu plus "1". In other words, the choice in name["5"] is tied to the 4th label in this list. If the user selects this choice (even though it may have a menu choice-character of "S") processing will begin immediately at the fourth label in the list. This is because the menu heading is always held in name["1"], thereby offsetting the physical choices on the menu and the label list by "1".

NOTE: Once processing is sent to the appropriate label by consequence of the user picking a menu choice, it will continue as normal and will NOT automatically return to the MENU command. If you want this to happen, you must arrange it that way. In other words, if you want the user to always be returned to the menu after executing any menu choice, then you must specifically arrange things this way, usually by putting a GOTO back to the original MENU command at the end of each label's processing routine.

Description:

MENU lets you create menus on processing tables. Processing menus let you select particular processing operations in the same way that user menus let you select particular programs from a list of programs.

The MENU command uses the same routines and defaults as Define User Menus. You can have up to 12 choices in a single, centered column or up to 24 in two columns; the entries "X - Exit" and "Enter Selection >" are added automatically; and any character except X or x can be used as a menu choice.

Examples:

Domen If:

```
Then: dim tstmenu["4"]
If:
Then: tstmenu["1"]="Order Processing"
If:
Then: tstmenu["2"]="1:Get order status"
If:
Then: tstmenu["3"]="2:Update pickticket screen"
If:
Then: tstmenu["4"]="S:Change order status"
If:
Then: menu tstmenu status,scr4upd,fix
If:
Then: end <=<user presses "X" processing falls to this line
```

Status If: 4 gt "1"

```
Then: show"@The status of this order is OPEN" ; goto domen
If: 4 le "1"
Then: show"@The status of this order is CLOSED";goto domen
```

scr4upd If:

```
Then: screen 4 ; goto domen
```

Fix If:

```
Then: input q(1,yesno) "Is this order locked? (y/n) "
If: q ne "Y"
Then: 14="N" ; goto domen
If:
Then: 14="Y" ; goto domen
```

IMPORTANT: Multi-user versions. If you execute a MENU command from @when processing or regular INPUT processing, the record on which you are standing is locked. This means that others wishing to access this record either in Inquire, Update & Add (rclerk, dclerk) or with output processing (rreport, dreport) will not be able to do so. They will get a "Record is being updated, access denied" message.

Therefore, use good judgment as to when to put up a processing menu and when to use another mechanism for providing choices and consequent actions. The MENU command can be used in @entsel processing, which does not lock the current record, if it will serve the desired purpose.

HINT: Processing menus can be made very dynamic by loading the menu array with varying text, perhaps based on user choices. The highlighted cursor can also be placed on desired menu choices from within processing by making use of the PUSHKEY command and arrow keys.

MERGEVAL Version 5.6

the same as `FIELDVAL()` but for `IMPORT` rather than `LOOKUP`. Gives you a way of using an expression for the field number.

That is:

```
xx = alias(3)
```

is the same as

```
xx = mergeval(alias,"3")
```

but you can also use something like `MERGEVAL(alias,something+"2")`

MESSAGE

Note: This command is only available if you have purchased the **fileProGI Developer Toolkit** . Refer to the documentation provided with the fileProGI Developer Toolkit for syntax and examples.

MID()

Syntax:

Then: a=MID(f,"s","n")

Finds and copies the middle of a field.

MID(f,"s","n")=newvalue

Replaces characters within a field.

"f" is the field on which to perform the operation.

"s" is the number of the starting position within the field.

"n" is the number of characters starting from "s" to use.

Version Ref: 3.x

5.0(Enhanced)

Description:

A system function used to find and copy the middle of a string, and to replace characters within a field.

5.0 Version 5.0 enhanced the MID function to allow expressions as the first parameter.

Example:

```
MESGBOX "You were born in the year" < mid(InfoFile[5],"7","4")
```

The above line gets 4 characters starting in position 7 from field 5 in the file named InfoFile.

Note :

If you use a "true" expression as the first parameter, you have to make sure you know the length/type of the result of the expression. It may not be the same as you think. For example, the expression "1"/"2" is of type (25,F). More likely, you will use this to place lookup fields or array items as the parameter.

Examples:

```
If: mid(@cd,"7","2") eq "97"
```

```
Then: show "This record was created in 1997." ; end
```

```
If: @id ne mid(14,"5","8")
```

```
Then: show "This invoice belongs to someone else." ; end
```

```
If:
```

```
Then: mid(14,"5","8")=@id ; end
```

The terms in the MID function are each expressions. However, the start position and the number of characters must be surrounded by quotes if you are using literal numbers. Using real fields in either of these positions requires that you understand that the value of these fields will be substituted before the function is applied.

MIN()

Syntax:

Then: a=MIN(n)

Then: n=MIN(aa,ab, ...)

Version Ref: 4.1

Description:

Finds minimum value of field "n" on output, or returns the lowest value in the list of fields given.

Examples:

For financial analysis, you want the average sales per sales representative, plus, for comparison, his or her maximum and minimum sales (see the "AVG" and "MAX" commands also). "Total Sales" is field 22. To find the minimum value, you'd put dummy field M on a subtotal line in the report format, and write the element as follows:

Then: M=MIN(22)

Find the minimum of a list of values.

Then: mx = MIN(ab,ac,ad,ae,5,12)

Restrictions:

For aggregate minimums, calculations are made at the subtotal and total breaks.

For comparative minimums, the fields do not have to be numeric, but they must all be comparable. For example, you may supply a list of dates or times, but not both. There must be at least two fields. If only one field is supplied, the aggregate form of the function is used.

MOD()

Syntax:

Then: MOD(exp1,exp2)

Returns the remainder of exp1 divided by exp2.

Version Ref: 4.1

Description:

Returns the remainder of a division. The first parameter is divided by the second parameter and the remainder is the value returned. The sign of the result is the sign of the original number.

Examples:

Then: mo =MOD(no,"100")

If "no" is equal to 247 then "mo" would be equal to 47 after the statement is executed.

mode(path/filename) (ver. 5.8.03)

Mode(path/filename) will return the octal permission mask on a file from within processing.

see also [group\(\)](#)

MOUSEPATH

Syntax:

Then: MOUSE PATH ON

Then: MOUSE PATH OFF

Version Ref: 5.0.9

Description:

Allows you to turn off forced cursor path in fileProGl. Default is ON meaning that fileProGl enforces screen cursor path

Note: Same as CURSOR PATH

MSGBOX

Syntax:

Then: MSGBOX(row,col) message,prompt,keylist

Version Ref: 4.1

Description:

The MSGBOX processing command lets you display a message in a popup window on the screen. It will display a string "message" in a popup window at a designated position on the screen until the user presses ENTER or a key from a specified list. The key pressed by the user can be captured and acted upon appropriately.

Row and Column

If row and column are supplied, the upper left corner of the window appears at the row and column coordinates. If row and column are not supplied, the window is centered on the screen. The upper-left corner of a screen is row,col ("1","1").

Width and Height

The width of the window is determined by the longest line of text. The height of the window is determined by the number of text lines.

Message

The message to appear in the window is a string expression. You can add additional lines to this message by placing a "\n" at the point where you want a new line to begin. All of the SHOW codes are available to MSGBOX. For example:

Then: msgbox "Text line 1\nText line 2"

Prompt

Prompt is the prompt you want to appear telling the user what keys can be used to remove the window (and/or take certain actions) to continue. The prompt appears in the lower-right corner of the window. If prompt is not specified, then the default prompt of "Press ENTER" is used.

Keylist

Keylist is the list of keys, in addition to ENTER, that will remove the window. These keys can be trapped and acted upon with further processing. If keylist is not specified, the ENTER key is the default for removing the window.

NOTES: ENTER is the special key defined as ENTR. MSGBOX will display what is set as the keylabel for this key.

After the message box is executed, @BK is set to the keystroke entered.

The arguments to MSGBOX (row,col, the message, the prompt and the keylist) can be variables. If they are variables (or real fields), you must NOT put quotes around them. If they are literals, such as the keylist below (SLN), then they must be in quotes. Case is not significant (except if the key is a shifted key such as "@", then a 2 will not work, only a shifted 2 "@!" will work).

MSGBOX is exactly the same thing as ERRORBOX except for the colors which are assigned to it. The colors for MSGBOX are POPUPNORMAL and POPUPINVERSE. This environmental variables allow you to give a consistent look and feel to your programming by showing "errors" in one set of colors and "messages" (or navigational questions) in another set of colors.

Examples:

```
If: 4 ne "0"
Then: end
If:
Then: o="NO money on this record. Save it, Delete it, or Fix it?"
If:
Then: m="(s/d/f)====>"
Then: msgbox("10","5") o,m,"SDF"
If: @bk eq "S"
Then: end
If: @bk eq "D"
Then: delete ; end
If: <== the F key in keylist will default here, so will ENTER.
Then: restart
If: exists(fn) gt "0"..
Then: msgbox "File already exists!\nContinue?",
"Press \rY\r or \rN\r", "YN"
If: @BK eq "Y"
Then: gosub mkfile
```


NEXTDIR()

Syntax:

Then: XX=NEXTDIR()

The format of the returned fixed length string is:

Format	Extension	Size	Date	Time	Fullname
32	10	14	10	9	32
Abcdefghijklm	abcdefg hij	01234567891234	01/01/2000	12:12:34a	-----

There is a single space between each of the portions of the string.
 The format, extension, and Fullname are left justified.
 The size is right justified with commas.
 The date and time are right justified.

Version Ref: 4.8

Description:

Each execution of this command returns a detailed file specification that the preceding OPENDIR() located. Subsequent NEXTDIR() allows you to build a list of files found with sizes, dates, etc.

NOT HTML (not included in filePro Lite)

Syntax:

If: NOT HTML Then: "Do Something"

Version Ref: 4.8

Description:

Tests for a HTML command on a previous line for the result code indicating success or failure.

NUMFIELD()

Syntax:

Then: xx = NUMFIELD(lookupname)
Then: xx = NUMFIELD(-)

lookupname is the name of a lookup.

- represents the current file (file in which the processing resides).

The return value is the number of fields in the designated file. This value is assigned to the variable XX.

Version Ref: 4.5

Description:

NUMFIELD obtains the number of fields in a lookup file.

Examples:

Then: lookup cust k=1 i=a -ex
Then: aa = numfield(cust)

NUMRECS()

Syntax:

Then: xx = NUMRECS(lookupname)
Then: xx = NUMRECS(-)

lookupname is the name of a lookup.

- represents the current file (file in which the processing resides).

The return value is the number of records in the designated file. This value is assigned to the variable XX.

Version Ref: 4.5

Description:

NUMRECS obtains the number of records in a filePro file.

IMPORTANT: Deleted and unused (blank) records are included in the count.

Examples:

If:
Then: lookup cust k=1 i=a -ex
If:
Then: aa = numrecc(cust)

OPEN()

Syntax:

Then: aa = OPEN(filename,mode)

"filename" is the name of the file to be opened.

"mode" is the access mode to use (see below).

"aa" is the file handle.

Parameters

mode r - read access

w - write access

c - create if doesn't exist

0 - truncate if already exists

b - binary mode

t - text mode

Version Ref: 4.5 (not included in filePro Lite)

In Version 5.7.04, and new flag A was added.

Anything written to the file will be appended to the end of the file.

Description:

This function returns a file handle to access the opened file through the other file I/O functions.

If the file cannot be opened, a negative number is returned. This number is the negative of the system error number.

Note : The file handle returned is analogous to the file handles used internally by the operating system, but the numbers are not necessarily the same. Mode flags can be combined. For example, rwb means read and write access, in binary mode. If neither "r" or "w" is supplied, read-only is used.

Binary/text mode does not affect Unix systems, as Unix makes no distinction. Under MS-DOS, you must make sure to specify binary/text mode as appropriate. If neither is specified, binary mode is used. Note that for portability purposes, you should specify text mode under Unix if appropriate.

Examples:

Then: mo = "/tmp" { @jd { @m

Then: tf = open(mo,"rwc0")

Set a filename based on the user id and record number being updated. Open this file for reading and writing. Create the file if it doesn't exist, and truncate it, if does. This file can now be used by referring to it as "tf" with most of the other file I/O functions.

OPENDIR() (not included in filePro Lite)

Syntax:

N = opendir(format type, filename)

N = opendir()

N = opendir("PRC_MASK")

N = opendir("SCR_MASK", "fpcust")

N = opendir("*.html")

no format type, no filename - Creates a list of all filePro files.

Creates a list of all prc files for the current file.

Creates a list of all screens for the fpcust file.

Creates a list of all *.html screens in the current files directory.

The preset terms that the OPENDIR() function understands are as follows.

Mask:	Windows:	UNIX:
PRC_MASK	.prc	Prc.*
TOK_MASK	.tok	Tok.*
SCR_MASK	screen.*	Screen.*
IDX_MASK	index.*	Index.*
BRW_MASK	.brw	Brw.*
OUT_MASK	.out	Out.*
SEL_MASK	.sel	Sel.*
HTML_SCR_PRC_MASK	scr*.prc	prc.scr*
HTML_SCR_TOK_MASK	scr*.tok	tok.scr*
HTML_BRW_PRC_MASK	brw*.prc	prc.brw*
HTML_BRW_TOK_MASK	brw*.tok	tok.brw*

Version Ref: 4.8

Description:

OPENDIR() allows you to do the equivalent of a DOS "DIR" or UNIX "ls" command. It returns the number of files in a filePro directory that meet the criteria.

Version 6.0.01

NEW command OPENDIR2 to handle long-named files and paths.

e.x.

N = OPENDIR2(mask, path, fmt_sz, ext_sz, nam_sz)

All arguments are optional.

Returns the number of files which match the parameters. If no parameters are given, a count of all filePro filenames is returned. If only a mask is specified, it is applied to the current filePro file. There are pre-defined masks for filePro files, but any mask may be used. Only one OPENDIR() or OPENDIR2() may be active at one time.

mask - the filter applied to the directory, same as OPENDIR()

path - the directory to use

fmt_sz - the total size to use to store the format (default 32)

ext_sz - the size used to store the extension (default 10)

nam_sz - the size used to store the filename (default 32)

Note: the numbers used to define the lengths change the size used by the system controlled "@DIRLIST" arrays.

OUTS

Syntax:

Then: OUTSx

Sends "x" to the serial port as described by the environment variable PFOUTS.

Version Ref: 4.0

Description:

Sends a string of characters to a serial port.

Examples:

OUTS 3

Will send field 3 to the serial port.

The environment variable PFOUTS must be in the environment. It describes the port name, and attributes of the serial protocol, i.e., baud rate, word length, parity, and stop bit.

PFOUTS is set as:

```
PFOUTS="tty1A,9600,8,N,1" (Unix)
set PFOUTS "1,9600,8,N,1" (DOS)
```

Autodialer

(Assumes a modem is connected to the designated port.)

```
@keyP If: 'dial a modem and hang up (after timeout)
Then: 'so I one can pick up the phone and talk
Then: pb(14)=20
If: pb co "("
Then: pb="1"{mid(20,"2","3"){mid(20,"7","8")}
Then: outs "ATDT"{pb};H "{chr("13")"
```

The above code strips out characters that a modem doesn't like and prepends a 1 if the call is long distance. The OUTS command dials the number on a cheap modem attached to this line and waits just long enough for you to pick up the phone and talk...by then, the modem has hung up gracefully. This allows one to pull up any filePro record that has a phone number in field 20 and instantly dial it by pressing "P".

PAGE

Syntax:

Then: page

Available only for reports.

Version Ref: 3.x

Description:

Forces end of page when printing reports.

PAGE, in conjunction with the system-maintained field @LC, "line count," is used to set a particular page size for reports. When PAGE is encountered on an output processing table, the program does a form feed and prints the report's heading at the top of the next page.

Examples:

To obtain a new page every 45 lines or whenever the total dollar amount on the page exceeds \$1000.00. You could set up the processing as follows:

```
Then: to(7,.2,g)=tot(10)
      If: @lc eq 45 or to ge "1000.00"
      Then: page; to=""
```

where "to" is the total field and "@lc" is the system-maintained line-count field.

PDF_CLOSE()

Version Ref: 6.1 (USP6.1.01)

Syntax:

```
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.

See also: [Fill-In-The-Blank PDFs](#)

PDF_FIELDTYPE)

Version Ref: 6.1 (USP6.1.01)

Syntax:

```
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

See also: [Fill-In-The-Blank PDFs](#)

PDF_FIELDTYPE2()

Version Ref: 6.1 (USP6.1.01)

Syntax:

```
name = 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](#).

See also: [Fill-In-The-Blank PDFs](#)

PDF_GETFIELDNAME()

Version Ref: 6.1 (USP6.1.01)

Syntax:

```
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`.

See also: [Fill-In-The-Blank PDFs](#)

PDF_GETNUMFIELDS()

Version Ref: 6.1 (USP6.1.01)

Syntax:

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

See also: [Fill-In-The-Blank PDFs](#)

PDF_GETVALUE()

Version Ref: 6.1 (USP6.1.01)

Syntax:

```
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.

See also: [Fill-In-The-Blank PDFs](#)

PDF_GETVALUE2()

Version Ref: 6.1 (USP6.1.01)

Syntax:

```
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](#).

See also: [Fill-In-The-Blank PDFs](#)

PDF_OPEN()

Version Ref: 6.1 (USP6.1.01)

Syntax:

```
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.

See also: [Fill-In-The-Blank PDFs](#)

PI()

Syntax:

result = PI()

Version Ref: 4.8

Description:

Trig Function: Returns the value of PI or 3.14159265

POFFIELD

Version 6.0.01

POFFIELD will allow you to move to a specific field in a POPUP UPDATE

Syntax: POFFIELD, fld

POPUP

Syntax:

Then: POPUP(row,col) f,s

Displays screen "s" of lookup file "f" at "(row,col)". If position is omitted, popup will be centered. If - (hyphen) is used for the filename, then the screen must be in the current file and no lookup is needed. A POPUP must be taken down (cleared from the screen) with the CLEARP command.

Version Ref: 4.0

Description:

Pops up a specified screen, from the lookup file, to display the record specified by the lookup. The lookup file screen will be sized for display in the popup screen by eliminating unused space to the right and bottom of the screen. The location of the upper left corner of the popup screen can be specified (optional) If no location is specified, the popup screen will be centered on the current screen.

There are two ways you can use a pop-up screen: 1) to view and update a record in the lookup file; 2) to view and update a record in the current file. In either case, a popup will not remain on the screen unless it is forced to do so. It will simply display and immediately clear too fast for the user to see. In order to keep a popup visible, you must use another filePro command. Usually, the command SHOW "@@" is placed immediately after a POPUP which will keep it on the screen until the user presses <RETURN>.

Pop-up Screen From a Lookup File

Then: popup("r","c") lookup,screen

"r" (optional) is the row coordinate of the upper left corner of the popup screen

"c" (optional) is the column coordinate of the upper left corner of the popup screen

"lookup" is the assigned name of the lookup specifying the file and record to display in the popup screen

"screen" is the name or number of the screen in the lookup file to display in the popup screen.

Pop-up Screen From the Current File

Then: popup("r","c") -,screen

"r" (optional) is the row coordinate of the upper left corner of the popup screen

"c" (optional) is the column coordinate of the upper left corner of the popup screen

A dash (-), used in place of a lookup name. This will displays the current record of the current file on a screen in the current file. No LOOKUP is needed in this case.

Only one popup screen can be displayed at a time. Putting up a second popup screen will cause the first to be taken down. (See the CLEARP command below for clearing a popup screen.) If you use a popup screen in conjunction with a browse lookup, they must be taken down in the reverse order.

Pop-up Screen and Passwords

Even though a password is assigned to a screen when defining a screen, since its decided at design time when to use a POPUP screen, the screen password is ignored. Use INPUTPW or other means to control access to POPUP screens when included in your lookups.

POPUP UPDATE

Syntax:

Then: POPUP(row,col) UPDATE f,s,m

Displays screen "s" of lookup file "f" at "(row,col)" and puts the user into update mode on this screen, positioning the cursor in field "m". If (row,col) is omitted, the popup will be centered. If - (dash) is used for the filename, then the screen must be in the current file and no lookup is needed. If field is omitted, the cursor is placed in the first field on the popup screen. A POPUP UPDATE will stay up until taken down by a CLEARP command, even after the user has pressed the BREAK or SAVE key to leave the POPUP UPDATE.

Version Ref: 4.0

IMPORTANT: The special benefit of POPUP UPDATE is that the contents of the POPUP screen will be saved to the looked-up record (or to the current record if - is used as the filename). Thus, if you look up a particular customer record and POPUP UPDATE a screen from that file and the user makes changes to the fields on this screen, these changes will automatically be written to that record in the customer file when the user presses the SAVE key. If the user presses the BREAK key, the looked-up record will not be changed.

System-maintained field, @SN, will contain the name of the popup screen

HINT: The following benefit applies to using POPUP UPDATE with the current file (dash): When-processing (@WEF, @WLF, @WHP, and @WBL, etc.) on this specific type of POPUP UPDATE can be performed by doing it from dummy fields placed on this screen. This is an extremely valuable processing construct, since processing is not normally allowed while on a POPUP UPDATE screen.

Pop-up Screens and Passwords

Even though a password is assigned to a screen when defining a screen, since its decided at design time when to use a POPUP screen, the screen password is ignored. Use INPUTPW or other means to control access to POPUP screens when included in your lookups.

CLEARP

Syntax:

Then: CLEARP

Examples:

Description:

This command removes a popup and cleans up the screen. It is a good idea to always use this command after a popup.

PRINT

Syntax:

```
Then: print
```

PRINT can be used on output processing tables only.

Version Ref: 3.x

Description:

Prints current record.

IMPORTANT: If a PRINT statement is used anywhere on the processing table, then it **MUST** be encountered or the record will **NOT** print.

Examples:

```
If: 4 ne "C"  
Then: end  
Then: print ; end
```

When the above code is run against a group of records, only those which have field 4 equal to "C" will print on the report. All other records will not show up on the report. Normally (without any PRINT statement on the processing table) every record will be printed on a report. This is a very powerful feature of PRINT and you must be aware of exactly how it functions or you will not understand why some records are printing and others aren't. If the PRINT statement is encountered AND it has a TRUE "if" condition or a blank "if" condition (which defaults to TRUE), the record will print, otherwise, it will be dropped off the report and, more importantly, **OUT OF ANY TOTALING** that filePro is doing for you automatically (using =N on the output format)!

PRINT lets you control the printing of individual records from processing. Use PRINT to do such things as processing all records but printing only certain ones; and printing more than one copy of a record. To print multiple forms per record, you can set up a counter and then loop through the set of elements until the count is exhausted.

```
Then: input ct(2,0) "Print how many copies?"  
loop  If: ct gt "0"  
Then: print; ct = ct-"1"; goto loop
```

PRINTCODE()

Version Ref: 6.2 (USP6.1.02)

c = 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		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")
```

PRINTER Commands

Syntax:

```
Then: printer "printercommand"  
Then: printer local  
Then: printer file "filepathname"  
Then: printer reset
```

```
Then: printer flush
```

where:

"printercommand" is the same as an operating system printer command (UNIX/XENIX). An example is "printer lp-dlaser" This will override the Printer Destination (@PD) value that is set from the default printer setup.

"printer local" redirects output to a printer connected to the user's terminal (UNIX/XENIX)

"filepathname" is the name of a file to which the output is redirected (UNIX/XENIX or DOS). (An example: printer file "/tmp/output".) "filepathname" can also be a destination file name for going directly to a printer (bypassing the spooler). Example: "printer file /dev/tty01"

"printer reset" cancels any previous printer commands in processing and returns to whatever printer command was in effect before processing started (UNIX/XENIX or DOS)

"printer flush" flushes any printer buffers within filePro.

Version Ref: Various

Description:

PRINTER lets you redirect output to a printer other than the system printer, or to a printer file.

PRINTER can be used on all processing tables. One obvious use for PRINTER is in input processing, where you might want to send forms and hardcopies to a special printer.

PRINTER "command"	Sends output to "command". (UNIX/XENIX only)
PRINTER FILE "filename"	Sends output to file "filename".
PRINTER LOCAL	Sends output to terminal printer. (UNIX/XENIX only)
PRINTER NAME "name"	Sends output to printer "name".
PRINTER TYPE "type"	Sets printer type.
PRINTER RESET	Resets the printer.
PRINTER @PN	Resets page back to 1
PRINTER FLUSH	Flushes printer buffers within filePro.

IMPORTANT: If you use one or more of the other PRINTER functions, you must execute a PRINTER RESET before altering any of them or using new ones. The PRINTER RESET clears any PRINTER operations from the processing table and puts things back to the condition they were in when the processing table was first encountered. This is an important function to remember when a series of PRINTER routines does not function the way you think it should.

PUSHKEY

Syntax:

```
Then: PUSHKEY "c"  
Then: PUSHKEY "[code]"  
Then: PUSHKEY "abc[code]xyz[code]..."  
Then: PUSHKEY exp
```

c = character(s)

[code] - Special Keys (those returned by @sk, like "[ENTR]" or "[SAVE]" or "[CRUP]").

exp - The argument to PUSHKEY can be an expression that resolves to keyboard codes and characters.

Version Ref: 4.0

Description:

Places "keystrokes" into the keyboard queue as if they were pressed by the user.

Examples:

@keyP If:

```
Then: PUSHKEY "u[CDWN][CDWN]33[ENTR]open[SAVE]" ; end
```

The above code will put users into UPDATE mode, move the cursor down two fields, type the number "33", press ENTER, type the word "open" and then press the SAVE key for them. With one keystroke, the user performs many keystrokes.

Note: Two things are important to understand about how PUSHKEY operates. First, the programmed sequence of keystrokes happens from exactly where the cursor is positioned when the next END statement is reached. Second, nothing happens until that END statement is reached, and then the PUSHKEY executes.

Examples:

To run an extended selection set called "new," before any records are updated. You could use PUSHKEY in your @ENTSEL processing as follows:

```
Then: pushkey "X22Lnew[ENTR][ENTR]"; end
```

"X" Exits from the current record.

"2" Selects The "scan For Records" option from the Inquire, Update, Add, menu.

"2" Selects the "Extended Selection" option from "Scan For Records" submenu.

"L" Loads a selection set.

"new" Inputs the name of the selection set.

"[ENTR]" Inputs the four-letter special key code for a <RETURN>, to enter the selection set name.

"[ENTR]" Inputs the four-letter special key code for a <RETURN> to execute the selection and retrieve the records.

"end" Ends the @ENTSEL when-processing

NOTE As of Version 6.0 and higher, if an extended selection is set, all functions of dclerk will honor the selection criteria until the extended selection is manually cleared. (Option 3)

PUSHKEY accepts "[BRKY]". When PUSHKEY puts "BRKY" into the Inquire, Update, Add buffer, it performs as though the BREAK key was pressed. The following example shows how to use this feature to prevent anyone from updating or adding records:

@update If:

```
Then: errorbox "update not allowed."
```

```
  If: @OS = "DOS"
```

```
Then: pushkey "[BRKY]" ; screen; end
```

```
Then: pushkey "[BRKY][BRKY]" ; screen ; end
```

Note: Remember that @UPDATE must use the SCREEN command. Using @keyU would not prevent records from being added through "Add Records" mode. Also, in "Add Records" mode, "[BRKY][BRKY]" will prevent a blank record from being created.

PUTENV

Syntax:

Then: `PUTENV envname,value`

where "envname" is the name of the environment variable.
"value" is the value to assign to this variable.

Note: Any value set through PUTENV is lost upon exit from running the filePro program.

Version Ref: 4.5

Description:

Stores a environment value in the environment of programs called through the SYSTEM command.

Examples:

Run a program that requires "DOC" to point to a document directory.

Then: `PUTENV "DOC",getenv("PFDATA") & "/" & getenv("PFDIR") & "/docs"`

Then: `SYSTEM "rundoc"`

QRCODE()

Version Ref: 6.1 (USP6.1.01)

Syntax:

```
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:

If:

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

RAND()

Syntax:

Then: a = RAND(n)

Returns a pseudo-random number between 0 and 32767.
Uses "n" as optional seed value.

Version Ref: 4.1

5.0(Enhanced)

Description:

Returns a pseudo random number.

5.0 Enhanced to allow a negative seed value with current time to seed the generator.

Examples:

Then: aa = mod(rand(xx), "100") + "1"

GETRAND if: inkey ne "BRKY"

Then: aa = mod(rand(" "), "100") + "1"; display; goto GETRAND

The above example returns a random number between 1 and 100. Notice that the seed value "rand(xx)" is only used ONCE and the second element, "GETRAND" loop, uses a NULL value for RAND e.g. rand(" "). RAND with a positive number should be seeded only once to work properly. For 2nd and subsequent iterations, use a NULL e.g. RAND(" ").

Then: xx = RAND("-1")

The above seeds the generator with a negative value using current time.

READ()

Syntax:

Then: bytes = READ(handle,destination,length)

Then: bytes = READ(handle,destination)

"handle" is the file handle returned by OPEN() or CREATE().

"destination" is the field to store the information.

"length" is the number of bytes to read.

"bytes" is the number of bytes read.

If length is not specified, the length of the destination field is used.

Return value

Returns the number of bytes read. 0 usually indicates end-of-file.

In a binary file, a return value of less than the request amount usually indicates that end-of-file has been reached. In a text file, it is possible to read fewer bytes than requested, even if EOF is not encountered.

Note: Destination must be a field, not an expression.

Version Ref: 4.5 (not included in filePro Lite)

Description:

Reads from a file.

READLINE()

Syntax:

Then: bytes = READLINE(handle,destination,length)
Then: bytes = READLINE(handle,destination)

"handle" is the file handle returned by OPEN() or CREATE().
"destination" is the field to store the information.
"length" is the maximum number of bytes to read.
"bytes" is the number of bytes read.

If length is not specified, the length of the destination field is used.

Version 6.0.01.07 - using "-1" for length to force read to EOL

Return value

The number of bytes read, not including the newline character.

Destination must be a field, not an expression.

Data is read from the file up to a maximum of length bytes, or until a newline character is encountered, whichever comes first. The newline character is discarded if encountered.

Version Ref: 4.5 (not included in filePro Lite)

Description:

Reads a line of text from a file.

Note: The file must be opened in text mode, or undefined behavior may result.

READMAP

Version Ref: v6.2 (6.1.01 USP)

```
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.

Example

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:"

READOUTPUT()

Syntax:

XX = READOUTPUT(filename,formatname [, section])

where "filename" is the name of the filePro file, "formatname" is the output format name and section (optional) is the type of information you want returned.

Version Ref: 4.8 (not included in filePro Lite)

5.0 (Enhanced)

Description:

READOUTPUT returns the portions of an output format.

Notes:

Section parameter options include:

0 = Returns the text portion of the output format, convertinig graphics characters to "+-|".

1 = Returns the text portion of the output format but returns graphics characters as-is e.g (0x00-0x0A).

2 = Returns the text portion of the output format but returns graphics characters as spaces.

3 = Returns only the graphics characters, as "+-|" text.

4 = Returns only the graphics characters, as-is.

5 = Returns size information of the format, as a list of colon-separated 3-digit values as "nnn:nnn:nnn:" consisiting of:

width, length, type, # across the page, # down page, # of header lines, # of footer lines (always zero), # of data lines, # of lines per subtotal/grandtotal break.

Notes :

Codes 3 through 5 were added as a version 5.0 enhancement. For codes 3 and 4, all regular text is returned as spaces.

If no section is specified, the default section is "0".

If the output format cannot be read (ie. Doesn't exist) then a null value is returned.

READSCREEN()

Syntax:

Then: `XX=READSCREEN(filename,screenname[,section])`

"filename" is the name of the filePro file

"screenname" is the screen name.

The optional third parameter is the section of the screen to be read.

Version Ref: 4.8 (not included in filePro Lite)

5.0 (Enhanced)

6.2 (Enhanced)

Description:

READSCREEN returns the text portion of a screen as a single 1600 character field.

Currently supported sections are:

Third parameter options

0: Graphics characters are returned as text equivalents. (i.e., '|', '-', '+')

1: Graphics characters are returned as is.

2: Graphics characters are returned as spaces.

3: Return only the graphics characters, as "+-|" text.

4: Return only the graphics characters, as-is.

5: Return size information of the screen.

- width
- height
- format type
- records across page
- records down page
- number of header lines
- number of footer lines (always zero for now)
- number of data lines
- number of break levels (including grand total)
- 9 entries, one for each break level, innermost first

6: Return the color attributes as a 1600-character field.

7: Returns cursor path in a colon separated list of fields.

Notes:

Section codes are optional parameters and if left blank will default to code "0".

If the screen cannot be read (ie: doesn't exist) then READSCREEN() returns a null value. For code 0, text equivalents for graphic characters are '|', '-', '+'.
Codes 3 through 6 were added as a version 5.0 enhancement. For codes 3 and 4, all regular text is returned as spaces.

Code 7 was added as a version 6.2 (6.1.02 USP) enhancement.

If the screen cannot be read (i.e., doesn't exist), then READSCREEN() returns a null value.

REPLACE() (version 5.8.03)

This processing command will return a search and replace string of data from either a field or variable.

Syntax is: `xx = replace(fld,from,to [,flag])`

For example:

```
yy="This is a test"; xx=replace(yy, "a test", "best")
```

yy will equal "This is a test" and xx will equal "This is best"

Flags:

"0" or no flag is case insensitive for the search

"1" is case sensitive for the search

"0" is the default

RECLEN()

Syntax:

Then: `XX = reclen()`

Then: `XX = reclen(lookupname)`

Given no argument (), RECLEN returns the record length of the current file. Otherwise, the function returns the record length of the specified lookup file.

Version Ref: 4.8

Description:

RECLEN() returns the record length of the specified file.

REMOVE()

Syntax:

Then: aa = REMOVE(filename)

"filename" is the name of the file to be removed.

Return value

0 if successful

Negative if failed

Note : The file must be closed in order to be removed.

Version Ref: 4.5

Description:

Removes a file from the disk.

REPEAT()

Syntax:

Then: `xx = REPEAT(text,len)`

"text" is the string of characters to repeat.
"len" is the number of characters in the result.

Return value:

A string of len characters, created by repeating the characters in text as many times as necessary.

Version Ref: 4.5

Description:

Repeats a string of characters.

Examples:

Draw a horizontal line

Then: `show("10","1") repeat("\ G0","240")`

Use PUSHKEY to push 10 right-arrowkeys

Then: `pushkey repeat ("[CRGT]","60")`

RESET

Syntax:

Then: reset

Version Ref: 3.x

Description:

Reruns an output process against the selected records without re-selecting them. It must be used from within @wgt processing.

Examples:

```
@wgt If:  
Then: hh="newvalue";reset;end
```

RESTART

Syntax:

Then: restart

Version Ref: 3.x

Description:

RESTART returns the user to the beginning of the processing table and puts him or her back in update mode on the first screen updated. When he or she re-records the input, the program re-processes the record, starting from the top of the table.

Examples:

You want to make sure the user enters the account number, since the following operations depend on that number being there. You could write the element as follows:

```
If: 5 eq ""  
Then: SHOW "You Missed Account Number" ; restart
```

if field 5 is empty, the message "You Missed Account Number" appears on the screen. The user is returned to the first field on the first screen so that he or she can type in a valid number.

RESTART can only be used on INPUT processing. It must be the last command on the line, since anything after it will be ignored.

RETURN

Related Commands

GOSUB

Syntax:

Then: RETURN

Version Ref: 3.x

Description:

RETURN ends a GOSUB routine, and returns the processing to the first statement following the current GOSUB statement. RETURN must be the last command on any "Then" line, since processing following it on that line will be ignored. RETURN must not be encountered without a previous GOSUB.

Examples:

```
@mf6 Then: GOSUB "addup";end
```

If: 'any number of lines containing

Then: 'other code

```
addup If: 'now lets total a few fields
```

```
Then: tt(8,,2)=5+6+7+8+9+10
```

If: 'now we will show the total and return where we left off

```
Then: show"@Total is $" { tt; RETURN
```

The subroutine "addup" can be called by @mf6,7,8,9 and 10 rather than totaling these fields 5 different times on the table.

RTOD()

Syntax:

Then: $\text{result} = \text{RTOD}(\text{angle})$

Version Ref: 4,8

Description:

Converts radians to degrees.

Examples:

Then: $\text{angle_in_degrees} = \text{RTOD}(1.5678)$

SAVE

Syntax:

Then: SAVEON

Then: SAVEOFF

Version Ref: 5.0

Note: ESCAPE=OFF will also disable saving of records. SAVE is synonymous to "ESCAPE". SAVE=OFF will disable saving of records. Default is "ON".

SCREEN

Syntax:

Then: SCREEN s
Then: SCREEN s,f
Then: SCREEN s,f,o

SCREEN has several formats available, each allowing the desired level of cursor control.

SCREEN	Refreshes the current screen, and places cursor in first field of cursor path.
SCREEN s	Switches to screen "s", then positions the cursor in the first field of the cursor path
SCREEN s,f	Switches to screen "s", then positions the cursor at field "f".
SCREEN s,f,o	Switches to screen "s", then positions the cursor in field "f" at offset "o". "o" is the offset within the specified field on which to position the cursor. NOTE: An offset of -1 will place the cursor in the first empty position of the field.

Version Ref: 6.0.02

SCREEN command can switch fields in a POPUP UPDATE -, provided no screen name is passed to the SCREEN command.

Examples:

```
SCREEN ,33  
SCREEN ,pa,"-1"
```

When a screen number, letter, name or expression is given, SCREEN rewrites the entire screen. In other words, all SHOW or other messages are cleared from the screen, and all fields are displayed with their current value. When no screen number, letter, name or expression is given, the program rewrites and redisplay the changed fields without clearing messages or rewriting unchanged fields.

NOTE: If no cursor path is defined for a screen, and no field specified by the SCREEN command, the cursor will be put in the first non-protected field on the screen.

Version Ref: 4.5

VERY IMPORTANT: The parameters "s", "f", and "o" are expressions. Since SCREEN will recognize single number and single letter screen names, and also recognize unquoted field numbers and letters as field designations, you must enclose any of these parameters in parentheses if you want them to be resolved as expressions. The full version of the SCREEN command can therefore be thought of as follows:

Then: screen (exp), (expf), (expo)

where:

"exp" is any expression evaluating to a screen number, letter, or name

"expf" is any expression evaluating to a field number or letter.

"expo" is any expression evaluating to an offset.

```
Then: screen 3,2,"3"  
Then: screen (aa),"4","3"  
Then: screen A,B,"3"  
Then: screen "3","2","3"  
Then: screen "3","2",(aa)  
Then: screen "inv","2","1"
```

The above are all valid commands, and:

```
Then: screen 33,2,"3"  
Then: screen aa,"4","3"  
Then: screen A,B,3  
Then: screen "3",22,"3"  
Then: screen "3","2",aa  
Then: screen inv,"2","1"
```

the above are all invalid commands and will generate an syntax error. Note that the offset parameter requires quotes if it is to be a number, it does not behave like the "s" and "f" parameters.

NOTE: If you want the cursor to go to the first available space in the field (the character after the last non-blank character), use -1 as the offset, using screen 0, field 1.

Then: screen 0,1,"-1"

or

Then: screen "0","1","-1"

NOTE: Screen formats in filePro can have names of up to 7 characters for UNIX/XENIX systems, and up to 3 characters for DOS systems. A named screen is accessible only through processing. Note that single letter screen names, if not in quotes, are always treated as uppercase. For screen names of more than one character, in quotes, case is significant on UNIX/XENIX systems, but not on DOS systems.

Description:

On the Automatic table, SCREEN is used to access one screen rather than another, and make this screen the default screen. The default screen is the screen Input processing and @KEY processing will return to when an END statement is encountered.

When a SCREEN command is encountered during Input or @KEY processing, the user is put into update mode on the designated screen. When the user presses <ESC> to record the input on this screen, processing will resume with the command immediately following the SCREEN command.

When a SCREEN command is encountered during @WLF or @WEF processing, the user is put into update mode on the designated screen. When the user presses <ESC> to record the input on this screen, processing ends. @WLF and @WEF processing are not considered Input processing. These routines run independently from Input processing, and do not affect the position of the Input processing's execution pointer. Each @WLF and @WEF routine has its own pointer, and processing ends for these special routines only when one of the specific @WLF, @WEF "closing" commands is encountered. SCREEN is one of these commands.

Restrictions:

If there is more than one SCREEN statement on an automatic processing table, only the last SCREEN statement encountered is performed.

SCREEN can only be used on Input and Automatic processing tables.

SEEK()

Syntax:

Then: aa = SEEK(handle,offset,whence)
Then: aa = SEEK(handle,offset)

"handle" is the file handle returned by OPEN() or CREATE().
"offset" is the position within the file.
"whence" flags how to interpret the offset.

Parameters

whence 0 - absolute position in the file
1 - position relative to the current position
2 - position relative to end-of-file

Return value

Returns the resulting absolute position in the file.

Note: Offset can be negative when using relative positioning.

Version Ref: 4.5

Description:

Sets the current location within an opened file.

SELECT

Syntax:

Then: select

SELECT can only be used on sort/selection processing tables.

Version Ref: 3.x

Description:

Selects records for further processing by an output table.

SELECT lets you control the selection of individual records while on a sort/selection (-v) table. Like the PRINT command, a record is selected only if it encounters the SELECT command (i.e., it passes a selection condition).

Examples:

```
If: 4 ne "C"  
Then: end  
Then: select ; end
```

When the above code is run against a group of records, only those which have field 4 equal to "C" will be selected and handed to the output process. All other records will not show up in the output. Normally (without any SELECT statement on the processing table) every record will be selected and passed to the final output process. This is the main and only purpose of the SELECT command, and you must be aware of exactly how it functions or you will not understand why some records are showing up in your outputs and others are not. Not only do records have to encounter the SELECT command to be chosen for inclusion in the output phase, but they must meet the designated criteria as well, i.e., if the SELECT statement is encountered AND it has a TRUE "if" condition or a blank "if" condition (which defaults to TRUE), the record will be selected, otherwise it will be dropped out of the output.

Note: SELECT can not select a record more than once.

SELECTBOX () Version 5.8.02

This command works similar to LISTBOX() but allows for type-ahead of the listed words.

```
aa=SELECTBOX(array,first,last,row,col,height,width,prompt_string,prompt_row,prompt_col, case,init)
```

Example:

```
n=selectbox(dat,"1","25","5","5","8","60","Type here ","1","2","1")
```

Values for case are:

0 = case insensitive (default)

1 = case sensitive

2 = 'filename' sensitive - that the input is case-sensitive based on the O/S filename case sensitivity. That is,

Windows is case-insensitive, and Unix/Linux are case-sensitive.

This is useful when being used to select a filename from a list

init is the starting position when rendered

NOTE: This variable can be set in the config file to determine the default case value so that it does not need to be programmed on the command line.

```
PFSELECTCASE=n (n = "0", "1", or "2", default=0)
```

NOTE: a space in the type-ahead keystrokes will not be accepted and will clear the entered next and move to the next selection.

This was corrected in a newer revision.

NOTE: input area is the length of the longest item in the list.

IMPORTANT: If you SELECTBOX and Input Text and Prompt is positioned in such a way that one may overlap the other, you may experience screen presentation issues. Make sure they do not overlap.

PFSELECTBOXCASE (ver 5.8.02)

This sets a global method for SELECTBOX()

This variable can be set in the config file to determine the default case value so that it does not need to be programmed on the command line.

Values for case are:

0 = case insensitive (default)

1 = case sensitive

2 = 'filename' sensitive - that the input is case-sensitive based on the O/S filename case sensitivity. That is, Windows is case-insensitive, and Unix/Linux are case-sensitive. This is useful when being used to select a filename from a list.

SET

Syntax:

```
SET array  
SET arrayname,value,start,len  
SET arrayname,value,start  
SET arrayname,value
```

"arrayname" is the name of the array.
"value" is the value to which each array element is set.
"start" is the starting subscript to use for the "setting".
"len" is the number of entries to set from the starting element.

Version Ref: 4.5

Description:

Fills an array with a specified value.

If "len" is not specified, all elements through the end of the array are set.
If "start" is not specified, the entire array is set.

NOTE: The given value must be of an edit type that is compatible with the array elements.

SHOW

Syntax:

Then: SHOW exp

Version Ref: 3.x

Description:

SHOW displays an expression on the screen. It will show this expression centered on line 23 of the screen.

Examples:

```
If:
Then: show "This invoice is over customer's credit limit."
```

will display as

```
      This invoice is over the customer's credit limit.
```

You can clear a message put up with SHOW by SHOWing nothing.

```
@mf2 If: 2 gt 19
      Then: show "This invoice is over customer's credit limit."
      If:
      Then: end
@mf3 If: 3 eq "Y"
      Then: show "" ; end
```

Using SHOW "" will always clear line 23 of the screen, regardless of what is there or how it got there.

To show literals, they must be enclosed in quotes.

Prompt for user acknowledgment

The user can be automatically prompted for acknowledgment of a SHOW message, by making use of a special function of the SHOW statement. If the first character of a SHOW expression is an "@", the expression will be shown centered on line 23 as always, and the prompt "Press ENTER to continue." will be centered on line 24 below it. The user must press ENTER to clear the SHOW from the screen.

```
If:
Then: show "@This invoice is over customer's credit limit."; end
```

The above code will display as:

```
      This invoice is over the customer's credit limit.
      Press ENTER To Continue
```

IMPORTANT: Remember, the SHOW message is an expression. You can build it out of strings, fields and expressions. A simple example is as follows:

```
If:
Then: show "@Invoice over customer's credit limit of $" {14};end
```

where field 14 holds the credit limit. The following message is displayed.

```
      This invoice is over the customer's credit limit of $5000
      Press ENTER To Continue
```

NOTE: Using SHOW "@" by itself will put up the prompt "Press ENTER To Continue" centered on line 24 by itself with nothing on line 23, and wait for the user to comply.

SHOW with Screen Positioning

Syntax:

Then: SHOW(r,c,exp)

"r", "c" and "exp" are all expressions.

Displays the expression at position (r,c).

IMPORTANT: Same as SHOW command, but does not allow the use of the "@" to prompt for user acknowledgment. If the "@" is included along with row and column positioning, it will be ignored and the user will not be prompted for acknowledgment.

SHOW Codes

You can insert into a SHOW expression the codes for reverse video. At runtime, filePro will show these parts of the SHOW expression with this attribute (usually this means highlighting the area).

You can insert into a SHOW expression the code for a key's function, rather than the key name itself. At runtime, filePro will display the appropriate key label for the terminal type or computer on which you are running.

You can insert codes for the keypad graphics characters. At runtime, filePro will display the characters.

You can insert a hexadecimal code for any characters in a computer's character set. At runtime, filePro will display the characters.

(DOS only) You can insert codes to set and change background/foreground colors and intensity, to make your messages more attractive and conspicuous.

If:

```
Then: show "Are you\r SURE \r you want to delete this? "
```

The above code will display the message with the word SURE in reverse video.

NOTE: The SHOW Codes can also be used on HELP screens.

SHOW POPUP

Syntax:

Then: SHOW POPUP (row,col,popupnum) exp
Then: SHOW POPUP (row,col) exp
Then: SHOW POPUP exp

"r","c" are the screen location of the upper-left corner of the popup window.

"popupnum" is the popup window number; if popupnum is not specified, 1 is used. "popupnum" is an expression.

"exp" is the message to display.

NOTE: Popups that overlap another popup must be taken down in the opposite order in which they were displayed. "r" and "c" may each be negative to specify centering in that direction. If not specified, the popup is centered both vertically and horizontally.

Version Ref: 4.5

Description:

SHOW POPUP displays an expression in a popup window. These windows can be numbered.

SHOW POPUP windows must be cleared with the CLEARS command.

CLEAR

Syntax:

Then: CLEAR popupnum
Then: CLEAR

"popupnum" is the popup screen number as specified with SHOW POPUP. If not specified, ALL popups are cleared.

NOTE: If you have overlapping popup windows, you must clear them in the opposite order they were created.

Description:

CLEAR takes down a SHOW POPUP window. If no popup is on the screen, it does nothing. CLEAR can take down specific numbered SHOW POPUP windows. (Do this in the reverse order they were put up.)

Examples:

If:
Then: lookup cust k=1 i=a -nx
If: not cust
Then: showpopup ("7","9") "Customer not on file";clears;restart

SHOWRAW

Syntax:

Then: `SHOW RAW exp`

Where "exp" is an expression containing the message to display.

Version Ref: 5.0.6

Description:

Sends the text with no interpretation or formatting by filepro. This allows easier access to the filePro SHOW function when using terminal emulator features as provided by FACETWIN since the sequences sent are not manipulated by filePro.

Example:

```
SHOW RAW chr("27") & "[2]start winword " & filename & "" & CHR("13")
```

The above would send an escape sequence containing text ending with a carriage return to the screen.

SHOWCTR

Syntax:

Then: showctr(row) exp

Row is the row on which to display the screen message.
Both row and expression are expressions.
A row **MUST** be specified.

Version Ref: 4.5

Description:

SHOWCTR displays a message on the screen, centering it on the specified row.

Examples:

Display the default filePro prompts.

```
Then: showctr("24") "\r D\r-Delete, \r H\r-Hardcopy,  
\r U\r- Update, \r X\r-Exit, \r F\r-Print Form,  
\r B\r-Browse"
```

Display any centered prompt.

```
Then: showctr("23") "Press \r K4 \r to save, \r KY\r to cancel."
```

Display centered message on variable line number.

```
If: bb eq "fullscreen"  
Then: aa(2,.0)="23" ; showctr(aa) "This is a full screen view."
```

SHOWTOCOL

Syntax:

Then: SHOWTOCOL(row,col) exp

"row and col" are the ending screen location

"exp" is the message to display

"exp" is an expression

Version Ref: 4.5

Description:

SHOWTOCOL displays a message on the screen, specifying the ending (rightmost) column of the message.

Examples:

IF:

Then: showtocol("19","70") "CREDIT IS VERIFIED"

SIGN()

Syntax:

Then: $xx = \text{SIGN}(\text{num_expr})$

num_expr is the number from which to take the sign.

Return value is -1 if the number is negative, 0 if zero, and 1 if positive.

Version Ref: 4.5

Description:

Performs the signum function, returning the sign of the input.

Examples:

SIN()

Syntax:

Then: result = SIN(angle)

Version Ref: 4.8

Description:

Angles given in radians.

Trig Function to returns the SINE of an angle.

Example:

Then: result = sin(angle)

SINH()

Syntax:

Then: result = SINH(angle)

Version Ref: 5.0

Description:

Trig Function to returns the SINEH of an angle. The result is given in radians.

Example:

Then: result = SINH(45)

SKIP**Syntax:**

Then: SKIP

Version Ref: 3.x

Description:

Skip to next field in the cursor path; used with @wef only.

SLEEP

Syntax:

Then: SLEEP time

"time" is the time in milliseconds to stop processing.

Note: 1 second = 1000 milliseconds, 1 minute = 60000 milliseconds

Version Ref: 4.5

Description:

Stops processing for a specified amount of time.

Examples:

@keyT If:

Then: show"@The date is" < @td

Then: sleep "2000";end

SORT

Syntax:

Then: sortN = m
Then: sortN(l,t,o) = m
Then: sortN(l,t,o) = exp

where: "N" is a number from 1 to 8 (to match the break positions on the output format.)

"m" is the field (real, dummy, or lookup) to sort on

"l" is the field length

"t" is the field's edit type

"o" is either A for ascending or D for descending (A is the default)

"exp" can be any expression

Without stated attributes, the dummy sort key takes on the attributes of the result of the expression. In other words, if the result of the expression is a 12-character number, the sort key is a 12-character numeric field. Just make sure that the edit types are compatible.

SORT can only be used on sort/selection processing tables.

The SORT command cannot be used to override the number of sort levels, only the sort keys.

SORT can only be used to override a sort key that HAS a sort key that has already been defined on the output format.

Version Ref: 3.x

Description:

Sorts output during SORT/SELECTION processing phase. (-v processing table). SORT lets you create customized and conditional sort keys from the sort/selection processing table. SORT will override sort keys (but not subtotal or total breaks) defined for an output format through Define Output..

Examples:

To sort in ascending order, based on a number that has a related name in another file, you can do a lookup to the other file and sort by that name. First, do the lookup on the other file (cust), and then specify the sort as follows:

Then: sort1(5,*,a)=cust(1)

SORTARRAY

Version Ref: 6.0.00

For user defined arrays only.

`xx = SORTARRAY(dat,"0")`

where ("0"|"A") is ascending, ("1"|"D") is descending.

Works with **ONLY** dummy field types. It does not work with mapped fields to real fields.

Returns -1 on error.

SORTINFO()

Syntax:

Then: `XX=`**SORTINFO**

Returns the sort information for the currently running output format.

Then: `XX=`**SORTINFO**(filename,formatname[,sortlevel])

The return value is in the format for all possible choices: field,length,ascend/decend,[subtotal]: 5, 8,a,y:2, 15,a,y

Description:

SORTINFO() returns the sort information for the specified output. It also includes the subtotal break information.

SOUNDEX()

Syntax:

Then: a = SOUNDEX(exp)

Version Ref: 4.1

Description:

The Soundex function returns a 4 character soundex code for a string expression or variable. The soundex code is comprised of a character (A-Z) followed by a 3 digit number. i.e., K152.

Key Letters and Equivalents

Code Returned

b, p, f, v	1
c, s, k, g, j, q, x, z	2
d, t	3
l	4
m, n	5
r	6

The letters a, e, i, o, u, y, w, and h are not coded. A name yielding no code numbers such as, "Lee," would return L000; one yielding only one code number would have two zeroes added, as "Kuhne," coded as K500; and one yielding two code numbers would have one zero added, as "Ebell," coded as E140. No more than three digits are used, so the name "Ebelson" would return E142, not E1425. Soundex was designed to group surnames that sound similar.

Examples:

- Witherspoon or Weathers -> W362
- Braddie or Brody -> B630
- Kragged or Kracht -> K623

Then: a = soundex(exp) ; b= soundex("Smith")

SPELLCHECK

[Spell Check - Using Processing](#)

[Spell Check - Merro Fields](#)

The 64-bit versions of filePro now use the industry-standard "hunspell" spell-check library. As noted by the hunspell website:

"Hunspell is the spell checker of LibreOffice, OpenOffice.org, Mozilla Firefox 3 & Thunderbird, Google Chrome, and it is also used by proprietary software packages, like Mac OS X, InDesign, memoQ, Opera and SDL Trados."

<http://hunspell.sourceforge.net/>

EXPLODE/ SPLIT (Ver. 6.0.02)

EXPLODE or SPLIT import data

Usage:

```
sz=SPLIT(array, string, delimiter)
```

array is the array that the data will be placed into

string is the data to split

delimiter is the sequence of characters to split on

NOTE: The array being used must have the size defined for its elements and cannot be an alias.

SQRT()

Syntax:

Then: $a = \text{SQRT}(n)$

Version Ref: 3.x

Description:

Returns the square root of "n".

Examples:

Then: $nn = "456"$

Then: show "@The square root of " nn <"is" < $\text{sqrt}(nn)$

The square root of variable "nn" is "21.3542".

STACKTRACE()

Version Ref: 6.1 (USP6.1.01)

Syntax:

```
n = STACKTRACE(array)
```

Fill an array with a processing trace, listing the current and past processing tables and their line numbers to the current line being executed. This will show lines "jumped" from gosubs and follow calls and functions.

Returns the number of elements that could fit into the array.

STATUS()

Syntax:

```
Then: handle = NEW STATUS()
Then: STATUS handle GET
Then: handle SET
```

Version Ref: 5.0.6

Description:

The STATUS object allows you to save/return the status of break, cursor, video, escape, and background. Allows subroutines to enable or disable these items, and then restore them to their original state.

Example:

```
MySub
If:
Then: declare save_status(5,0,g)
If: save_status = ""
Then: save_status = new status()
If:
Then: status save_status get ' save current status
If:
Then: break off; cursor on
'... more processing here ...
If:
Then: status save_status set ' restore original status
If:
Then: return
```

STRING FUNCTIONS

All functions that take a position default to the first character in a field if not specified.

All "is" functions return "1" for true and "0" for false.

x=isalpha(fld [, pos])

Is the character at the position given a letter?

x=isdigit(fld [, pos])

Is the character at the position given a number?

x=isalnum(fld [, pos])

Is the character at the position given a letter or number?

x=isspace(fld [, pos])

Is the character at the position given a whitespace character?

' ', '\t', '\n', '\r', '\v', '\f'

x=islower(fld [, pos])

Is the character at the position given lowercase?

x=isupper(fld [, pos])

Is the character at the position given uppercase?

x=isxdigit(fld [, pos])

Is the character at the position given a hexadecimal character?

'0'-'9', 'A'-'F'

x=iscntrl(fld [, pos])

Is the character at the position given a control character?

ASCII codes 0x00 (nul) - 0x1f (US), and 0x7f (del)

x=isprint(fld [, pos])

Is the character at the position given a printable character?

ASCII codes greater than 0x1f (US) not including 0x7f (del)

x=ispunct(fld [, pos])

Is the character at the position given a punctuation character?

x=isgraph(fld [, pos])

Is the character at the position given a character with a graphical representation?

The characters with graphical representation are all those characters than can be printed (as determined by isprint) except for space.

x=tolower(fld [, pos])

Return the character at the position given as a lowercase character.

x=toupper(fld [, pos])

Return the character at the position given as an uppercase character.

str=strtolower(fld)

Return the entire string converted to lowercase.

str=strtoupper(fld)

Return the entire string converted to uppercase.

aa=ltrim(fld)

left trim

aa=rtrim(fld)

right trim

aa=trim(fld)

trim both left and right

STRtok()

Syntax:

Then: n=STRtok(string, characters[, startpos])

Then: l=strtok(aa, "1234567890,/[]{}!@#\$/%^&*+=_<>?", 1)"

Note that this is different from the INSTR() function in that it will return the first occurrence of ANY of the specified characters, not a "string" of characters.

Version Ref: 4.8

Description:

STRtok() returns the location of the first occurrence of any of the specified characters found in a second string, starting at an optional position in that string. The default starting position is 1.

SWITCHTO

Syntax:

Then: SWITCHTO name

"name" is the name of screen to switch to.

Version Ref: 4.5

Description:

Switch to a different screen, without returning to the original screen upon ending processing (which is the default operation of the SCREEN command which also switches screens). Normally, when INPUT processing ends you are returned to the SCREEN you were on when processing started. The SWITCHTO displays a specific screen, and sets it to be the screen to be returned to, instead of the original screen.

Examples:

```
@keyP If:  
Then: switchto "pay";end
```

Allow only certain users access to screen 9.

```
@key9 If: "*root*kenb*ronk*lauraw*" co ("*" & @id & "*")  
Then: switchto "9" ; end  
If:  
Then: errorbox "You are not authorized for screen 9." ; end
```

SYNC

Syntax:

Then: SYNC lookupname
Then: SYNC -
Then: SYNC

"lookupname" is the name of the file to sync.
use - to represent the current (main) file.
if no filename specified, all files are synched

NOTE: Many computer systems have a write behind disk cache, where information is not immediately written to the disk, in order to help increase disk performance. Should the system crash between the time filePro writes a record and the system writes the cache, that formation is lost.

Under MS-DOS, a file's directory entry is not updated until the file is closed. If you are adding new records beyond the current end of file and do not exit filePro properly, all new records will be lost. The SYNC command forces the update of the file's directory entry.

The SYNC command allows you to have some control over when the information is written to disk. Note that indiscriminate use of the SYNC command can degrade overall system performance.

Version Ref: 4.5

Description:

Flushes any disk writes pending on a file.

Examples:

Create an audit trail record, and force it to be written from cache, and the directory entry updated:

Then: lookup audit = filename r=free -ex
Then: copy audit ; write audit ; sync audit

Even though filePro will write the record with the WRITE command, it is possible that the operating system will cache the write. Also, MS-DOS systems will not update the files directory entry to reflect the possible size change from the new record. Should the system crash before the cache is written, or filePro exits, these changes can be lost.

SYSTEM

Syntax:

Then: SYSTEM "command"

Then: SYSTEM NOREDRAW "command"

"command" is an expression.

"command" can be any Operating System command accessible from the operating system command line (DOS), or shell (Unix).

If NOREDRAW is omitted, the screen will be redrawn after the command is executed.

Version Ref: 4.1 (for NOREDRAW)

Description:

SYSTEM lets the program execute operating-system commands while processing records. When the SYSTEM processing is finished, the user is returned to the next statement after the SYSTEM command.

Examples:

Take the user to another filePro file in Inquire, Update & Add without forcing them to completely leave the current file. When they leave the other file, they are returned back to the end statement.

@keyG If:

Then: system "/appl/fp/rclerk otherfile -s1"

Then: end

TAN()

Syntax:

Then: result = TAN(angle)

Version Ref: 4.8

Description:

Trig Function to provide angles given in radians.

Returns the TANGENT of an angle.

TANH()

Syntax:

Then: result = TANH(angle)

Version Ref: 5.0

Description:

Hyperbolic trig Function to return the tangent of an angle in radians.

Example:

Then: result = TANH(angle)

TELL()

Syntax:

Then: aa = TELL(handle)

"handle" is the file handle returned by OPEN() or CREATE().

Return value

Returns the current absolute position in the file.

Version Ref: 4.5

Description:

Gets the current location in the file.

TOHTML()

Syntax:

Then: xx = TOHTML(yy)

Version Ref: 4.8 (not included in filePro Lite)

Description:

TOHTML() converts special HTML characters '<', '>', '"', and '&' to their HTML equivalents: '<', '>', '"', and '&'. This permits you to include these characters within the text of an HTML document.

Example:

You want to display e-mail addresses. If you had

```
addr = "Bill Randall <wrandall@fileproplus.com>"
```

...

```
HTML :TX addr
```

the e-mail address would not appear, as it would be interpreted as an HTML tag. You would need to do:

```
addr = "Bill Randall <wrandall@fileproplus.com>"
```

...

```
HTML :TX TOHTML(addr)
```

which would generate:

```
Bill Randall &lt;wrandall@fileproplus.com&gt; ;
```

This would be interpreted and displayed as expected.

(The function can be used anywhere in processing that an expression can be used, not just in the HTML command itself.)

TOT()

Syntax:

Then: $a = \text{TOT}(n)$

Gets a total or subtotal on field "n".

Version Ref: 3.x.

Description:

A system function that tells filePro Plus to keep a running total of a field or fields.

There is more than one way to get cumulative totals. One is to put the total-field indicator on a total or subtotal line of a report. Another is:

Then: $a = a + n$

However, neither is as flexible as the TOT function. In the first case, TOT lets you total conditionally; "=" doesn't. In the second case, TOT lets you write statements like:

Then: $tt = \text{tot}(22) * \text{tot}(23)$

or:

Then: $tt = \text{tot}(22 * 23)$

Note that the results of the two equations are quite different from each other. In record 1, field 22 contains a value of 10 and field 23, a value of 12. In record 2, field 22 contains a value of 3 and field 23, a value of 8. The result of equation 1 is 260: $(10 + 3) * (12 + 8)$. The result of equation 2 is 144: $(10 * 12) + (3 * 8)$.

Examples:

You want to print a report that lists average sales per sales representative, maximum and minimum sales, and at the end of the report, the total number of sales transactions for all representatives.

Field 22 is "Total Sales." The element would be written as follows:

Then: $TS = \text{TOT}(22)$

To print the total, put *TS on a total line of the report format.

TVM_xx

Financial Functions

Time-value-of-money functions have been implemented to calculate n/i/pv/pmt/fv values, given the other 4.

Examples:

```
N = TVM_N(i,pv,pmt,fv)
I = TVM_I(n,pv,pmt,fv)
PV = TVM_PV(n,i,pmt,fv)
PMT = TVM_PMT(n,i,pv,fv)
FV = TVM_FV(n,i,pv,pmt)
```

where:

```
N = number of payments
I = Interest rate
PV = present value
PMT = amount of payment
FV = future value
```

Notes:

The formula used is:

$$100 \cdot (1 - \text{sppv}) \cdot \text{pmt} \cdot \frac{1}{i} + \text{pv} = -\text{fv} \cdot \text{sppv}$$

where "sppv" is the single payment present value:

$$\text{sppv} = \frac{1 - (1 + i)^{-n}}{i}$$

It is assumed that payments are made at the end of each period.

UPDATE

Syntax:

Then: update

Note: Available on automatic processing tables only.

Version Ref: 3.x

Description:

The UPDATE command puts the user into update mode automatically and/or conditionally from automatic processing. In effect, it presses "U" for the user.

Examples:

Since automatic processing happens at least twice, it is important to test whether or not the user is already in update mode to prevent "updating" again and again and again... Use the following combination of commands to prevent the user from looping through the automatic processing forever.

On the automatic processing table:

```
If: q eq ""  
Then: q(1,,g)="1"; update  
If: q eq "2"  
Then: q=""
```

Put the following at the bottom of the INPUT processing table:

```
Then: q="2"
```

USER

USER timeout (ver 5.8.02)

Syntax:

Then: USER label = progname

Version Ref: 3.x NIX only (not included in filePro Lite)

Version Ref: 5.7.04 Windows supports this command (not included in filePro Lite)

Version Ref: 5.8.02 Added timeout parameter (not included in filePro Lite)

Description:

Send/receive data to/from user supplied program.

Examples:

As in lookups, there are two steps. First you identify the user program name:

Then: user prgname

Then: user prgname=pathname

where:

"prgname" is the name you assign to the external program. You must use the complete pathname if the user program is not in the current directory. Next, you set up field assignments. To write to a user program, use this syntax:

Then: prgname=m

where "prgname" is the external program and "m" is a *filePro Plus* field or expression. To write a number of arguments, repeat the statement: prgname=m; prgname=n; prgname=o; and so on.

To read from a user program:

Then: m=prgname

where:

"m" is a *filePro Plus* field

"prgname" is the external program

To read in a number of assignments, repeat the statement: m=prgname: n=prgname: o=prgname; and so on.

To test for an end-of-file or program termination, put the user program name on the condition line.

You can now also specify a timeout (in milliseconds):

user alias = (command, timeout)

When reading from the user command, if nothing is received within that many milliseconds, the read will return a null string -- "" -- and you can put "timeout(alias)" on an "if" line to see if the null string was because of a timeout.

When you make an assignment to a user program, *filePro Plus* writes to the user program's standard input. The field ent will be terminated with a "\n" (new-line character). When you read in values or otherwise reference the user program, *filePro Plus* will read from the standard output of the user program (up to a new-line character).

When you read in values or otherwise reference the user program, *filePro Plus* will read from the standard output of the user program (up to a new-line character).

IMPORTANT: When reading from a user program, *filePro Plus* executes the program only once, not over and over. Therefore, make sure the user program itself loops until it reaches an end-of-file. If you wish to run the same user command more than once you must close it before rerunning it.

close 'prgname'

Also, when using the user command, do not add any other functions to the same line as the user command as it will render the command invalid.

User Defined Functions

User defined functions - Forward declare functions to be used:
(function|func) [file.]name([dim|var] var1, [dim|var] var2, ...)

e.g.

```
function fplib.showlock(var pid)
function fplib.log(file, line, what)
function somefunc(dim myarray)
```

Call a function:

```
[x]=[file.]name(var1, var2, ...)
```

Return a value from a function:

```
return(value)
```

Can pass fields: real, dummy, longvar

Can pass arrays: Alias and system arrays are copied to a non-aliased array. Non-aliased arrays are passed by reference.

Function names must be at least 3 characters in length.

Functions cannot modify values outside of its scope.

Functions do not call automatic processing.

Functions cannot modify real fields.

Functions cannot be called unless they are declared.

Functions can pass values by reference (changes made to the value will carry back out of the function, only to arrays).

Functions can optionally return a value.

Parameter names must be at least 3 characters in length.

Parameters will be passed to the function using the name they were defined with in the declaration statement.

Environment variables:

PPFUNCDBG=(ON|OFF), default OFF.

If ON the debugger will be allowed to continue into the function call. If OFF the debugger will skip over user defined functions.

NOTE: Debug statements inside of functions will still be able to

be activated. If debug is set inside of a function, it will

continue even after the function is left.

Example:

Processing table for fibonacci:

```
If:          ' Declare for future use
Then: function fibonacci(nval)
If:          ' Get the parameter
Then: declare extern nval
If: nval le "1"      ' Return the result
Then: return(nval)
If:          ' Return the result
Then: return(fibonacci(nval-"1")+fibonacci(nval-"2"))
```

Usage:

```
If:          ' Declare for future use
Then: function fibonacci(nval)
If:          ' Call the function
Then: n=fibonacci("9")
If:          ' Display the result
Then: msgbox ""{n      ' Prints "34"
```

VIDEO

Syntax:

```
Then: VIDEO off
Then: VIDEO on
Then: VIDEO sync
```

Version Ref: 4.5

Description:

Turns video updates on and off, or synchronizes screen to most current appearance without turning video on or off.

VIDEO SYNC forces the screen to be updated to the current state, without turning video back on.

If video updating is turned off, it is automatically turned back on if filePro needs to get a keystroke, and no PUSHKEYs are pending. This prevents the system from appearing frozen when it is in fact waiting for user input.

IMPORTANT: There are instances when you might need VIDEO SYNC or else you will not "see" what you expect to see. Because of the internal workings of filePro's "virtual screen", a change may happen to the screen that the process does not show. Consider the following code:

```
Then: end
@keyT If:
Then:
Then: showpopup "the time is"<@tm
If: @os eq "dos"
Then: sleep "3000"
If: @os eq "unix"
Then: sleep "3"
Then: clears ; end
```

The above code will not show anything on the screen! This is because filePro works internally, updating and displaying a virtual screen. If a change and a clear of that change are put up in sequence with nothing else happening, by the time the screen gets displayed, it is already in its cleared state. The code can be fixed by adding the VIDEO SYNC command to line 3 as follows:

```
Then: showpopup "the time is"<@tm ; video sync
```

This tells filePro to refresh all changes to the screen at that point, regardless of what follows. A blank screen is a fairly rare instance in filePro, but it is distressing not to know the explanation for this behavior when you do come across it.

HINT: When used with PUSHKEY, turning off the video updates with VIDEO OFF can produce very clean operations. Use VIDEO ON to turn screen updating back on when the pushkey is done.

Examples :

Use index A to find the name "Brody".

```
Then: video off ; pushkey "X4ABrody[ENTR]" ; end
```

Automatically perform a long sequence of commands that normally update the display, and let the user know whats going on.

```
Then: showpopup "Performing calculations..." ; video off ; clears
```

...put first part of calculations here...

```
Then: showpopup "Done first part. Starting part 2..." ;
video sync ; clears
```

... put next part here...

```
Then: video on ; msgbox "Done."
```

TIP on VIDEO SYNC

Use VIDEO SYNC to force the screen to be immediately drawn after showing the message. Of course, this may slow things down if it draws many messages. (Note how filePro updates the screen once a second, rather than every X records.)

Perhaps a subroutine?

```
VidSync
If:
Then: Declare LastTime(8,hms,g)
If: LastTime ne @tm
Then: video sync ; LastTime = @tm
If:
Then: return
```

Then simply "gosub VidSync" to update the screen. If more than one second has elapsed since the last call (or if this is the first call), then the screen will be updated.

WAITKEY

Syntax:

Then: a= WAITKEY Wait for next keystroke. Used on "then" lines.
If: WAITKEY = "k" Same as above, but used on "if" lines.

Version Ref: 3.x

Description:

Waits for the next keystroke.

Examples:

WAITKEY used on the "then line:

WAITKEY stores the next key pressed in the prescribed dummy variable. You can test this variable and act accordingly upon its being one value or another. If you want to intercept Special Keys (from the Key Label list) you must designate a dummy variable of at least 4 characters in length, and an edit type capable of storing text.

```
wrcode If:
  Then: a(4)=waitkey
  If: a eq "Z"
  Then: show"@The user pressed a Z" ; goto doZ
  If: a eq "Y"
  Then: show"@The user pressed a Y" ; goto doY
  If: a eq "SAVE"
  Then: show"@The user pressed ESCAPE." ; ESCAPE
  If: a eq "CRUP"
  Then: show"@The user pressed the UParrow';pushkey "[CRUP]";end
  Then: goto wrcode
```

The code above shows how WAITKEY functions. If the user does not press any of "Z", "Y", "SAVE" or "CRUP" it will keep waiting until one of these keys is pressed. This is a powerful mechanism for forcing the user to press only those keys you want pressed.

WAITKEY use on the "if" line:

On the "if" lines, WAITKEY allows you to wait for particular characters and act upon receiving them from the user.

```
If: waitkey="z"
Then: show"@The user pressed a Z" ; end
Then: show"@The user pressed something other than a Z" ; end
```

Capturing special keys

If waitkey contains a null value, the key pressed was a special key, not a character key, and will be contained in the system-maintained field for special keys, @SK. The value will be a four-letter code from the Special Key Codes.

Here is an example in which the nature of the key captured, character or special, determines which routine is called, "gotchar" or "gotspec":

```
Then: aa=waitkey
If: aa ne ""
Then: goto gotchar
If: @sk="CDWN"
Then: goto gotspec
```

IMPORTANT: WAITKEY cannot detect or respond to the BREAK key.

WOM()

Syntax:

Then: `xx = WOM(date_expr)`

"date_expr" is the date to use.

"weeks" is always considered Sunday through Saturday.

Version Ref: 4.5

Examples:

If the year starts on Friday, then the first week is January 1st and 2nd, and January 3rd starts the second week.

WOQ()

Syntax:

Then: $xx = \text{WOQ}(\text{date_expr})$

"date_expr" is the date to use.

"weeks" is always considered Sunday through Saturday.

Version Ref: 4.5

Examples:

If the year starts on Friday, then the first week is January 1st and 2nd, and January 3rd starts the second week.

WOY()

Syntax:

Then: $xx = \text{WOY}(\text{date_expr})$ week of year

"date_expr" is the date to use.

"weeks" is always considered Sunday through Saturday.

Version Ref: 4.5

Examples:

If the year starts on Friday, then the first week is January 1st and 2nd, and January 3rd starts the second week.

FOR

Version Ref: 6.1 (USP6.1.01)

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

- 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.

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.

Example - For Loop

Processing:

```
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
```

WHILE

Version Ref: 6.1 (USP6.1.01)

WHILE cnd DO label

- 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.

Example - While Loop

Processing:

```
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|LOOP UNTIL

Version Ref: 6.1 (USP6.1.01)

LOOP label WHILE cnd

LOOP label UNTIL cnd

- 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.

Example - Loop While

Processing:

```
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
```

WORDWRAP()

Syntax:

Then: `xx = WORDWRAP(text, width [,options])`

where "text" is the text to be word-wrapped, width is the number of characters to be included per line and options are as follows.

- 0 - Return the word-wrapped text as-is, excluding any hard returns. (default)
- 1 - Return the text with trailing spaces removed.
- 2 - Return the text as-is, including any hard return at the end.

Return value

Returns the number of lines that will result for the given field or expression for the specified width per line.

Notes: Also see [@WORDWRAP\[\]](#)

Version Ref: 5.6

Example:

```
Then: dim wraptext(999)
Then: xx = WORDWRAP(9,"70","1"); c(3,.0)="1"

Loop
If: c lt xx
Then: wraptext(c) = @WORDWRAP(c); c=c+"1"; goto Loop
Then: end
```


WRITE

Syntax:

```
Then: WRITE  
Then: WRITE filename
```

WRITE, by itself, without a filename argument, forces all open files be updated to their in-memory versions.

WRITE filename, instructs that just the designated file be updated to its in-memory version.

Version Ref: 4.5 (not included in filePro Lite)

IMPORTANT: On multi-user systems, "WRITE filename" also causes the lock to be removed from "lookup files" which have had a lock placed on them by the -p lookup option. In other words, on multi-user systems, it is possible for multiple users to view a record at the same time. However, only one user at a time can UPDATE a record. This is because while the user is updating the record, it is locked by filePro. Other users may view it, but may not write to that record until the locking user unlocks it. This prevents records from getting corrupted by two people changing data on them at the same time. If the record is retrieved by a lookup and the -p flag is used on the lookup, then no one else can update that record until the lock is removed. This is done with the "WRITE filename" command. You should write your code so as to retrieve records, update them, and release them as soon as possible. This keeps your databases open and easily used by everyone.

```
getnum If:  
Then: lookup ctl=control r=r -np  
If: not ctl  
Then: show "@problem, no control record" ; exit  
Then: l=ctl(4) ; ctl(4)=ctl(4)+"1" ; write ctl ; return
```

This code gets a unique number from a control file. Then it increments the value in the control file so that the next user coming for a unique number will get the next higher number. However, because the -p flag is used on the lookup line, this particular record is "locked" and no other process can retrieve a unique number until it is unlocked. This prevents two or more users from getting the same unique number! That is why there is nothing else between getting the unique number, incrementing the control number by "1", and then running the WRITE command against the control file. The WRITE command not only hands all the fields of the record to the O/S for storage on disk, but it ALSO unlocks the file at this particular record. Anyone else can now obtain the next available unique number, it will be one higher than the last number retrieved, and the next retrieval will increment it for the next user after that and so on. The idea to grasp here is that the locked record is not held in a locked state for any longer than absolutely necessary. Get the unique number, increment the control number, write the file to ensure that the change is written to disk before anyone else can retrieve this record, and obtain the next unique number.

NOTE: The CLOSE filename statement will also perform an "unlock" of a designated file. However, this will also close the file, and the next user accessing this file to obtain anything will have to wait a little longer while the O/S "opens" the file again. In a case where many hits will be made on a file in a short period of time, it is best to simply WRITE the file each time it is updated to provide faster operations.

HINT: If you are working on a new record, the system-maintained fields for that record will be empty until an automatic WRITE happens at the end of processing, or until you issue a WRITE command on the processing table. A good check to see if a record is new is to test whether @cd (creation date) is equal to "" (null). If it is, this record is currently new and has never been written to disk.

Description:

WRITE is a very important command. It has two important functions. The first is to force the memory copies of all the fields in a record to be written out to disk. (Actually they are just handed to the O/S whose job it is to really write them to disk. Another filePro command, SYNC, forces the O/S itself to write all of its buffers to disk. This means things waiting to be written to disk actually get written.) However, for most practical purposes, you can assume that issuing a WRITE is a guarantee that your data will be written to disk. The second function of WRITE is to unlock a specified file or all files if no file is specified.

Technical Notes

WRITE is different from the WRITE() function, which writes regular files opened with the OPEN() or CREATE() functions.

Version 6.0.02

The functions lock or unlocks nbyte bytes of the file specified by handle.

```
x=lock(handle,how[,nbyte])  
handle - an open handle to a file  
how - U|O : unlock bytes  
L|1 : lock bytes  
N|2 : lock bytes non-blocking  
nbyte - How many bytes in the file to lock, if omitted, lock  
the billionth byte in the file (file does not have to be  
that large)
```

```
x=unlock(handle[,nbyte])  
handle - an open handle to a file  
nbyte - How many bytes in the file to unlock, if omitted,  
unlock the billionth byte in the file (file does not  
have to be that large)  
returns "1" on success  
returns negated system error on error
```

WRITE()

Syntax:

```
Then: aa = WRITE(handle,source,length)
Then: aa = WRITE(handle,source)
```

where "handle" is the file handle returned by OPEN() or CREATE(), "source" is the data to write, and "length" is the number of bytes to write. If "length" is not specified, the length of the source field is used.

Return value

The number of bytes written.

Version Ref: 4.5 (not included in filePro Lite)

NOTE: In a text file, it is possible to write more bytes than specified. For a DOS text file a write of 10 bytes with a LF (line feed) character in it will cause 11 bytes to be written as the LF is converted to a CR/LF (carriage return, line feed pair). WRITELINE() is recommended for text files.

The write will be done to the current position in the file. This position can be set by SEEK(), otherwise it will be the byte immediately at the end of the last read/write operation.

WRITELINE()

Syntax:

Then: bytes = WRITELINE(handle,source,length)

Then: bytes = WRITELINE(handle,source)

"handle" is the file handle returned by OPEN() or CREATE().

"source" is the data to write.

"length" is the number of bytes to write.

"bytes" is the number of bytes written, including newline

If length is not specified, the length of the source field is used.

A newline character is appended to the information written.

Return value

The number of bytes written, including the newline.

Note: The file must be opened in text mode. A newline character is appended to the information written.

Version Ref: 4.5 (not included in filePro Lite)

Description:

Writes a line of text to a file.

XLATE()

Syntax:

Then: `xx = XLATE(text_expr,from_expr,to_expr)`

"text_expr" is the string to edit.

"from_expr" is a list of characters to translate from.

"to_expr" is a corresponding list of characters to translate to.

Return value:

A copy of the original string with each character translated as appropriate.

NOTE: If from_expr is longer than to_expr, then any character found in from_expr beyond that length will be deleted from the resulting string.

Version Ref: 4.5

Description:

Translates individual characters in a text string to other characters.

Examples:

Convert backslashes to forward slashes, braces to brackets, and eliminate some punctuation marks.

Then: `newname = XLATE(oldname," \ { } !@#$/%","/[]")`

Eliminate parentheses and dashes from a phone number, leaving just the digits. Then dial the number.

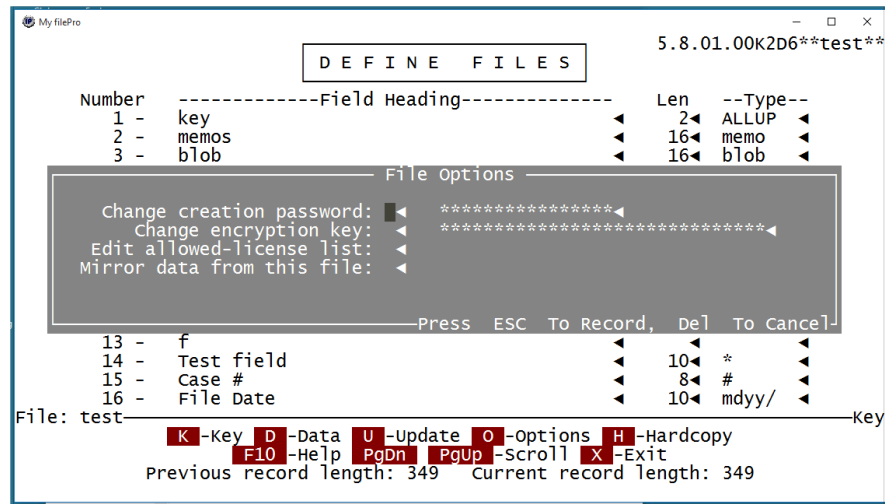
Then: `outs "ATDT" & XLATE(phone,"()-","") & chr("13")`

Note that the file I/O functions cannot be used to access any file with a name formatted as a filePro file. For example, you cannot access a file named "screen.1." "Index.A," or "key."

Advanced Options in Define files - Version 5.7.00

There is a new function in Define Files called O- Options

This pops up additional advanced options to select.



The first option allows you to change the creation password. Press Y and then enter the new creation password. Leaving the field blank will remove an existing creation password.

The second option for file encryption. Press Y and then enter the encryption key. **DO NOT LOOSE** the key as it will be required to access the file and it cannot be retrieved by filePro or fP Technologies. Leaving the field blank will remove an existing encryption key. **DO NOT encrypt an alien or ODBC file.**

The third option allows you to edit the allowed list for this file. If for some reason you wished to move this files data to a new machine and this file was encrypted, you would need to add the necessary information on the other filePro installation in order for it to properly read the encrypted data.

The final option is for mirroring this file to a dual write or secondary location. See [Dual Write/Mirror](#).

Edits Description

The edits feature in filePro allow you to easily validate data as it's entered and can be used to format data during importing from other sources and when posting. Because of the power and ease of programming that filePro edits provide, many programmers routinely use filePro to validate data which is downloaded from mainframe computers and other sources.

Edit Classifications

Edits can be classified and used in various ways but can be generally classified as follows.

System Edits	These edits that are maintained internally to filePro. This type of edit are primarily date edits such as MDY, MDY/, MDYY/, etc. These edits can be used by all filePro files and cannot be changed.
Global Edits	These edits are maintained in a table in "..\fp\lib\edits" by default or as specified by the PFGLOB environment variable. These edits can be used by all filePro files. User edits can be added to the Global Edits table.
Local Edits	These are user edits (also called file edits) that are file specific and maintained in the applicable filePro directory e.g. "..\filepro\file_name\edits" where file_name is the name of the filePro directory. A local edit will override a global edit with the same name.

Edit Syntax

In the following descriptions, "X" and "Y" are edit expressions, "L" is any literal, surrounded by quotes and N is a number.

Expression	Description
(X)	Parentheses may be used to separate expressions as in algebra.
[X]	The expression X is optional.
{ X }	The expression X may occur any number of times, but must occur at least once.
< L >	The literal may appear, but if it doesn't, filePro will add it. Examples: "Y<es>" will accept either "Y" or "Yes" as input and will turn a "Y" into a "Yes". ~"N'_<>" will accept any of the following as input and turn it into "No". N n no NO Nb nO
! L !	The literal must appear, and filePro will delete it.
X Y	Either expression is permitted. Example: "N" "N"! will accept only "N" or "Nb", and will turn a "Nb" into an "N".
X & Y	The data must conform to both expressions.
*	Accept any single character.
\	At beginning of line. Right-justifies the resulting field.
\C	At beginning of line. Right-justifies the resulting field using a fill character as specified by "C".
^	Ignores case differences. Takes effect where it occurs on line.
%	Turns off case conversion. Takes effect where it occurs.
~	Converts data to uppercase. Takes effect where it occurs.
_	Converts data to lowercase. Takes effect where it occurs.
@	Use to identify where to end a field. Example: ["-"] ({N} <>) ((<-><><>@) ("." (N<> @ NN[{N}]))) 2 w would be displayed 2.00 2.1 w would be displayed 2.10 2.12 w would be displayed 2.12 2.123 w would be displayed 2.123

Punctuation Combinations – Punctuation may be combined to form the following functions:

[{ X }] – The expression may occur any number of times, or not at all.

[! L !] – If the literal appears, it will be deleted.

Prompted Edits

filePro Plus lets you add prompts to your edits. When the user moves to a field that uses a prompted edit, the prompt will appear at the bottom of the screen before the user types anything into the field.

Syntax:

name 'prompt' normal edit syntax

where "name" is the name of the edit and "prompt" is the prompt text enclosed in apostrophes (do not use quotation marks).

System and Global Edits List

The following is a list of filePro defined GLOBAL edits.

Edit	Description
*	Accepts any value.
#	Accepts only 0-9, periods, dashes, slashes and blanks; right-justify; (will not sort in numeric order).
\$	Prepends a dollar sign; accept any number of digits and a negative sign in front of the point; show point and two decimal places, right-justify the result; (since it truncates rather than rounds, use only for result fields),n (where n = 0-8 or F); add a comma every 3 digits.
.0	shows integer values only; no decimals.
.n	(where n = 0-8); accept numeric; show n decimal places.
A	accept A-Z, a-z
ALLUP	accept words (usually names) in upper or lowercase, convert contents to uppercase
ALNUM	alphanumeric; restricted to A-Z, a-z, 0-9, and blanks
ASCII	accept the entire range of printable characters
BLOB	this edit is used for storing pictures and objects in the blob file This edit type defaults to the valid length of 16
CHEQUE	convert dollar amounts to words (used for printing checks); suggested length is 75
DMY	DDMMYY format; (6)
DMY/	DD/MM/YY format; (8)
DMYY	DDMMYYYY format; (8)
DMYY/	DD/MM/YYYY format; (10)
F	floating decimal
HM	HH:MM; minimum length of 5 for minutes + n for hours > 99
HMS	HH:MM:SS; minimum length of 8 for minutes + n for hours
LOWUP	change leading letters to caps; leave other caps intact
MDY	MMDDYY format; (6)
MDY/	MM/DD/YY format; (8)
MDYY	MMDDYYYY format; (8)
MDYY/	MM/DD/YYYY format; (10)
MEMO	this edit is used for storing text memos in the blob file This edit type defaults to the valid length of 16
MMM_YY	convert a MDY, MDYY, MDY/, MDYY/ date into MON/YEAR
N	accept 0-9 only
NUM	accept 0-9, supplies a zero if position is left blank
PARNEG	remove minus sign from a negative number and place parentheses around the number
PHONE	the entire number in phone number format; (to include area codes,
RJ	right-justify
RMINUS	move minus sign of negative number to the right side of number
SEX	accept only M, F, m, or f, and change lower to uppercase; (use a length of 1)
SSNUM	Social Security number; accept 0-9; show dashes;(len must be 11)
STATE	accept standard postal abbreviation; (length must be 2)
UNPAREN	Removes parentheses
UNPHONE	remove parentheses, hyphens, and spaces from phone # string
UPLOW	change leading letters to caps, all others lowercase, except Roman numerals & De, Del, La, Li, Lo, Mac, Mc, and San
TIME	HH:MM:SS; length of 8; accepts 00:00:00-23:59:59
YESNO	(use a length of 1)
YMD	YYMMDD format; (6)
YMD/	YY/MM/DD format; (8)
YYMD	YYYYMMDD format; (8)
YYMD/	YYYY/MM/DD format; (10)

ZIP

Accepts 5 or 9 digits with or without dash; show dash in 9-digit zip; (length must be 5 or 10)

ORJ

Right-justify and "0" fill

ENCRYPT / DECRYPT (not included in filePro Lite)

Syntax

result = ENCRYPT(data,method,key [,nonce])

result = DECRYPT(data,method,key [,nonce])

where

data	the data string to be encrypted.
method	the encryption method used e.g. Blowfish, RC2, AES, etc.
key	the character sequence used as the primary base to encrypt or decrypt the "data" field.
nonce	the encryption mode used by filePro. If nonce is not specified, filePro generates one.

Encryption Methods (not included in filePro Lite)

In general the choice of one encryption method over another has more to do with bid requirements or compatibility than it does with the merits of the method.

The encryption methods supplied are:

1. **Blowfish** - developed by Bruce Schneier as a replacement for
2. **Twofish** - an earlier method developed by Bruce Schneier DES - one of the AES competition finalists
3. **AES** - Advanced Encryption Standard - the latest government standard - chosen via an open submission/discussion/analysis process
4. **Rijndael** - the same as AES (rijndael is the method chosen for AES)
5. **DES** - Data Encryption Standard - the previous government standard 3DES - DES with a key size three times as large
6. **Safer+** - another AES submission developed by Cylink Corporation
7. **RC2** - developed by Ron Rivest for RSA Security

Additional References:

Blowfish - <http://www.schneier.com/blowfish.html>

Twofish - <http://www.schneier.com/twofish.html>

Safer+ - <http://csrc.nist.gov/CryptoToolkit/aes/round1/conf1/saferpls-slides.pdf>

AES/Rijndael - <http://csrc.nist.gov/CryptoToolkit/aes/rijndael/>

DES - <http://www.itl.nist.gov/fipspubs/fip46-2.htm>

3DES - <http://csrc.nist.gov/cryptval/des/tripledesval.html>

RC2 - <http://www.rsasecurity.com/rsalabs/node.asp?id=2249>

Encryption Mode (not included in filePro Lite)

The encryption mode used is CTR mode.

The **key** is the character sequence used as the primary base to encrypt or decrypt to desired field. A key is in some ways similar to a pass phrase used to validate a system login. The key should be not easily guessed, it should be protected as you would protect a login pass phrase. The encryption key has a required length (**Block Size**) that differs among encryption methods. Depending on the encryption method used, the key length can either be within a specified range or the key is a fixed length and must be one of a set of valid lengths. The following table provides Block Size and acceptable key ranges values allowed for each supported encryption method.

Method	Block Size	Key Range
Blowfish	8	8 - 56
RC2	8	8 - 128
AES	16	16, 24, or 32
Rijndael	16	16, 24, or 32
Twofish	16	16, 24, or 32
DES	8	7
3DES	8	21
Safer	16	16, 24, or 32

The length of the key must be between the specified values (inclusive) if a range is specified or be one of the specified values if a list of values is specified.

The encryption/decryption routines enforce this key length requirement and will exit with an error if the length of the supplied key is not valid for the requested method. There is no way encrypted data can be recovered if the key is lost - this point can not be emphasized enough.

Nonces are generated by filePro and are created by constraining the output of the Yarrow pseudo-random number generator to printable non-control characters (so they can be easily stored). Entropy is currently added to the Yarrow generator by manipulations of the system clock. The nonce used for encryption must be provided at decryption time but it introduces no cryptographic weakness if the nonce is known (as long as the key is kept secret). It does introduce a cryptographic weakness if the same nonce is reused with the same key.

The "nonce", sometimes called the initialization vector, is used to transform the key into the symmetric key that is actually used to transform the data. While the key must be held secret in order to protect the data the nonce can be revealed without compromising the security of the encrypted data. While the key can and usually will remain constant while encrypting multiple sets of the same type of data, the nonce should change for each separate set of data. Reusing the same nonce with the same key to encrypt multiple pieces of data weakens the encryption and makes it more susceptible to cryptographic analysis and exposure of the encrypted data. The same key and nonce used to encrypt the data must be supplied at the time of decryption in order for the data to be successfully decrypted.

An example of this might be encryption of data in medical records. You might use a nonce built from the patient ID and social security number. Then, a unique nonce could be generated (and regenerated) in processing from data which would not change stored in the record. If some of the nonce data might possibly change it wouldn't be to hard to build a routine to retrieve the encrypted field, decrypt it with the old data, re-encrypt it with the new data, and store it.

The length of the nonce for the encryption method has to match the block size for the encryption method.

For example:

Selecting Blowfish as the method would mean that the nonce would have to be 8 bytes and the key could be any length between 8 bytes and 56 bytes inclusive.

Selecting AES as the method would mean that the nonce would have to be 16 bytes long and the key would have to be 8, 16, or 24 bytes long.

Selecting 3DES as the method would mean that the nonce would have to be 8 bytes and the key would have to be 21 bytes.

Encrypt / Decrypt Example (not included in filePro Lite)

For this example, use "Define Files" to create a simple filePro file with 4 fields as follows.

```
FIELD 1  data 64 *
FIELD 2  method 20 *
FIELD 3  key 24 *
FIELD 4  nonce 24 *
```

Use "Define Processing" to create an "input processing table" with the following lines.

```
@wuk*
ba(64,*) = encrypt(1,2,3,4)
bb(64,*) = decrypt(ba,2,3,4)
display
end
```

Then use "Define Screen" and place fields *1, *2, *3, *4 !ba, and !bb on the screen.

Enter any 4 sets of values for data/method/key/nonce and press [F8] to display the encrypted and decrypted data in fields "ba" and "bb". Try various combinations for each field value e.g. "data", "method", "key", etc. The results of encryption will change with each combination of field values in field "ba" and the decrypted value (displayed in field "bb") should match the value entered in field "data".

Encrypting Fields - Caution (not included in filePro Lite)

Again, carefully think through the process of encrypting your data before applying encryption since you can destroy your data or at least make it difficult to properly access your data.

- Don't apply edits other than "" to stored data fields you encrypt.
- Avoid encrypting fields you need to build indexes on.
- Always make a backup before applying encryption.

Examples:

Unciphered	Ciphered	Edit Type	Edit Mask	Result
123-12-1234	a\$6}ûf23gh	ssnum	nnn-nn-nnnn	(blank)
Jim Smith	D¥\$kbnmbÃÇ	lowup	Aaa Aaaa	jiM SmiTh

In the case of field having the "ssnum" edit type, when encrypting, the ciphered data will not pass "ssn" edit since the edit mask limits the field to numbers and "-" dashes as shown above. The ciphered value "a\$6}ûf23gh" will obviously not pass the "ssnum" edit. When the edit fails, the resulting value is a blank field. Only use the "" edit type for ciphered fields to avoid this.

In the "JimSmith" example, although the result is not as expected, the data is not lost. The low up edit can be applied to convert the "j" back to uppercase. However, if you expect to be able to find the "JimSmith" record by either an index search or by scanning for records, the ciphered field value "D¥\$kbnmbÃÇ" is not very useful. Avoid encrypting fields that you need to search for or having indexes for this reason.

Credits Encryption

Credit to:

Tom St. Denis

tomstdenis@gmail.com

<http://libtomcrypt.org>

Whose encryption library we use to provide encryption within filePro.

He kindly placed the entire library in the public domain.

Color-Values

Screen colors can be set if the system provides for color. The following charts identify the hexadecimal codes which apply for each color. These codes can be used by setting the environment variable with the filePro configuration program or setting the appropriate environment variables. Use the [filePro configuration](#) editor or environment variables to specify your colors.

Normal Color Chart

0x08	0x01	0x02	0x03	0x04	0x05	0x06	0x07
0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f	
0x10	0x11	0x12	0x13	0x14	0x15	0x16	0x17
0x18	0x19	0x1a	0x1b	0x1c	0x1d	0x1e	0x1f
0x20	0x21	0x22	0x23	0x24	0x25	0x26	0x27
0x28	0x29	0x2a	0x2b	0x2c	0x2d	0x2e	0x2f
0x30	0x31	0x32	0x33	0x34	0x35	0x36	0x37
0x38	0x39	0x3a	0x3b	0x3c	0x3d	0x3e	0x3f
0x40	0x41	0x42	0x43	0x44	0x45	0x46	0x47
0x48	0x49	0x4a	0x4b	0x4c	0x4d	0x4e	0x4f
0x50	0x51	0x52	0x53	0x54	0x55	0x56	0x57
0x58	0x59	0x5a	0x5b	0x5c	0x5d	0x5e	0x5f
0x60	0x61	0x62	0x63	0x64	0x65	0x66	0x67
0x68	0x69	0x6a	0x6b	0x6c	0x6d	0x6e	0x6f
0x70	0x71	0x72	0x73	0x74	0x75	0x76	0x77
0x78	0x79	0x7a	0x7b	0x7c	0x7d	0x7e	0x7f

High Intensity Color Chart

0x80	0x81	0x82	0x83	0x84	0x85	0x86	0x87
0x89	0x8a	0x8b	0x8c	0x8d	0x8e	0x8f	
0x90	0x91	0x92	0x93	0x94	0x95	0x96	0x97
0x98	0x99	0x9a	0x9b	0x9c	0x9d	0x9e	0x9f
0xa0	0xa1	0xa2	0xa3	0xa4	0xa5	0xa6	0xa7
0xa8	0xa9	0xaa	0xab	0xac	0xad	0xae	0xaf
0xb0	0xb1	0xb2	0xb3	0xb4	0xb5	0xb6	0xb7
0xb8	0xb9	0xba	0xbb	0xbc	0xbd	0xbe	0xbf
0xc0	0xc1	0xc2	0xc3	0xc4	0xc5	0xc6	0xc7
0xc8	0xc9	0xca	0xcb	0xcc	0xcd	0xce	0xcf
0xd0	0xd1	0xd2	0xd3	0xd4	0xd5	0xd6	0xd7
0xd8	0xd9	0xda	0xdb	0xdc	0xdd	0xde	0xdf
0xe0	0xe1	0xe2	0xe3	0xe4	0xe5	0xe6	0xe7
0xe8	0xe9	0xea	0xeb	0xec	0xed	0xee	0xef
0xf0	0xf1	0xf2	0xf3	0xf4	0xf5	0xf6	0xf7
0xf8	0xf9	0xfa	0xfb	0xfc	0xfd	0xfe	0xff

Configuration Editor

If you want to employ an environment variable for ALL filePro programs and all users, you can put it into the filePro configuration file. This is done by running the configuration editor from the filePro Directory program. This editor allows you to place environment variables, one per line, along with the desired value. (Under Windows there is NO need to use the word "set" in this editor, it is automatically assumed.)

IMPORTANT: Values set inside the editor are overridden by changes made in the user's environment. In other words, if PFTMP is set in the configuration editor and again in the autoexec.bat or a user's personal .bat file (or ".profile" files under Unix), the global value set in the "config editor/file" file will be overridden by the local environment value.

The configuration editor is reached by choosing the filePro Directory from the filePro Plus Main Menu and then pressing the button marked "For Configuration Editor". (Under ANSI normally F6).

There are 3 environment variables, which CAN NOT be set in the configuration editor/file. They are PFPROG, PFDIR and PFDATA. These must be set outside of the file somewhere else in the environment prior to starting filePro.

5.0 Enhancement - Numeric configuration variables can be entered as octal values by prefixing the value with zero instead of "0x" for hexadecimal values.

Environment Variables

In the early days of filePro, the program was known as ProFile. Because of this, all environment variables were first designed as PFxxxx. When the first versions of filePro were released, all PFxxxx environment variables were linked to the same name starting with FP. For compatibility, this convention has been kept up forever. Any variable shown that starts with PF, such as PFDIR, PFDSK, PFNOBOX, etc. can also be used starting with FP, as in, FPDIR, FPDSK and FPNOBOX. You may feel free to use either version of the variable.

This reference shows the syntax for setting environment variables as: variablename=value. On Unix systems, these variables also need to be exported as: variablename=value for them to be recognized by filePro.

The method for setting environment variables on Microsoft platforms is "set variablename=value and they do not have to be exported.

VERY IMPORTANT: All environment variables can be set using the filePro Configuration Editor (Choice ? on the filePro Plus Main Menu), EXCEPT the following three special cases: PFPROG, PFDATA and PFDIR. These MUST be set externally in the environment for filePro to use them.

In Version 5.8.01, PFDIR2 and PFDATA2 were added to support filePro's mirroring (dual write) feature. (not available in Lite)

LOGFILE=filename

Sets filename for LOGTEXT command.

Version Ref: 4.5

PFCALLDBG=OFF

default ON, that can be used to have the debugger ignore calls when turned OFF.

Version Ref: 6.0.00

PFCHECKLOCK=ON

Warning if attempt to modify lookup w/o -p flg

Version Ref: 4.5

PFCHECKLOCKPOPOP

Controls logging of non-protected lookups.

If PFCHECKLOCK=ON and PFCHECKLOCKPOPOP=OFF, then the popup message does not appear, but the error is still logged.

Note: PFCHECKLOCKLOG=filename sets the filename for logging. (Default:Unix=/tmp/fp.log, Windows=/fp.log)

Version ref: 5.0

PFLOGAPPEND=ON

Appends LOGFILE instead of overwriting.

Version Ref: 4.5

Note: Use LOGAPPEND prior to version 5.0.14

PFLOOKUPNOFILE=ON

Allows you to check syntax on a prc table without it checking for valid filenames in any lookups that you have.

Version Ref: 5.0

PFNOTRAP=ON

(Unix only) PFNOTRAP=ON tells filePro not to trap SIGBUS and SIGSEGV errors.

Version Ref: 4.5

PFCALLDBG

Default ON, that can be used to have the debugger ignore calls when turned OFF. Note: DEBUG ON will still enable the debugger in a call.

Version Ref: 6.0.01

PFSECUREDEBUG

PFSECUREDEBUG=ON, default off. Disables !B escape in processing when set to ON. Prevents users from accessing the shell through the debugger.

Version Ref: 6.0.02

PFBIXBLANK=OFF|ON

Controls how filePro treats a null lookup key.

ON=4.1 way, where null=all blanks

OFF=4.5 way, where null key=lowest possible value.

Default is "ON"

Version Ref: 4.8

PFBIXBUILD=2

Tells dxmaint to use 4.1 style sorting during build whenever possible. Use this on HUGE files to improve performance - on smaller files it will actually harm performance. (dxmaint)

Version Ref: 4.8

PFBIXNODESIZE=nnn

Will have dxmaint build indexes with a nodesize of nK bytes ($1 \leq n \leq 63$) rather than the default value calculated by filePro. (dxmaint)

Version Ref: 4.8

PFBRWM=ON

Strip trailing blanks from browse lookup. (*clerk)

Version Ref: 4.5

PFDDEFXMAINT

Set PFDDEFXMAINT to OFF to revert to the old 4.1 index routine in ddefine

Version Ref: 5.8.03

PFF6PROMPT=OLD

(4.8.5K2) PFF6PROMPT=OLD will put back the old behavior of ignoring @WBL processing in determining which F6 prompt to display. (*clerk)

Version Ref: 4.8

PFIXGT=ON

This will allow *clerk to do a next-greater-than if no exact match is found when selecting through Index Selection. (*clerk)

Version Ref: 4.8

PFIXS=ON | OFF

Turns on or off "Index Scan" feature. (*clerk, *report)

Version Ref: 3.0

PFKEEPPIXVAL=OFF

PFKEEPPIXVAL=OFF causes dclerk to clear the index key prompt. Default of ON keeps the previous value entered. (*clerk)

Version Ref: 4.5.8

PFLKNL=NEW

Lookup "-nl" finds the last matching record instead of the first. (*clerk, *report)

Version Ref: 4.5

PFMAXALLOC=nnn

Maximum # of sort buffers allocated when re-building demand indexes (automatic indexes pre 4.5). Default is "16". (dxmaint)

Version Ref: 4.5

Version Ref: 5.8.01 the default was raised to 128

PFMAXASIZE=nnn

Sort buffer size when building demand indexes, (automatic indexes pre 4.5) (dxmaint)

Version Ref: 4.5

Version Ref: 5.8.01 the default was raised to 128,000

PFMAXTEMP=nnnn

Maximum virtual memory size when sorting. Demand indexes (auto pre 4.5) Default "1000". (dxmaint)

Version Ref: 4.5

PFMAXTFIL=nnn

Maximum virtual memory files used for sorting. Demand indexes (automatic pre 4.5) (dxmaint)

Version Ref: 4.5

PFNOIXHIDE=ON

Disables Index Hiding (All indexes are shown including hidden indexes).

Version Ref: 5.0

PFNUMIXBUF=nnn

Sets the number of index blocks to buffer in memory for 4.5-style indexes. (Index blocks are usually 1K each.) Default is 10.

Version Ref: 4.5

PFNUMIXBUILD=nnn

Same as PFNUMIXBUF, but for dxmaint only. Default is 200. Maximum is 32767.

Version Ref: 4.5

PFOLDIX=ON

Builds old style (4.1) indexes. (*clerk, *report, dxmaint)

Version Ref: 4.5

Environment Variables - Miscellaneous

ABE=ASCII

Save processing tables in ASCII format. (*cabe)

Version Ref: 1.0

CABEBACKUP ON|OFF (on by default)

CABEBACKUPMINS n (minutes between backups)

CABEBACKUPCT n (backup files per process)

While editing a process it will automatically be * backed up depending on the settings of these variables: * CABEBACKUP (ON|OFF) * CABEBACKUPMINS (MINUTES) * CABEBACKUPCT (NUMBER OF BACKUPS BEFORE ROLLOVER) * Backups can be restored through menu item 5.

Version 6.0.01

PF64K=ON

Turns on/off size warning in cabe. (*cabe)

Version Ref: 4.5

PFPDFFONTSIZE

Default FPML/PDF font size is 12. Variable PFPDFFONTSIZE=nnn allows you to set the default size.

PDPOSTPRINT

Version 6.0.00

PDPOSTPRINT=ON | OFF

This variable needs to set to ON and will add additional behavior to PFPOSTPRINT.

PFPOSTPRINT is set to execute when output is run. This is used to launch external programs on the results such as opening a RTF, PDF, etc. on the resulting output.

DPOSTPRINT set to ON will expand the available PFPOSTPRINT to PFPOSTPRINTnn where nn is a number between 1 and 99

When a specific printer is called by filePro, the associated PFPOSTPRINTnn will be launched on the resulting output.

i.e. Printer1 would be associated with PFPOSTPRINT1, Printer2 w/ PFPOSTPRINT2, etc.

PFADDWP=OFF

Turns off adding .wp extension to export word. (*clerk, *report)

Version Ref: 4.5

PFAUTOKSIZE=nnn

Override default tok size for auto processing. Equivalent of -ty flag. If not set, default value is "20000" prior to 5.6.0 and "100000" with 5.6.0 and later. (*cabe, *clerk, *report)

Version Ref: 4.8

PFBACKGROUND=OFF|ON

Turns off ability of user to throw session into background via -bg or !g for "**report" and "dxmaint". Default is "ON" (*report, dxmaint)

Version Ref: 4.8

PFBLANKOV=ON | OFF

Causes date math with blank dates to return "/OV". Default (unset) is OFF (*clerk, *report)

Version Ref: 4.8.8

PFBLDFREE=OFF | ON

Freechain build message. Default is OFF. (*clerk, *report)

Version Ref: 4.5

PFBREAK=OLD

Processing halts when break key pressed. (new or non-set behavior is to continue report only see PFCLKBREAK for clerk) (*report)

Version Ref: 4.5

PFBRWFORMPWD

Password protection of .brw formats

By setting the environment variable PFBRWFORMPWD to ON, one can then select certain .brw formats and assign a password to protect against

unauthorized changing and saving of the .brw format. Without knowing the password assigned to the .brw format and PFBRWFORMPWD set to OFF (default if not set) you will not be able to modify and save. With this set to ON, you will see a new Password option when accessing browse format in *clerk (IUA).

Version Ref: 5.8.03

PFBRWSLASH=OFF

Some European character sets (i.e.: Norwegian) use character 0x5C (the backslash in US-ASCII) as a letter of their alphabet. Setting PFBRWSLASH=OFF turns off filePro's backslash-code handling for browse lookups. (*clerk)

Version Ref: 4.8

PFCLKBREAK=OLD

Return to last function when break key pressed. (*clerk only - see PFBREAK for report)

Version Ref: 4.5

PFCLOSEPENDWARNING=OFF

Add PFCLOSEPENDWARNING=OFF to disable the warning if you attempt to close an HTML tag when it was not open.

Version Ref: 5.0.15

PFDDFXMAINT

ddefine will now use the version 4.5+ dxmaint interface when making indexes for new files. Set PFDDFXMAINT to OFF to revert to the old 4.1 index routine in ddefine

Version Ref: 5.8.03

PFDLGENTER=ON

Enter key acts like a save key.(*clerk)

Version Ref: 4.5

PFEDFAILBOX

(4.8.09) Causes "edit failed" messages to appear in a popup box, rather than flash at the bottom of the screen. (Similar to PFLOCKBOX=ON)

Version Ref: 4.8.9

Default: Unset is OFF.

PFEOF=nnn

Sets the End-of-field marker character for filePro's memo editor.

Version Ref: 5.0

Default: 17

PFEOB=nnn

Sets the End-of-Paragraph marker character for the filePro memo editor.

Version Ref: 5.0

Default: 17 or the override value set by "PFEOF".

PFEXPORTALL=ON

EXPORT ASCII/WORD would always export the same number of fields, regardless of whether the fields were assigned to on each record, even if they were only referenced in a comment. Now, filePro will only export the number of fields as the highest-reference field actually assigned.

For example:

If:

Then: out[1] = 1 ; out[2] = 4

If: xx = "y"

Then: out[3] = 3 ; out[4] = 4

If:

Then: ' out[5] = 5

filePro would previously always exported 5 fields. Now, if x="y" is true, it will export 4 fields, and if false will export 2 fields. To revert back to the old behavior, set PFEXPORTALL=ON.

Version Ref: 5.0.14

PFFIXEDLISTSIZE=ON

Version Ref: 5.0.9

Prevents filePro from shrinking selection lists. This allows screen readers for the blind to be programmed with fixed screen locations for such lists.

Default: OFF

PFFIXNOLOCK=OFF

Version Ref: 5.0.9

Turns off a change in behavior related to how filePro handles posting to a lookup that does not have a "-p" to lock the record. This returns the behavior to the method used in handling record locks prior to version 4.8.10

PFFORMTOKSIZE=nnn

Override default tok size for "FORM" command for @keyF (equivalent of -tf flag). If not set, default value is "20000" prior to 5.6.0 and "100000" with 5.6.0 and later. (*clerk)

Version Ref: 4.8

PFHELPAUTOGOTO=ON|OFF

Automatically forces F9 for index search upon entering help.

Version Ref: 5.0.2

PFHELDIR=path

Sets alternate help file directory. If help is not found in the specified path, filePro will look in \$PFPROG/fp/lib as well.

Note: This variable only applies for filePro /fp/lib help files. Application help files are not affected.

PFIDLEN=nnnn

Setting PFIDLEN=32 will cause @ID, @CB, and @UB to have a length of 32 rather than 8. (The only legal values are currently "8" and "32". Any other value is undefined.) Default is 8

Version Ref: 5.8.00

PFIMPBUF=nnnn

Increase default record length for importing ASCII files. Default "1024" prior to release 5.6.0 and "10000" with 5.6.0 and later. (*clerk, *report)

Version Ref: 4.1

PFINDEXX

This effects the X to exit availability when in Index Selection of *clerk. Default (not set) is when Index X does exist, X-Exit is not displayed or valid but if there is not Index X then it does display and is valid. PFINDEXX=ON means that X-Exit will be on all the time and if someone wishes to get to an X index they must arrow to it. PFINDEXX=OFF means that X-Exit will not display regardless of whether there is a X index or not.

Version Ref: 5.8.03

PFLBSIZE=nnn

Determines the maximum number of labels that can be in a processing table. Note that this includes labeled processing lines, lookups, arrays, aliases, and selection set names. (Note that using the same lookup name, alias, or selection set name more than once only counts as one entry.) The default value is 1000 but can be set to any value from 100 to 32,767.

Version Ref: 4.5

PFLISTSLASH=OFF

Some European character sets (i.e.: Norwegian) use character 0x5C (the backslash in US-ASCII) as a letter of their alphabet. Setting PFLISTSLASH=OFF turns off filePro's backslash-code handling for listbox. (*clerk,*report)

Version Ref: 4.8

PFLOCKBOX=OFF

Flashes "record is being updated" message. (*clerk, *report)

Version Ref: 4.5

PFLONGVARDOT=OLD

filePro used to accept a period in variable names. This is now not allowed (it never should have been allowed in the first place), in order to permit enhanced functionalities. To revert to the old behavior and allow periods in variable names, you can set PFLONGVARDOT=OLD. Note, however, that this will disable certain features, such as access to ODBC and biometrics, which require that periods not be allowed here.

Version Ref: 5.0.14

PFLOOKWIZPROT=ON | OFF

Changes the lookup wizard's "protect record". PFLOOKWIZPROT=ON will change the lookup wizard's "protect record" default to "Y". (*cabe)

Version Ref: 5.0.6

PFLX=ON | OFF

Globally disables the ability to create a browse lookup using the F6 key wherever it was possible to do so. Equivalent to the "-lx" command line flag. (*clerk)

Version Ref: 4.8

PFMASSUPDATE=off

Turns off the F4 Mass Update feature

Version Ref: 5.6

PFMBTO=nnn

(Unix only) Allows automatic timeout to any popup message boxes. PFMBTO=nn will cause all message boxes to time-out after nnn seconds, and act as if the user had pressed ENTER. (*clerk, *report)

Version Ref: 4.5.8 NIX

Version Ref: 5.8.00 Windows

PFMENBRK=OLD

Restores 4.1 behavior when pressing break in a menu. (*clerk, *report)

Version Ref: 4.5

PFMISSINGARG=OLD

PFMISSINGARG=OLD reverts back to old 4.1 behavior of ignoring missing arguments to command line flags, such as "-pn".

Version Ref: 4.8

PFMU=OFF

Turns off "protect lookup" in cabc lookup. (*cabc)

Version Ref: 4.5

PFNODF=ON

Disables ddefine, dexpand free-disk space check. (ddefine, dexpand). Use for large disk drives when there is an error reporting "Insufficient Disk Space" and there is obviously plenty of free disk space.

PFNODFMSG=OFF

Turns off ddefine's "PFNODF=ON" notice. (Default: ON)

Version Ref: 5.6.2

PFNOHELP=ON | OFF

Displays "No Help Available" if ON. (*clerk, *report)

Version Ref: 4.1

PFNOQUAL=OFF

Turns off "[NONE]" from the qualifier list. (*clerk, *report, dxmaint)

Version Ref: 5.0

PFOLDMEMO

New variable PFOLDMEMO, default value is ON. Setting it to OFF will enable the new tokenizing code that was causing rclerk/rrport to crash on blobs.

IMPORTANT: This is used in conjunction with the new maximum length parameter used in MEMO edit for use with WEBfilePro. Any attempt to use old compiled .tok processing tables with BLOBS will result in errors in memo handling. When using the new parameter and tokenized processing tables requires this set to OFF and the tok table recompiled.

Version Ref: 5.8.03

PFOLDONCE=ON

Although @ONCE in *report is documented as being run prior to any output being done, it was run while sitting on the last record read during the sort/select process. Some people thought that this meant that it was sitting on a selected record.

@ONCE has now been fixed to be not sitting on any record. However, some people depend on their incorrect interpretation of the old behavior, so setting PFOLDONCE=ON will "revert back" to a modified version of the old behavior, where it will now be run while sitting on the last record _selected_ during the sort/select process.

Version Ref: 5.0.14

PFOUTS=parameters

Specifies serial communication parameters. (*clerk, *report)

Version Ref: 4.1

PFPDFCOMPRESSMODE

"PFPDFCOMPRESSMODE=nnn" to set the PDF compression mode. Default: 2 (images only). Possible values are the sum of:

1 = text

2 = images

4 = meta data

15 = all

PFPOSTPRINT=cmdline

(4.8.05K3) When set to "cmdline", filePro will execute "cmdline filename" after any printout or hardcopy, if destination is a filename.

Version Ref: 4.8

PDPOSTPRINT=ON|OFF

This variable needs to set to ON and will add additional behavior to PFPOSTPRINT (see PDPOSTPRINT notes)

Version Ref: 6.0

PFQUAL=qualifier

Qualified data set to use. (ddir, dexpand, *clerk, *report, dxmaint)

Version Ref: 3.0

PFQUALMESG="text"

Replace "Enter File Name Qualifier" prompt spawned by '-md' flag with prompt of your own choosing. (ddir, dexpand, *clerk, *report, dxmaint)

Version Ref: 5.0

PFREADONLYWARNING=OFF

If: LOCKED(-) support has been added.

Version Ref: 5.7.01

PFREFRESHRATE=nnn

Sets the screen refresh rate during sort/select and output phases to once every "nnn" seconds. (dxmaint/*report)

Default = 1

Version Ref: 5.0.6

PFRETRY=nnn

Number of retries for locked read. (*clerk, *report)

Version Ref: 4.5

PFSCC=ON

This will enable the "!scc" shell-escape within dclerk and rclerk, which has been disabled by default. (*clerk)

Version Ref: 4.8

PFSELECTBOXCASE

This variable can be set in the config file to determine the default case value so that it does not need to be programmed on the command line.

PFSELECTCASE=n (n = "0", "1", or "2", default=0)

Version Ref: 5.8.02

PFSEMTIMEOUT

Watchdog code added to the session count code in filePro to prevent semaphore lockups. The value defaults to 3 seconds before it will unlock a broken semaphore. A value of 0 disables the new timeout.

PFSELFORMPWD

Password protection of .sel formats

By setting the environment variable PFSELFORMPWD to ON, one can then select certain .sel formats and assign a password to protect against unauthorized changing and saving of the .sel format. Without knowing the password assigned to the .sel format and PFSELFORMPWD set to OFF (default if not set) you will not be able to modify and save. With this set to ON, you will see a new Password option when accessing browse format in *clerk (IUA).

Version Ref: 6.0.00

PFSKIPLOCKED=nnn

Allows locked records to be skipped after nnn seconds. Records skipped not included in @rp. (*report)

Version Ref: 4.8

PFSKIPPEDLOG=file name

Log records skipped by PFSKIPLOCKED=nnn. (*report)

Version Ref: 4.8

PFSP=xxx

Overrides the site password stored in fppath. (*cabe)

Version Ref: 4.5

PFSYNC=ON | ALL

ON = sync after expanding file, ALL = sync after all writes. (*clerk, *report)

Version Ref: 4.5

PFTOKSIZE=nnn

Override the default token table size. If not set, default value is "20000" prior to 5.6.0 and "100000" with 5.6.0 and later. (*cabe, *clerk, *report)

Version Ref: 4.1

PFVER=ON

Show individual filePro program version.

Version Ref: 1.0

PFWGT0=ON

Forces dreport and rreport to do @WGTprocessing even if no records are selected. (*report)

Version Ref: 4.8

PFXFERDOS=OLD

(4.5, SCO Unix only) PFXFERDOS=OLD tells xferdos to use the old dosdir/doscp command syntax. Default is to use SCO's new syntax. (xferdos)

Version Ref: 4.5

PFZEROLENWARN

This setting allows you to turn off the compiler warning for assigning a value to a zero length field.

Version Ref: 6.0.00

AIX/UNIX/LINUX

PFQUIT=OFF

Disables the ability of UNIX users to press the CTRL \ to exit a program.

Version Ref: 4.8

PFROOTFIX=OFF

Some *NIX systems prevent a setuid program running with a real uid of root from executing child processes. The previous workaround (setting the real uid to "filepro" when running as root) causes some things (such as printer banner pages) to report "filepro" as the user. "PFROOTFIX=OFF" turns off the fix for systems that don't require it.

Version Ref: 5.0.12

PFSYSEUID=OFF

If OFF, then SYSTEM command will be executed without the filePro setuid. Some systems may not allow a program to reclaim a setuid after giving it up. On these systems, setting this variable will cause bad things to happen to filePro. SCO OSV5 does not have a problem with this. LINUX kernels prior to 1.1.37 have a problem. The default is ON

Version Ref: 4.8

PFTERM=terminal

Type of terminal being used. Must match termcap file (UNIX). Same as TERM except exclusive to filePro.

Version Ref: 4.0

PFUMASK=nnn

Controls the umask value when creating files using HTML:CR and JSFILE :CR and PDF files

Note: Use a decimal value for nnn, hex value 0xNNN or Octal value 0nnn (**version 5.0 or later**).

Basically, the easiest way to think about it is to take whatever base number you have 8 (octal) 10 (decimal) 16 (hex) and convert it to base 8 (octal)

Then to figure out the resulting permissions of your PFUMASK, take 777 minus whatever you got in base 8.

For example, 777 minus PFUMASK 111 would give you 666 permissions on a created file.

PFUMASK 222 would give you 555 for your created file permissions, 333 would be 444 and so on

Version Ref: 4.8.9

TERM=terminal

Type of terminal being used. Must match termcap file (UNIX). See PFTERM also.

Version Ref: 1.0

TERMCAP=filename

Filename overrides "/etc/termcap" file (UNIX).

Version Ref: 1.0

Windows/Network

PFCLOCK=ON | OFF

Enables or disables clock displayed in menus. Default is "OFF".

Caution: Setting PFCLOCK=ON will cause filePro to use excessive CPU time, while the clock is displayed at the menu, and may degrade performance of other programs.

Version Ref: 4.8

PFCURSOR=nnn

(Native windows version only) Allows you to set the height of the cursor within filePro. (Similar to obsolete CURSOR=LINE.) Enhanced to allow PFCURSOR=nnn, where "nnn" is a number from 1 to 100, which indicates the percentage of the height of the character cell that is filled by the cursor. PFCURSOR=LINE is equivalent to setting PFCURSOR=10. Default value is 99. Setting PFCURSOR=0 will cause filePro to use whatever is the current cursor height.

Version Ref: 4.8

PFFILES=nn

filePro uses more than 20 file handles.

Version Ref: 4.5 (Obsolete)

PFLICFILE

Override the default license path for fileProODBC.

Set PFLICFILE = %pfprog%\fp\license\licfp.dat

Default path is %pfprog%\fp\lib\licfp.dat

Version Ref: ODBC 1.0.01

PFKEYTAB=table

Changes filePro key table as specified.

Version Ref: 4.5 Default is **filepro.key** **dos.key** emulates Windows default keystrokes

Version Ref: 5.8 **filepro-del.key** file added to the lib folder to support using the DELETE key on Windows as the BREAK key. All other keystrokes remain the same as default filepro.

Set PFKEYTAB=filepro-del in the config file.

PFLABEL=OLD

Allow invalid characters in a prc tables.

Version Ref: 4.5

PFNET=ON

Use Network calls. (DOS only)

Version Ref: 3.0

PFNEWNTCONSOLE=ON

Forces startup code to create a new console for java RunMenu to execute. (Native95 only)

Version Ref: 4.8

PFSEVRROOT=path

Sets the path to the implied "root" directory for HTML and JSFILE commands.

Version Ref: 4.8

PFSKHEX=ON (Native windows only)

Version Ref: 5.0

Some copies of 4.8 native windows were accidentally released with debugging code enabled, that caused @SK to contain the hex value of any keystroke that didn't have a "real" @SK value. Since this breaks the ability to test @SK="", it has been turned off. But, since several people have commented that they like the behavior, setting PFSKHEX=ON will re-enable it.

PFSHOWMINERROR=ON

Shows value of GetLastError() when a system error occurs.

Default is OFF

Version Ref: 5.0.6

FileProGI

PFNEWSK=ON

Allows new @sk values to be seen by processing. Specifically, the only value right now is "MOUS", which will be seen as "ENTR" if this variable is not set to "ON".

Default is OFF

Version Ref: 5.0.6

PFFORCECURSORPATH=OFF

Cursor path is enforced within the GUI environment. Cursor path will not be enforced if value is set to "OFF"

Default is ON

Version Ref: 5.0.6

FileProODBC

PFODBCCOMMITTYPE=n

Selects the open-commit-type to use for high-level ODBC data sources, where:

0 = "SELECT * FROM tablename" (default)

Very slow on some data sources with very large files, but uses nothing non-standard.

1 = "SELECT * FROM tablename WHERE id_field = nnn" (Where "id_field" is the name of the ID field, and "nnn" is a valid ID.)

Usually faster, but may be slower on some systems, as filePro must first determine a valid ID to use.

2 = "SELECT TOP 1 FROM tablename"

Fastest version, but "TOP 1" is non-standard and not supported everywhere. Will cause ODBC failure on those DSNs that don't support it.

Version Ref: 5.0.14

PFCMARK=nn (Default (unset) operation = 00)

Marks the beginning of the century. "nn" is a 2 digit year between 00 and 99 (inclusive). Causes all 2 digit years before "nn" to be interpreted as 20nn and all 2 digit years from "nn" through 99 to be interpreted as 19nn.

Version Ref: 4.5

Default: 00, 2 digit dates span 1900 to 1999.

PFDIRFILTER=ON

Turns on the filter that verifies that only directories appear in the filePro filename list. Some Unix users had serious slowdowns with the filter enabled.

Version Ref: 4.8

(Default: OFF)

PFERRKEY=k

Where "k" is the key to return from a filePro/system error screen.

Version Ref: 4.5

Example:

PFERRKEY=#

Note: Set PFERRKEY to some symbol that isn't likely to be accidentally pressed before seeing the error message. Using "X" as the exit key is not recommended, as it will conflict with the case where the normal choices are Enter to continue or X to exit, as this will now be "X to continue or X to exit".

PFLANG=language

Allows different collating sequences for different language character sets.

Version Ref: 4.5

Syntax:

PFLANG=language or PFLANG=/full/path/to/collate/file

If only a language is specified, filePro uses the "language.col" file in the "fp/lib" directory. If a full path is specified, you must include the full filename, including ".col" if necessary. If not set, filePro uses its built-in tables, based on the default 8-bit extended ASCII character set (code page 437) and the English language.

Example:

Assuming filePro is installed in the "/appl/fp" directory, the following both load the Norwegian language file:

PFLANG=norway

PFLANG=/appl/fp/lib/norway.col

PFME=ON

Waits for Return key to be pressed for next field. Default is OFF. (*clerk)

Controls data entry. With PFME=ON, the user must press ENTER in order to move out of the field, or select a menu choice. With PFME=OFF, filePro moves cursor automatically whenever field is filled or menu choice is selected.

Version Ref: 1.0

Default: OFF

PFNAME=filename

Same as using Set/Change filename from menu.

Version Ref: 1.0

PFOLDWRITE=ON

WRITE in processing was not properly holding variable field data

PFOLDWRITE=ON will revert back to the original behavior if this fix effects an application

Default is OFF Version Ref: 6.0

PFREADONLY=ON

Forces read-only attribute on session/file.

Version Ref: 4.8

PFSYSYR4=ON | OFF

Forces @TD, @CD, @UD, and @BD to 4-digit years.

Version Ref: 4.8

PFUFLAG=ON

Forces "-u" equivalent in *report.

Version Ref: 4.8.8

Environment Variables Output

PFNB=ON

Turn banner printing off.

Version Ref: 3.0

PFNTPRTERR

(Native windows only)

If a printer error other than ERROR_NOT_READY, ERROR_WRITE_FAULT, or ERROR_IO_DEVICE occurs, the error might not be reported correctly, and you can get "error -1" rather than the usual "printer not ready" warning.

Default: Unset is OFF.

Version Ref: 4.8.10

PFONEHEAD=ON

Report prints header lines only once.

Version Ref: 4.5

PFPPQ=ON

Acts as if the "-PQ" flag was passed to *clerk/*report.

Version Ref: 5.8.01

PFPPQEXCLUDE

PFPPQEXCLUDE=<arguments> where arguments are printername1, printername2, etc.. These printers will be excluded from the selection list when using -PQ. This setting can be in the config file or in the environment.

Version Ref: 6.0.00

PFPPDFONTSIZE=nnn

Sets default font size for fpm/pdf

PFPRINTER=printertype

Sets the printer type and destination.

PFPRINTER=printertype

- Default (unset) operation = none

PFPRINTER=local

- Like -pt will send to local LPT1 even if network printer or captured port.

PFPRINTER=screen

- Like -pv will print to the screen.

Default: Default printer type and destination.

PFPRINTERxx= Default:

set to default printer

PFPRINTERxx=name,type,destination,description

(Default (unset) operation = default printer)

1-99 sets the characteristics of the specified printer. These are normally defined in Printer Maintenance and stored in the filePro plus configuration file. The Printer Maintenance editor allows the definition of the first 9 printers. The file must be edited manually or with the Configuration Editor found through the filePro Directory choice on the filePro Plus Main Menu. All printers added after #9 must be sequential and not skip any number. If you have PRINTER10, PRINTER11 and PRINTER13 defined, nothing higher than PFPRINTER12 will be recognized.

Version Ref: 4.8

Example:

PFPRINTER18=deskjet, hp-1600, PRN, send output to color printer.

PFPRRT=device/file

Directs output to a device or filename.

Version Ref: 3.0

PFPRRTC=printer_type

Sets the printer type.

Version Ref: 4.0

PFPT=ON

Local printing on (AIX/LINUX/UNIX only).

Version Ref: 4.0

PFSELECTBOXCASE

This sets a global method for SELECTBOX()

Version Ref: 5.8.03

PFPTO=nnn

Wait time in sec. for "Printer Ready".

(Windows only)

Version Ref: 4.0

PFTIMEOUT=nn

Same as PFPTO. Default 10 seconds.

Version Ref: 4.0

PFSPPOOL=spooler

Selects spooler/printer attached to spooler.

Example: PFSPPOOL="lp -s -dlaser"

Version Ref: 3.0

PFHCFF=ON

Page eject sent to printer after "H" for Hardcopy is pressed. (*clerk)

Version Ref: 4.1

SHOWPROGRESS=ON (GI & Web)

Default OFF

When set to ON, while the report processing is being run (not the selection processing), the center text is suppressed. This allows for things such as progress bars and status information to be drawn without being overwritten on the screen.

Environment Variables - Path

PFCONFIG=path

Overrides default path for config where the default path is /fp/lib/config.

Version Ref: 4.8

PFDATA=drive:

Drive letter for the "/filePro" directory.

Version Ref: 1.0

PFDATA=/dirname

Sets drive where filePro data files are located.

Version Ref: 1.0

Example: /hd1 (UNIX) or C: (Windows).

Also see: fppath

PFDIR=path

Path for the "/filePro" directory.

Also see: fppath

Version Ref: 1.0

PFDLDIR=path

Sets the path to the directory where a file to downloaded to a printer resides. PFDLDIR is used with %"filename" print code.

Version Ref: 4.1

PFDSK=drive_list

Identifies data drives. Overrides PFIGN. Do not use drive Z: on Windows OS since this drive letter is reserved for use in some systems such as Novell so is ignored by filePro.

Version Ref: 3.0

PFGLOB=filename

Sets the file to use as the global edits table. The file must reside in fp/lib directory and should contain the edits from the original global edits file. Note: The file name must be an acceptable Windows or UNIX path with the file name.

Version Ref: 4.1

Default: filePro uses default global edits file in the lib directory, " edits ".

Example: PFGLOB=/apl/fp/lib/myedits will use file named "myedits" instead of the default filename "edits" as the global edits table.

PFIGN=drives

Drives to ignore. (Windows only)

Version Ref: 1.0

Default: All drives are scanned for data directories

Example: PFIGN=AB tells filePro to ignore drives A and B when looking for data.

NOTE: Where possible, use PFDSK rather than PFIGN.

PFMENU=path

Set path for user menus.

Version Ref: 4.1

PFPERL=path

Identifies the path for the PERL executable used for PERL scripts in menus.

Version Ref: 5.0

Allow dmakemenu to create Perl scripts, not just shell scripts. If you have Perl on your system, these scripts can be portable, even between Windows and Unix. You mark the script as Perl by preceding the "/fp/menus/menuname.-x" command with "++". (You have to add this mark manually at the moment.) This will execute the script via "perl scriptname". You can set the environment variable PFPERL to point to the executable (ie: PFPERL=c:\bin\perl.exe) if it's not in the PATH. Further, you can use any environment variable to point to any command processor by placing the name of the environment variable that points to the executable between the "+" signs. For example, "+MYHELL+/fp/menus/menuname.-3" will use the program pointed to by the MYHELL environment variable.

PFPPROG=path

Path to the "/fp" directory.

Default: blank, so "fp " is appended to nothing and /fp or \fp becomes filePro program directory.

Sets the path to the "fp " directory, where filePro programs are located. Used by appending "fp " to the contents of the variable.

Version Ref: 1.0

Example: set PFPFROG=C:\fpdemo would look for filePro program files in C:\fpdemo\fp

Also see: fppath

PFTMP=directory

Tells filePro where to place temporary files when sorting and selecting records in Request Output and Index Maintenance.

Version Ref: 1.0

NOTE: The specified directory must already exist.

Default: under Windows, same directory as file s map. Under UNIX, /tmp.

DIALOGINVERSE=0xNN

Sets the foreground and background inverse colors for filePro dialog boxes.

Version Ref: 4.1

Default: Value of POPUPINVERSE

(for color-values see color charts)

DIALOGNORMAL=0xNN

Sets the foreground and background colors of filePro dialog boxes.

Version Ref: 4.1

Default: Value of POPUPNORMAL

ERRORINVERSE=0xNN

Version Ref: 3.0

ERRORNORMAL=0xNN

Controls colors of error boxes.

Version Ref: 3.0

HELPINVERSE=0xNN

Sets the foreground and background inverse colors for help screens.

Version Ref: 3.0

Default: Value of TEXTINVERSE

HELPNORMAL=0xNN

Sets the foreground and background colors of help screens.

Version Ref: 3.0

Default: Value of TEXTNORMAL

MENUBORDER=0xNN

Sets foreground and background colors of filePro menu borders.

Version Ref: 3.0

MENUINVERSE=0xNN

Controls menu inverse colors.

Version Ref: 3.0

MENUNORMAL=0xNN

Controls menu normal colors.

Version Ref: 3.0

PFBRWFORMPWD

By setting the environment variable PFBRWFORMPWD to ON, one can then select certain .brw formats and assign a password to protect against unauthorized changing and saving of the .brw format. Without knowing the password assigned to the .brw format and PFBRWFORMPWD set to OFF (default if not set) you will not be able to modify and save.

Version Ref: 5.8.03

PFDDEFCOLOR=ON|OFF (default ON)

When on, ddefine will create color screens, when off, monochrome.

Version Ref: 6.0.00

PFDIALOGPROMPT=LEFT/RIGHT/CENTER

Controls location of dialog box prompts. (Only first character is significant.)

Version Ref: 5.0

PFDROPSHADOW=ON|OFF

Turns the drop shadow on or off.

Version Ref: 3.0

PFENTSELDISABLE=list

This overrides the default options for *clerk and allows you to disable keystrokes (and remove prompts for) for the "Enter Selection" prompt. Any of the keys normally listed by filePro at the prompt e.g. D, H, U, X, F, and B can be disabled by including them as a "list" values.

For example, PFENTSELDISABLE=DF would disable (and remove the prompts for) "D-Delete" and "F-Print Form". It would also cause the new HELP prompt to appear as '?' if not included in the list.

A question mark (?) represents HELP. Note that the HELP key cannot be disabled with this variable, although you can keep the new prompt from appearing. The default value is "?", which causes the same prompts as before to appear. Any "invalid" keys listed are ignored.

Note that any @KEY events are not disabled. Only filePro's default behavior for the keystrokes is affected. Also note that if the HELP key label is longer than 3 characters, enabling all of the prompts will not fit in 80 columns.

PFINSERTMODE=ON

Set insert mode on by default in *cabe/*clerk.

Version Ref: 5.0.9

PFMEMOINSERTMODE=ON

Sets default memo editor insert mode to "on".

Version Ref: 5.0.10

PFNOBOX=ON

Eliminates the boxes around menus, headers, etc.

Version Ref: 4.1

PFSHADOWCOLOR=0xNN

Sets the color of the drop shadow. Default = 0x08 (dark gray on black)

PFSHOWROWCOL=OFF

Turns off the row/column display in programs like dscreen, dmoedef, and *cabe. It can confuse screen readers for the blind as the numbers are read every time you press a key.

Version Ref: 5.0.9

POPUPNORMAL=0xNN

Color code for popup windows foreground.

Version Ref: 3.0

POPUPINVERSE=0xNN

Color code for popup windows background.

Version Ref: 3.0

TEXTINVERSE=0xNN

Sets the foreground and background colors of menus, prompts, screens, fields, help text.

Default: Black foreground, white background.

Version Ref: 3.0

TEXTNORMAL=0xNN

Sets the foreground and background colors of menus, prompts, screens, fields, help text.

Version Ref: 3.0

Default: White foreground, black background.

PFMONO=ON

Tells filePro to use monochrome screens with a color video card.

(Windows only) - used primarily with dscreen.

Version Ref: 4.1

Setting Environment Variables

To set an environment variable for filePro WITHOUT using the built-in ENVIRONMENT CONFIGURATION EDITOR, you must follow the proper procedure based on the operating system you are using. (The configuration editor is ideal for setting variables in a global manner for all users. There are many times when you will not want to do this.)

Windows

Syntax: set variable=value

Normally, these environment variables would be put into the installation created batch file e.g. fulldev.bat or a personalized .bat file, which calls a filePro program. Under Windows, CASE is not significant for environment variables, i.e., setting ABE, aBE and Abe are all the same and will work.

Unix

Syntax: variablename=value ; export variablename

Under Unix, variables not exported will not be available to filePro. Normally, these environment variables would be put into the ".profile" file of the user who will be calling the filePro program. Or, environment variables can be put into a personalized script file, which calls a filePro program specifically.

IMPORTANT: Under Unix, CASE is VERY significant for environment variables, i.e., assigning values to ABE, aBE and Abe and exporting them, is defining three separate variables and filePro will only "see" or "use" the one that is ALL IN CAPS. Remember to keep all filePro environment variables under Unix ALL UPPERCASE or they will not work.

Background Processing Problems

If an error occurs while an operation is running in background, the message related to the error appears at the cursor position with the following addendum.

Error while Running filePro in Background.

The message is also "mailed" to you for later reference (see "mail" command in your operating system manual). You don't have to break out of the program but you may want to refresh your screen by pressing the <REDRAW> key (usually <CTRL> <L>). Refer to the [Terminal Guide](#) or termcap definition for the applicable key.

Error Initializing a Port

If an error occurs when attempting use a port on NIX systems such as "Error initializing serial port for 'OUTS'" with the command OUTS or PFOUTS, the permission and/or owner settings may need to be set. Try the following.

```
chmod 600 /dev/ttyXX  
chown filepro /dev/ttyXX  
chgrp uucp /dev/ttyXX
```

where XX is the actual port e.g. (1a, 1A, 2a, 2A etc..)

Unlocking Files

To control access to various parts of filePro, filePro maintains a locking system to prevent simultaneous access to various parts of the system when such access can cause problems. For instance, only one user can expand a file and only if nobody else is using the file at that point in time. The locking system will create a flag in the appropriate file to prevent file integrity problems. These files exist on multi-user and network systems in each filePro directory and qualifier as LOCKFILE, LOCKFQ1, LOCKFQ2, etc. (where Q1 and Q2 are qualifiers) and are maintained and accessed by all the applicable programs.

Although filePro manages the locking and unlocking processes routinely, unexpected events like power interruptions, hard disk failures, or inadvertent machine resets can leave a corrupted lock file if files were open when the interruption occurred. If this happens, you will have to unlock the files using filePro's unlock feature. Use the directory option to delete lock files by selecting the file and then pressing "D" to delete. Select "M" to delete the lock file for each file open during the unexpected event. You can also use a menu flag "-L" to delete a lock file with the ddir or dprodir program. Refer to [flag](#) options. Keep in mind that lock files are created for each set of data so you may have to set the qualifier name when using the directory option. When using the menu flags, the "-M" flag must be used in conjunction with the "-L" flag to unlock qualified data files.

filePro Error Messages

Code	Description
001	File not found You may be trying to reference a file that has never existed, has been renamed or has been deleted from the system. Return to processing table and check the name for typographical errors.
002	No or invalid map A disk drive error occurred while the program was reading or saving the map file. You may have to rewrite or re-save the file format using <i>Define Files</i> .
003	Sentence contains too many selection sets. Maximum is 5. You can reference no more than five sets in any one selection sentence.
004	Group not found in selection set. You mistyped a group or set name, listed a group or set that was never defined or used the restricted words "AND" or "OR" in your group names. Check your group names and list of selection sets using the filePro directory option.
005	Selection sentence too complex. There are too many groups and sets involved. Simplify your sentence.
006	Process contains a syntax error at position indicated. This error occurs when running a syntax check or when running a program if you didn't check the syntax of the processing table. The line of processing containing the error will have a caret "^" beneath the point of the error. If there appears to be no error at the point of the "^", check for errors preceding the caret. Another difficult thing to spot is an "undefined" dummy variable. The "^" will point to the undefined variable. The error will also occur when in <i>Inquire, Update, Add</i> and a lookup variable is not found. Example: 1=cust(2) ^ In this case, field 2 in cust was not found. Make sure that you have included a NOT condition to prevent this error when the value cannot be found in the lookup file. Example: If: NOT cust Then: END
007	Invalid selection set. You have somehow named a set incorrectly or used the wrong selection set name on a command line. Check the list of selection set names using the filePro directory option.
008	Selection contains self-reference. In the selection sentence. You have typed the name of a set you are currently defining or you have referred to a set that, in turn, refers to this set. Check your selection set names using the filePro directory option.
009	Error in termcap file. Alert your system administrator. LINUX/UNIX/XENIX versions of filePro: The termcap file (the file that contains information on the types and configuration of the terminals attached to your system) has an error in it. The file /appl/ftp/termcap applies. Correct the file or restore from a backup.
010	Terminal type not found. Alert your system administrator. LINUX/UNIX/ZENIX versions of filePro: The termcap file may be missing an entry for your terminal type. Either add the terminal type or set the terminal type to a different termcap entry that exists in /appl/ftp/termcap.
011	Required terminal feature not available. Alert your system administrator. Either a feature (for example a hardware instruction that clears the screen) or crucial configuration information (for instance - how to position the cursor) is missing. Refer to your system manuals and procedures on how to correct the problem.
012	Standard i/o has been redirected. You have redirected input or output incorrectly on the command or menu action line. You will not be able to run the filePro program until the redirection problem is resolved.
013	Too many filePro files open at once. Maximum is 60. You are looking up form or posting to too many files. Rewrite the process. Note that although you can use up to 60 files with filePro, the operating system can impose limitations. Although filePro closes some files when no longer needed, it cannot close all of them so the system limit could be exceeded.
014	Out of memory. The Central Processing Unit (CPU) doesn't have enough RAM to handle the filePro operation or program. You may need to add more memory to the unit or tune your system. Refer to minimum system and configuration requirements.
015	Processing tables element too complex. Simplify your processing.
016	Too many lookups. Maximum is 32 per processing set.
017	Automatic index too unbalanced. Run Index Maintenance to rebuild the index.
018	Disk error. This error usually indicates a hardware problem. Refer to your computer's operations manual.
019	Invalid index. Run Index Maintenance to rebuild the index or delete the invalid index.
020	Lookup failed. Correct the key field(s) and try again. While processing, the program could not find the record needed for the lookup. Check that you are using the right cross-reference data and that the data is listed in the lookup file. Also make sure that you are trapping for the non-existence of a data value by using a NOT condition following the lookup.
021	Checksum error: Format has been illegally modified. Either someone has tried to change the format from outside filePro or someone has made a copy of a format under another name and you have accessed the new copy. Re-create the format using the relevant filePro creation program.

- 022 **Incorrect password.**
You have typed a password incorrectly for the third time. When you press <RETURN>, you are returned to the menu item from which you came.
- 023 **Can't do this in automatic processing.**
Operation at position indicated is allowed in input processing only.
- 024 **Bad assignment at position indicated.**
Indicates a mismatch in field assignments. For example, you are trying to post data to a system maintained field or trying to move data from one field to another with a incompatible field type e.g. moving a date from a date field to a decimal field. Correct the processing table.
- 025 **Invalid screen format.**
Something is called "screen.n" but is not a valid filePro screen. Try re-saving the screen using *Define Screens* . If that doesn't correct the problem, delete the invalid screen from the operating system prompt or in *Define Screens* .
- 026 **Math overflow.**
This error occurs when you are attempting to perform math on non-numeric fields or have exceeded the limits of the math functions.
- 027 **Bad argument on command line**
You have a logical or typographical error on your command line. The "command line" is the line on which you type program names and parameters (file, selection set names, etc.) when defining user menus or when accessing programs from the operating system prompt.
- 028 **Invalid field in processing set.**
The field being used for a lookup doesn't exist. Correct the field number.
- 029 **Invalid index in processing set. (DOS only - Obsolete)**
The index being used for a lookup doesn't exist. Either build the index or change the index number/letter in Define Processing to refer to a valid index. Also, the program may have too many open files. Increase the number of files by setting the PFILES variable.
- 030 **Lookup without a field.**
The lookup has been defined without specifying the field to be used. This error occurs in *Define Processing* . You forgot the key-field number. Use the Define Lookups option key to prevent this type of mistake.
- 031 **Required index hasn't been specified.**
You forgot to put your index number/letter in your LOOKUP statement. Use the define lookups key <F5> to avoid this error.
- 032 **Goto destination not found.**
You have misspelled or forgot to define the label that a GOTO or GOSUB statement is suppose to access.
- 033 **Not available.**
Somebody else is using it; try again later.
You have attempted to modify a format while somebody else is using it or somebody else is modifying the same format. This error may also occur if the system is reset, in the middle of modifying a format or when files are open. Refer to UNLOCK procedures.
- 034 **User edit too complex.**
The edit is referencing itself or is nested too deeply. (A nested edit is one that refers to a second edit type that refers to a third edit etc.). Correct by simplifying the edit. The nesting limits are 15 for DOS; 20 for LINUX, UNIX, XENIX.
- 035 **Edit name not found.**
This error occurs in *Define Edits*. You have referenced an edit that does not exist. Check the list of global and local edits for the existence of the edit and typographical errors.
- 036 **Incomplete file.**
A disk drive error may have occurred while you were changing and/or saving a file. Have you had recent hardware problems? You may have to restore data from backups or rewrite your formats.
- 037 **Selection sentence contains a syntax error at position indicated.**
This message occurs when you have a selection set containing a syntax error. Fix the problem at the specified position.
- 038 **Not a menu.**
You have attempted to use a corrupted menu or a file that is not a menu.
- 039 **Field not found. Lookup has not been performed.**
This error occurs in *Inquire, Update, Add or Request Output* when the processing table is written in such a way that the program skips the lookup. For example, you may have a GOTO in the wrong place or be using the same destination file name for two separate lookups. The error can also occur when using LOOKUP-N ("don't report an error") without checking to see if the lookup failed.
- 040 **Edit type not found.**
This error occurs when defining dummy fields in processing tables and when accessing the file in *Inquire, Update, Add* . You have listed an edit type that doesn't exist or your PFGLOB variable is set wrong. Update the edit table using *Define Edits* , change the edit type used for the field in *Define Files* or correct the PFGLOB variable setting.
- 041 **File not available.**
Somebody else is modifying the file; try again later.
You cannot run the requested program while someone else is modifying the file. This error occurs when the system is inadvertently shutdown while filePro files are open or a session is not properly closed when in a window. Refer to UNLOCK procedures.
- 042 **Invalid parameter .**
A parameter to TOT, MN, MAX, AVG or MD is bad. The field you are trying to use is alphanumeric when it should be numeric, a date when it should be a decimal number, etc. Since the error appears during syntax checking, you can press <RETURN> to return to the processing table and change the field numbers, operation, etc.
- 043 **Merge name incompatibility.**
Word processor or spreadsheet merge has been given two different names. You can't use the same file name more than once on the same processing table. Check for the same merge name being used in 2 different elements on the processing table.
- 044 **RETURN without a GOSUB.**
You have a RETURN but no GOSUB. Check your processing to make sure that you haven't used a GOTO instead of a GOSUB or that you are not skipping the GOSUB function.
- 045 **Too many nested GOSUBs.**
A "nested GOSUB" is a subroutine that calls another subroutine that calls a third subroutine, etc. The limit is 16. Rewrite your subroutine to reduce the number of nested GOSUBs.
- 046 **Edit failed.**
When processing data, you may have copied information from one field that doesn't pass the edit of the second field. Change your processing or change the edit type or method.

- 047 **Invalid data-capture format.**
If you are using a filePro data transfer program, this message indicates that the format has been incorrectly defined or can't be used by filePro. Consult the transfer manual for instructions.
- 048 **Error Reading Text Index.**
- 049 **Error Reading Message File.**
This error occurs if the "errmsg" file is missing or corrupted. Re-install the filePro software to correct.
- 050 **Segment lengths do not match.**
Key and data segments contain differing number of records. This error occurs if there is a disk drive "write" problem or if a file-restructuring fails or is canceled improperly. Use the Expand File option to expand the file by one record to synchronize the key and data segments. If this does not work, restore your files from your most recent backups and start over.
- 051 **Bad Import Format. Not a valid DIF file.**
The spreadsheet file from which you want to read data is not recognizable by filePro. Recreate or revise the DIF file and try again.
- 052 **Merge Conflict.**
Cannot import and export same file. You have set up an IMPORT statement with the same name as EXPORT statement. Change one of the names.
- 053 **Attempt to read past end of merge file.**
When importing data from a DIF file, you have come to the end of the file without telling the program when to end. Add a line to the processing routine that tells the program to move onto something else e.g. if: NOT filename (no more data), then: END or something else.
- 054 Function not available for this file type.
- 055 **Unknown alien file type.**
Different versions of filePro may support different types of alien file formats. The version you are using cannot access this particular format. Upgrade to the most recent version of filePro.
- 056 **The tokenization table is too small.**
Use '-t bigger size' to make it bigger in your user menu or from a system prompt. Refer to FLAGS.
- 057 **Wrong index type.**
Re-build the index with the correct Index Maintenance version.
The index was built using an older version of filePro than the version of *clerk or *report that you are using. Make sure that you are using the correct version of the *Index Maintenance* (dxmaint) program.
- 058 **The processing table is not encoded.**
Use *Define Processing* to encode the processing table before using.
- 059 **Too many locks.**
On network DOS systems only, the operating system of filePro plus limits the number of files locked at any one time. You have exceeded this number (usually 200); to fix the problem, simply close some of the files used in processing.
- 060 **Duplicate files found while scanning drives.**
Occurs if filePro detects more than one copy of the same file name on multiple drives. Set the environment variable *PFDSK* to limit filePro's view of the multiple drives or remove one of the copies.
- 061 **Can't Find the Printer.**
Make sure that your printer destination and mapping is correct.
- 062 **Reference to a field that doesn't exist.**
You may have re-defined a file structure when using Define Files and failed to modify your processing to match.
- 063 **Invalid runtime format .**
Use correct version of 'rcabe' to re-compile processing table.
- 064 **Index Is Too Big.**
Rebuild the index using a smaller key.
- 065 **Too many users**
You have exceeded the number of registered user licenses. Limit the use of system calls from processing tables or upgrade your license to add more users.
- 066 **Too many users**
You have exceeded the number of registered user licenses. Limit the use of system calls from processing tables or upgrade your license to add more users.
- 067 **Screen must be at least 80x24.**
You are attempting to use a screen resolution that is less than 80x24. Update or re-configure hardware.
- 068 **No such printer.**
You are attempting to use a printer that is not defined. Refer to Printer Maintenance options.
- 069 **Can't nest CALL's**
Remove the CALL from the CALLED processing table.
- 070 **TOK table exceeds 64K (only for 16 bit systems)**
Reduce the size of the processing table.
- 071 **Invalid key-map table.**
Check the setting of the PFKEYTAB variable.
- 072 **Invalid language (sort/collate) table.**
Check the value of the PFLANG variable to make sure that it is a valid table.
- 073 **Exceeded demo version limits.**
You are limited to 100 records per file and 200 lines per processing table.
Create test files that do not exceed these limits.
- 074 **Cannot establish connection to license server.**
Make sure that the license server is running.

- 075 **Im error: NO LICENSES REGISTERED**
Contact fP Technologies technical support.
- 076 **Im error: LICENSE EXPIRED**
Contact fP Technologies technical support.
- 077 **Im error: NO LICENSES FOUND FOR THIS PRODUCT**
Contact fP Technologies technical support.
- 078 **Im error: UNKNOWN REPLY FROM LICENSE MANAGER**
Contact fP Technologies technical support.
- 079 **fileProODBC Error: Cannot open DSN.**
Use Define Files (High Level ODBC) or Define Processing (Low Level) to set the correct DSN.
- 080 **fileProODBC Error: Cannot open table.**
Ensure that the table name is correct and exists in the data source.
- 081 **Spellcheck error.**
Contact fP Technologies technical support.
- 082 **Feature not licensed.**
You have attempted to use a feature that is not included in your license.
Contact fP Technologies sales to purchase the licensed feature.
- 083 **License error.**
This error will occur if the license is in the wrong place, improperly named or is corrupted.
Check that "licfp.dat" exists in your \$PFPROG/ftp/lib directory. Use program "licinfo" to check the licence. Contact fP Technologies technical support and provide the results returned by "licinfo".
- 084 **License manager platform mismatch.**
The license that you are attempting to use does not match the platform that you are using it on.
If you have multiple licenses, make sure that you are using the correct license. Use program "licinfo" to check the license and provide the results of this program to fP Technologies sales to obtain a new license or correct version of filePro.
- 085 **Generic XML error.**
- 086 **Failed to decrypt record.**
- 087 **Failed to verify encryption key in map.**
- 088 **This file is not licensed to run on this system.**
- 089 **Encrypted file / grace period mismatch.**

Other System Errors

Cannot Open File

Example:

C:\fp50\filepro\test: No such file or directory

Error occurs when the parent \filePro directory is missing or a filePro sub-directory is missing or invalid. Check your path environment settings and make sure that the \filePro directory exists for the path specified by PFDSK and PFDIR or the values in fppath.

Memory could not be read from

This error occurs in Windows and indicates that a file is invalid. These errors are sometimes difficult to isolate but if approached logically can be resolved. Make a note of the error and what menu option this error occurs on e.g. Define Files, IUA, Request Output, etc.

Examples:

When using Define Files you receive the following error.

The Instruction at 00401128 referred memory at ffffffff the memory could not be read from. This program has performed an illegal operation and will be shutdown.

The above error is probably due to an invalid MAP file. Somebody may have inadvertently revised the MAP using a non-filePro editor or you may have had a disk error that corrupted the map. Restore the MAP and MAP.TMP from a backup and try again. If you don't have a backup, try copying MAP.TMP to MAP after renaming MAP to something else like MAP.OLD.

Note: An invalid local edits file can also cause this type of error in *Define Files* since the local edits table is accessed when defining a file. Try renaming the EDITS file to EDITS.OLD to isolate as to whether the map or edits file is the problem. If renaming the EDITS file solves the problem, print the EDITS file using a text editor and reenter your local edits for this file.

If you get the "Instruction error" when defining local edits, the problem can be due to an invalid MAP file or invalid EDITS file. Try renaming the EDITS file to EDITS.OLD and go to the *Define Files* option. If you still get the error, you have an invalid map file. Go to above section and resolve the map problem.

If you get the "Instruction error" when using Inquire/Update/Add or "Request Output", these are the most difficult to resolve and can be due to any of following. Use the following chart to isolate your problem.

Problem	Process
Environment not properly set.	Isolate by determining if the error occurs on more than one file or all files. If the error occurs for all files, check your environment settings and try again. Refer to setting path environment variables.
Corrupted Index	Try removing all indexes and rebuilding one at a time until the error re-occurs. Make sure that you are using the same versions of the index maintenance program *clerk and *report. Contact filePro sales if the version numbers are different.
Corrupted key or Data file	Use the fpCopy utility to create a duplicate file using "Copy formats only". Try entering data into the new file. If you are able to enter data into the new file, you more than likely have a corrupted key or data files.
Corrupted edits file	Rename the edits file to edits.old or edits.sav and try to define files. If you cannot view the map with the Define Files option, the problem has been isolated to the map file. If renaming the edits solves your problem, print the edits.old or edits.sav file using a text editor and reenter the local edits for the filePro file.
Corrupted map file	Use fpCopy to copy the file without data e.g. "Copy formats only". If you cannot copy the map file, it is corrupted. If you are able to copy the map file, try entering data. If you cannot enter data into the new file, you probably have a corrupted local edits file. Follow the procedure for correcting a corrupted edits file.
Divide by 0	Divide by zero "/D0" is caused by invalid math expressions that include a zero as a divisor. This can also be caused by attempting to use a string as a divisor. Make sure that fields used in math formulas use numbers and not strings and divisors are not equal to zero.
Overflow	"/OV" is an overflow when attempting to use an invalid date or date math expression. If PFCMARK has not been properly set, some 2 digit years will not be considered as valid and will create this error.
Error Expanding File	This error may appear if the data segment has been corrupted. You may also see "Error Reading Free Chain Pointer" meaning that filePro is having trouble finding the next free record due to the corrupted file. Restore a recent backup of your data prior to system failure.

System Errors

Code	Description
#0	System error has occurred - System Error 0 Something is wrong but the system can't tell exactly what. You may be out of disk space, the program or operating system may have overwritten memory, have a bad copy of a program, etc. Reboot the system (according to proper procedures) and try the operation again. If the problem persists, re-install the filePro programs and try again. Note: This error can also occur when there are too many open files. Refer to SystemError #4.
#1	Bad function number. An internal and uncommon error. Follow the instructions for Systemerror #0.
#2	File not found. The operating system cannot find a file. If the file does exist, follow the instructions for Systemerror #0.
#3	No such path. The operating system cannot find a directory or folder. Follow the instructions for Systemerror #0.
#4	Too many open files. You have reached the operating system limit for open files. Change the file number setting in your system configuration to at least 20 and make more liberal use of the CLOSE command for your lookups. Although filePro closes some files when no longer needed, it cannot close all of them so the system limit may be exceeded.
#5	Access denied. This error occurs on network systems when you are denied access to certain files. Make sure that you have proper rights and access to filePro files and have isolated the problem by logging in with supervisor or administrator rights.
#6	Invalid handle. An internal and uncommon error. Follow the instructions for Systemerror #0.
#7	Memory control blocks destroyed. An internal and uncommon error. Follow the instructions for Systemerror #0.
#8	Not enough memory. You don't have enough RAM to run the program. There are a variety of solutions depending on the operating system and system configuration. Refer to system configuration and minimum system requirements in our FAQ for recommended config.sys settings and hardware requirements.
#9	Invalid memory block address. Part of a program may have been overwritten. Refer to instructions for Systemerror #0.
#10	Invalid environment An internal and uncommon error. Follow the instructions for Systemerror #0.

Windows Error Messages

No.	Code	Error
0	0x0000	The operation completed successfully.
1	0x0001	Incorrect function.
2	0x0002	The system cannot find the file specified.
3	0x0003	The system cannot find the path specified.
4	0x0004	The system cannot open the file.
5	0x0005	Access is denied.
6	0x0006	The handle is invalid.
7	0x0007	The storage control blocks were destroyed.
8	0x0008	Not enough storage is available to process this command.
9	0x0009	The storage control block address is invalid.
10	0x000A	The environment is incorrect.
11	0x000B	An attempt was made to load a program with an incorrect format.
12	0x000C	The access code is invalid.
13	0x000D	The data is invalid.
14	0x000E	Not enough storage is available to complete this operation.
15	0x000F	The system cannot find the drive specified.
16	0x0010	The directory cannot be removed.
17	0x0011	The system cannot move the file to a different disk drive.
18	0x0012	There are no more files.
19	0x0013	The media is write protected.
20	0x0014	The system cannot find the device specified.
21	0x0015	The device is not ready.
22	0x0016	The device does not recognize the command.
23	0x0017	Data error (cyclic redundancy check)
24	0x0018	The program issued a command but the command length is incorrect.
25	0x0019	The drive cannot locate a specific area or track on the disk.
26	0x001A	The specified disk or diskette cannot be accessed.
27	0x001B	The drive cannot find the sector requested.
28	0x001C	The printer is out of paper.
29	0x001D	The system cannot write to the specified device.
30	0x001E	The system cannot read from the specified device.
31	0x001F	A device attached to the system is not functioning.
32	0x0020	The process cannot access the file because it is being used by another process.
33	0x0021	The process cannot access the file because another process has locked a portion of the file.
34	0x0022	The wrong diskette is in the drive. Insert %2 (Volume Serial Number: %3) into drive %1.
36	0x0024	Too many files opened for sharing.
38	0x0026	Reached end of file.
39	0x0027	The disk is full.
50	0x0032	The network request is not supported.
51	0x0033	The remote computer is not available.
52	0x0034	A duplicate name exists on the network.
53	0x0035	The network path was not found.
54	0x0036	The network is busy.
55	0x0037	The specified network resource or device is no longer available.
56	0x0038	The network BIOS command limit has been reached.
57	0x0039	A network adapter hardware error occurred.
58	0x003A	The specified server cannot perform the requested operation.
59	0x003B	An unexpected network error occurred.
60	0x003C	The remote adapter is not compatible.
61	0x003D	The printer queue is full.
62	0x003E	Space to store the file waiting to be printed is not available on the server.
63	0x003F	Your file waiting to be printed was deleted.

64	0x0040	The specified network name is no longer available.
65	0x0041	Network access is denied.
66	0x0042	The network resource type is not correct.
67	0x0043	The network name cannot be found.
68	0x0044	The name limit for the local computer network adapter card was exceeded.
69	0x0045	The network BIOS session limit was exceeded.
70	0x0046	The remote server has been paused or is in the process of being started.
71	0x0047	No more connections can be made to this remote computer at this time because there are already as many connections as the computer can accept.
72	0x0048	The specified printer or disk device has been paused.
80	0x0050	The file exists.
82	0x0052	The directory or file cannot be created.
83	0x0053	Fail on INT 24
84	0x0054	Storage to process this request is not available.
85	0x0055	The local device name is already in use.
86	0x0056	The specified network password is not correct.
87	0x0057	The parameter is incorrect.
88	0x0058	A write fault occurred on the network.
89	0x0059	The system cannot start another process at this time.
100	0x0064	Cannot create another system semaphore.
101	0x0065	The exclusive semaphore is owned by another process.
102	0x0066	The semaphore is set and cannot be closed.
103	0x0067	The semaphore cannot be set again.
104	0x0068	Cannot request exclusive semaphores at interrupt time.
105	0x0069	The previous ownership of this semaphore has ended.
106	0x006A	Insert the diskette for drive %1.
107	0x006B	Program stopped because alternate diskette was not inserted.
108	0x006C	The disk is in use or locked by another process.
109	0x006D	The pipe has been ended.
110	0x006E	The system cannot open the device or file specified.
111	0x006F	The file name is too long.
112	0x0070	There is not enough space on the disk.
113	0x0071	No more internal file identifiers available.
114	0x0072	The target internal file identifier is incorrect.
117	0x0075	The IOCTL call made by the application program is not correct.
118	0x0076	The verify-on-write switch parameter value is not correct.
119	0x0077	The system does not support the command requested.
120	0x0078	This function is only valid in Win32 mode.
121	0x0079	The semaphore timeout period has expired.
122	0x007A	The data area passed to a system call is too small.
123	0x007B	The filename, directory name, or volume label syntax is incorrect.
124	0x007C	The system call level is not correct.
125	0x007D	The disk has no volume label.
126	0x007E	The specified module could not be found.
127	0x007F	The specified procedure could not be found.
128	0x0080	There are no child processes to wait for.
129	0x0081	The %1 application cannot be run in Win32 mode.
130	0x0082	Attempt to use a file handle to an open disk partition for an operation other than raw disk I/O.
131	0x0083	An attempt was made to move the file pointer before the beginning of the file.
132	0x0084	The file pointer cannot be set on the specified device or file.
133	0x0085	A JOIN or SUBST command cannot be used for a drive that contains previously joined drives.
134	0x0086	An attempt was made to use a JOIN or SUBST command on a drive that has already been joined.
135	0x0087	An attempt was made to use a JOIN or SUBST command on a drive that has already been substituted.
136	0x0088	The system tried to delete the JOIN of a drive that is not joined.

137	0x0089	The system tried to delete the substitution of a drive that is not substituted.
138	0x008A	The system tried to join a drive to a directory on a joined drive.
139	0x008B	The system tried to substitute a drive to a directory on a substituted drive.
140	0x008C	The system tried to join a drive to a directory on a substituted drive.
141	0x008D	The system tried to SUBST a drive to a directory on a joined drive.
142	0x008E	The system cannot perform a JOIN or SUBST at this time.
143	0x008F	The system cannot join or substitute a drive to or for a directory on the same drive.
144	0x0090	The directory is not a subdirectory of the root directory.
145	0x0091	The directory is not empty.
146	0x0092	The path specified is being used in a substitute.
147	0x0093	Not enough resources are available to process this command.
148	0x0094	The path specified cannot be used at this time.
149	0x0095	An attempt was made to join or substitute a drive for which a directory on the drive is the target of a previous substitute.
150	0x0096	System trace information was not specified in your CONFIG.SYS file, or tracing is disallowed.
151	0x0097	The number of specified semaphore events for DosMuxSemWait is not correct.
152	0x0098	DosMuxSemWait did not execute; too many semaphores are already set.
153	0x0099	The DosMuxSemWait list is not correct.
154	0x009A	The volume label you entered exceeds the label character limit of the target file system.
155	0x009B	Cannot create another thread.
156	0x009C	The recipient process has refused the signal.
157	0x009D	The segment is already discarded and cannot be locked.
158	0x009E	The segment is already unlocked.
159	0x009F	The address for the thread ID is not correct.
160	0x00A0	The argument string passed to DosExecPgm is not correct.
161	0x00A1	The specified path is invalid.
162	0x00A2	A signal is already pending.
164	0x00A4	No more threads can be created in the system.
167	0x00A7	Unable to lock a region of a file.
170	0x00AA	The requested resource is in use.
173	0x00AD	A lock request was not outstanding for the supplied cancel region.
174	0x00AE	The file system does not support atomic changes to the lock type.
180	0x00B4	The system detected a segment number that was not correct.
182	0x00B6	The operating system cannot run %1.
183	0x00B7	Cannot create a file when that file already exists.
186	0x00BA	The flag passed is not correct.
187	0x00BB	The specified system semaphore name was not found.
188	0x00BC	The operating system cannot run %1.
189	0x00BD	The operating system cannot run %1.
190	0x00BE	The operating system cannot run %1.
191	0x00BF	Cannot run %1 in Win32 mode.
192	0x00C0	The operating system cannot run %1.
193	0x00C1	%1 is not a valid Win32 application.
194	0x00C2	The operating system cannot run %1.
195	0x00C3	The operating system cannot run %1.
196	0x00C4	The operating system cannot run this application program.
197	0x00C5	The operating system is not presently configured to run this application.
198	0x00C6	The operating system cannot run %1.
199	0x00C7	The operating system cannot run this application program.
200	0x00C8	The code segment cannot be greater than or equal to 64KB.
201	0x00C9	The operating system cannot run %1.
202	0x00CA	The operating system cannot run %1.
203	0x00CB	The system could not find the environment option that was entered.
205	0x00CD	No process in the command subtree has a signal handler.
206	0x00CE	The filename or extension is too long.

207	0x00CF	The ring 2 stack is in use.
208	0x00D0	The global filename characters, * or ?, are entered incorrectly or too many global filename characters are specified.
209	0x00D1	The signal being posted is not correct.
210	0x00D2	The signal handler cannot be set.
212	0x00D4	The segment is locked and cannot be reallocated.
214	0x00D6	Too many dynamic link modules are attached to this program or dynamic link module.
215	0x00D7	Can't nest calls to LoadModule.
230	0x00EB	The pipe state is invalid.
231	0x00E7	All pipe instances are busy.
232	0x00EB	The pipe is being closed.
233	0x00E9	No process is on the other end of the pipe.
234	0x00EA	More data is available.
240	0x00F0	The session was cancelled.
254	0x00FE	The specified extended attribute name was invalid.
255	0x00FF	The extended attributes are inconsistent.
259	0x0103	No more data is available.
266	0x010A	The Copy API cannot be used.
267	0x010B	The directory name is invalid.
275	0x0113	The extended attributes did not fit in the buffer.
276	0x0114	The extended attribute file on the mounted file system is corrupt.
277	0x0115	The extended attribute table file is full.
278	0x0116	The specified extended attribute handle is invalid.
282	0x011A	The mounted file system does not support extended attributes.
288	0x0120	Attempt to release mutex not owned by caller.
298	0x012A	Too many posts were made to a semaphore.
299	0x012B	Only part of a Read/WriteProcessMemory request was completed.
317	0x013D	The system cannot find message for message number 0x%1 in message file for %2.
487	0x01E7	Attempt to access invalid address.
534	0x0216	Arithmetic result exceeded 32 bits.
535	0x0217	There is a process on other end of the pipe.
536	0x0218	Waiting for a process to open the other end of the pipe.
994	0x03E2	Access to the extended attribute was denied.
995	0x03E3	The I/O operation has been aborted because of either a thread exit or an application request.
996	0x03E4	Overlapped I/O event is not in a signalled state.
997	0x03E5	Overlapped I/O operation is in progress.
998	0x03E6	Invalid access to memory location.
999	0x03E7	Error performing inpage operation.
1001	0x03E9	Recursion too deep, stack overflowed.
1002	0x03EA	The window cannot act on the sent message.
1003	0x03EB	Cannot complete this function.
1004	0x03EC	Invalid flags.
1005	0x03ED	The volume does not contain a recognized file system. Please make sure that all required file system drivers are loaded and that the volume is not corrupt.
1006	0x03EE	The volume for a file has been externally altered such that the opened file is no longer valid.
1007	0x03EF	The requested operation cannot be performed in full-screen mode.
1008	0x03F0	An attempt was made to reference a token that does not exist.
1009	0x03F1	The configuration registry database is corrupt.
1010	0x03F2	The configuration registry key is invalid.
1011	0x03F3	The configuration registry key could not be opened.
1012	0x03F4	The configuration registry key could not be read.
1013	0x03F5	The configuration registry key could not be written.
1014	0x03F6	One of the files in the Registry database had to be recovered by use of a log or alternate copy. The recovery was successful.
1015	0x03F7	The Registry is corrupt. The structure of one of the files that contains Registry data is corrupt, or the system's image of the file in memory is corrupt, or the file could not be recovered because the alternate copy or log was absent or corrupt.

1016	0x03F8	An I/O operation initiated by the Registry failed unrecoverably. The Registry could not read in, or write out, or flush, one of the files that contain the systems image of the Registry.
1017	0x03F9	The system has attempted to load or restore a file into the Registry, but the specified file is not in a Registry file format.
1018	0x03FA	Illegal operation attempted on a Registry key which has been marked for deletion.
1019	0x03FB	System could not allocate the required space in a Registry log.
1020	0x03FC	Cannot create a symbolic link in a Registry key that already has subkeys or values.
1021	0x03FD	Cannot create a stable subkey under a volatile parent key.
1022	0x03FE	A notify change request is being completed and the information is not being returned in the caller's buffer. The caller now needs to enumerate the files to find the changes.
1051	0x041B	A stop control has been sent to a service which other running services are dependent on.
1052	0x041C	The requested control is not valid for this service
1053	0x041D	The service did not respond to the start or control request in a timely fashion.
1054	0x041E	A thread could not be created for the service.
1055	0x041F	The service database is locked.
1056	0x0420	An instance of the service is already running.
1057	0x0421	The account name is invalid or does not exist.
1058	0x0422	The specified service is disabled and cannot be started.
1059	0x0423	Circular service dependency was specified.
1060	0x0424	The specified service does not exist as an installed service.
1061	0x0425	The service cannot accept control messages at this time.
1062	0x0426	The service has not been started.
1063	0x0427	The service process could not connect to the service controller.
1064	0x0428	An exception occurred in the service when handling the control request.
1065	0x0429	The database specified does not exist.
1066	0x042A	The service has returned a service-specific error code.
1067	0x042B	The process terminated unexpectedly.
1068	0x042C	The dependency service or group failed to start.
1069	0x042D	The service did not start due to a logon failure.
1070	0x042E	After starting, the service hung in a start-pending state.
1071	0x042F	The specified service database lock is invalid.
1072	0x0430	The specified service has been marked for deletion.
1073	0x0431	The specified service already exists.
1074	0x0432	The system is currently running with the last-known-good configuration.
1075	0x0433	The dependency service does not exist or has been marked for deletion.
1076	0x0434	The current boot has already been accepted for use as the last-known-good control set.
1077	0x0435	No attempts to start the service have been made since the last boot.
1078	0x0436	The name is already in use as either a service name or a service display name.
1100	0x044C	The physical end of the tape has been reached.
1101	0x044D	A tape access reached a filemark.
1102	0x044E	Beginning of tape or partition was encountered.
1103	0x044F	A tape access reached the end of a set of files.
1104	0x0450	No more data is on the tape.
1105	0x0451	Tape could not be partitioned.
1106	0x0452	When accessing a new tape of a multivolume partition, the current blocksize is incorrect.
1107	0x0453	Tape partition information could not be found when loading a tape.
1108	0x0454	Unable to lock the media eject mechanism.
1109	0x0455	Unable to unload the media.
1110	0x0456	Media in drive may have changed.
1111	0x0457	The I/O bus was reset.
1112	0x0458	No media in drive.
1113	0x0459	No mapping for the Unicode character exists in the target multi-byte code page.
1114	0x045A	A dynamic link library (DLL) initialization routine failed.
1115	0x045B	A system shutdown is in progress.
1116	0x045C	Unable to abort the system shutdown because no shutdown was in progress.
1117	0x045D	The request could not be performed because of an I/O device error.

1118	0x045E	No serial device was successfully initialized. The serial driver will unload.
1119	0x045F	Unable to open a device that was sharing an interrupt request (IRQ) with other devices. At least one other device that uses that IRQ was already opened.
1120	0x0460	A serial I/O operation was completed by another write to the serial port. (The IOCTL_SERIAL_XOFF_COUNTER reached zero.)
1121	0x0461	A serial I/O operation completed because the time-out period expired. (The IOCTL_SERIAL_XOFF_COUNTER did not reach zero.)
1122	0x0462	No ID address mark was found on the floppy disk.
1123	0x0463	Mismatch between the floppy disk sector ID field and the floppy disk controller track address.
1124	0x0464	The floppy disk controller reported an error that is not recognized by the floppy disk driver.
1125	0x0465	The floppy disk controller returned inconsistent results in its registers.
1126	0x0466	While accessing the hard disk, a recalibrate operation failed, even after retries.
1127	0x0467	While accessing the hard disk, a disk operation failed even after retries.
1128	0x0468	While accessing the hard disk, a disk controller reset was needed, but even that failed.
1129	0x0469	Physical end of tape encountered.
1130	0x046A	Not enough server storage is available to process this command.
1131	0x046B	A potential deadlock condition has been detected.
1132	0x046C	The base address or the file offset specified does not have the proper alignment.
1140	0x0474	An attempt to change the system power state was vetoed by another application or driver.
1141	0x0475	The system BIOS failed an attempt to change the system power state.
1150	0x047E	The specified program requires a newer version of Windows.
1151	0x047F	The specified program is not a Windows or MS-DOS program.
1152	0x0480	Cannot start more than one instance of the specified program.
1153	0x0481	The specified program was written for an older version of Windows.
1154	0x0482	One of the library files needed to run this application is damaged.
1155	0x0483	No application is associated with the specified file for this operation.
1156	0x0484	An error occurred in sending the command to the application.
1157	0x0485	One of the library files needed to run this application cannot be found.
1200	0x04B0	The specified device name is invalid.
1201	0x04B1	The device is not currently connected but it is a remembered connection.
1202	0x04B2	An attempt was made to remember a device that had previously been remembered.
1203	0x04B3	No network provider accepted the given network path.
1204	0x04B4	The specified network provider name is invalid.
1205	0x04B5	Unable to open the network connection profile.
1206	0x04B6	The network connection profile is corrupt.
1207	0x04B7	Cannot enumerate a non-container.
1208	0x04B8	An extended error has occurred.
1209	0x04B9	The format of the specified group name is invalid.
1210	0x04BA	The format of the specified computer name is invalid.
1211	0x04BB	The format of the specified event name is invalid.
1212	0x04BC	The format of the specified domain name is invalid.
1213	0x04BD	The format of the specified service name is invalid.
1214	0x04BE	The format of the specified network name is invalid.
1215	0x04BF	The format of the specified share name is invalid.
1216	0x04C0	The format of the specified password is invalid.
1217	0x04C1	The format of the specified message name is invalid.
1218	0x04C2	The format of the specified message destination is invalid.
1219	0x04C3	The credentials supplied conflict with an existing set of credentials.
1220	0x04C4	An attempt was made to establish a session to a network server, but there are already too many sessions established to that server.
1221	0x04C5	The workgroup or domain name is already in use by another computer on the network.
1222	0x04C6	The network is not present or not started.
1223	0x04C7	The operation was cancelled by the user.
1224	0x04C8	The requested operation cannot be performed on a file with a user mapped section open.
1225	0x04C9	The remote system refused the network connection.
1226	0x04CA	The network connection was gracefully closed.

1227	0x04CB	The network transport endpoint already has an address associated with it.
1228	0x04CC	An address has not yet been associated with the network endpoint.
1229	0x04CD	An operation was attempted on a non-existent network connection.
1230	0x04CE	An invalid operation was attempted on an active network connection.
1231	0x04CF	The remote network is not reachable by the transport.
1232	0x04D0	The remote system is not reachable by the transport.
1233	0x04D1	The remote system does not support the transport protocol.
1234	0x04D2	No service is operating at the destination network endpoint on the remote system.
1235	0x04D3	The request was aborted.
1236	0x04D4	The network connection was aborted by the local system.
1237	0x04D5	The operation could not be completed. A retry should be performed.
1238	0x04D6	A connection to the server could not be made because the limit on the number of concurrent connections for this account has been reached.
1239	0x04D7	Attempting to login during an unauthorized time of day for this account.
1240	0x04D8	The account is not authorized to login from this station.
1241	0x04D9	The network address could not be used for the operation requested.
1242	0x04DA	The service is already registered.
1243	0x04DB	The specified service does not exist.
1244	0x04DC	The operation being requested was not performed because the user has not been authenticated.
1245	0x04DD	The operation being requested was not performed because the user has not logged on to the network. The specified service does not exist.
1246	0x04DE	Return that wants caller to continue with work in progress.
1247	0x04DF	An attempt was made to perform an initialization operation when initialization has already been completed.
1248	0x04E0	No more local devices.
1300	0x0514	Not all privileges referenced are assigned to the caller.
1301	0x0515	Some mapping between account names and security IDs was not done.
1302	0x0516	No system quota limits are specifically set for this account.
1303	0x0517	No encryption key is available. A well-known encryption key was returned.
1304	0x0518	The NT password is too complex to be converted to a LAN Manager password. The LAN Manager password returned is a NULL string.
1305	0x0519	The revision level is unknown.
1306	0x051A	Indicates two revision levels are incompatible.
1307	0x051B	This security ID may not be assigned as the owner of this object.
1308	0x051C	This security ID may not be assigned as the primary group of an object.
1309	0x051D	An attempt has been made to operate on an impersonation token by a thread that is not currently impersonating a client.
1310	0x051E	The group may not be disabled.
1311	0x051F	There are currently no logon servers available to service the logon request.
1312	0x0520	A specified logon session does not exist. It may already have been terminated.
1313	0x0521	A specified privilege does not exist.
1314	0x0522	A required privilege is not held by the client.
1315	0x0523	The name provided is not a properly formed account name.
1316	0x0524	The specified user already exists.
1317	0x0525	The specified user does not exist.
1318	0x0526	The specified group already exists.
1319	0x0527	The specified group does not exist.
1320	0x0528	Either the specified user account is already a member of the specified group, or the specified group cannot be deleted because it contains a member.
1321	0x0529	The specified user account is not a member of the specified group account.
1322	0x052A	The last remaining administration account cannot be disabled or deleted.
1323	0x052B	Unable to update the password. The value provided as the current password is incorrect.
1324	0x052C	Unable to update the password. The value provided for the new password contains values that are not allowed in passwords.
1325	0x052D	Unable to update the password because a password update rule has been violated.
1326	0x052E	Logon failure: unknown user name or bad password.
1327	0x052F	Logon failure: user account restriction.
1328	0x0530	Logon failure: account logon time restriction violation.

1329	0x0531	Logon failure: user not allowed to log on to this computer.
1330	0x0532	Logon failure: the specified account password has expired.
1331	0x0533	Logon failure: account currently disabled.
1332	0x0534	No mapping between account names and security IDs was done.
1333	0x0535	Too many local user identifiers (LUIDs) were requested at one time.
1334	0x0536	No more local user identifiers (LUIDs) are available.
1335	0x0537	The subauthority part of a security ID is invalid for this particular use.
1336	0x0538	The access control list (ACL) structure is invalid.
1337	0x0539	The security ID structure is invalid.
1338	0x053A	The security descriptor structure is invalid.
1340	0x053C	The inherited access control list (ACL) or access control entry (ACE) could not be built.
1341	0x053D	The server is currently disabled.
1342	0x053E	The server is currently enabled.
1343	0x053F	The value provided was an invalid value for an identifier authority.
1344	0x0540	No more memory is available for security information updates.
1345	0x0541	The specified attributes are invalid, or incompatible with the attributes for the group as a whole.
1346	0x0542	Either a required impersonation level was not provided, or the provided impersonation level is invalid.
1347	0x0543	Cannot open an anonymous level security token.
1348	0x0544	The validation information class requested was invalid.
1349	0x0545	The type of the token is inappropriate for its attempted use.
1350	0x0546	Unable to perform a security operation on an object which has no associated security.
1351	0x0547	Indicates a Windows NT Server could not be contacted or that objects within the domain are protected such that necessary information could not be retrieved.
1352	0x0548	The security account manager (SAM) or local security authority (LSA) server was in the wrong state to perform the security operation.
1353	0x0549	The domain was in the wrong state to perform the security operation.
1354	0x054A	This operation is only allowed for the Primary Domain Controller of the domain.
1355	0x054B	The specified domain did not exist.
1356	0x054C	The specified domain already exists.
1357	0x054D	An attempt was made to exceed the limit on the number of domains per server.
1358	0x054E	Unable to complete the requested operation because of either a catastrophic media failure or a data structure corruption on the disk.
1359	0x054F	The security account database contains an internal inconsistency.
1360	0x0550	Generic access types were contained in an access mask which should already be mapped to non-generic types.
1361	0x0551	A security descriptor is not in the right format (absolute or self-relative).
1362	0x0552	The requested action is restricted for use by logon processes only. The calling process has not registered as a logon process.
1363	0x0553	Cannot start a new logon session with an ID that is already in use.
1364	0x0554	A specified authentication package is unknown.
1365	0x0555	The logon session is not in a state that is consistent with the requested operation.
1366	0x0556	The logon session ID is already in use.
1367	0x0557	A logon request contained an invalid logon type value.
1368	0x0558	Unable to impersonate via a named pipe until data has been read from that pipe.
1369	0x0559	The transaction state of a Registry subtree is incompatible with the requested operation.
1370	0x055A	An internal security database corruption has been encountered.
1371	0x055B	Cannot perform this operation on built-in accounts.
1372	0x055C	Cannot perform this operation on this built-in special group.
1373	0x055D	Cannot perform this operation on this built-in special user.
1374	0x055E	The user cannot be removed from a group because the group is currently the user's primary group.
1375	0x055F	The token is already in use as a primary token.
1376	0x0560	The specified local group does not exist.
1377	0x0561	The specified account name is not a member of the local group.
1378	0x0562	The specified account name is already a member of the local group.
1379	0x0563	The specified local group already exists.
1380	0x0564	Logon failure: the user has not been granted the requested logon type at this computer.

1381	0x0565	The maximum number of secrets that may be stored in a single system has been exceeded.
1382	0x0566	The length of a secret exceeds the maximum length allowed.
1383	0x0567	The local security authority database contains an internal inconsistency.
1384	0x0568	During a logon attempt, the user's security context accumulated too many security IDs.
1385	0x0569	Logon failure: the user has not been granted the requested logon type at this computer.
1386	0x056A	A cross-encrypted password is necessary to change a user password.
1387	0x056B	A new member could not be added to a local group because the member does not exist.
1388	0x056C	A new member could not be added to a local group because the member has the wrong account type.
1389	0x056D	Too many security IDs have been specified.
1390	0x056E	A cross-encrypted password is necessary to change this user password.
1391	0x056F	Indicates an ACL contains no inheritable components
1392	0x0570	The file or directory is corrupt and non-readable.
1393	0x0571	The disk structure is corrupt and non-readable.
1394	0x0572	There is no user session key for the specified logon session.
1395	0x0573	The service being accessed is licensed for a particular number of connections. No more connections can be made to the service at this time because there are already as many connections as the service can accept.
1400	0x0578	Invalid window handle.
1401	0x0579	Invalid menu handle.
1402	0x057A	Invalid cursor handle.
1403	0x057B	Invalid accelerator table handle.
1404	0x057C	Invalid hook handle.
1405	0x057D	Invalid handle to a multiple-window position structure.
1406	0x057E	Cannot create a top-level child window.
1407	0x057F	Cannot find window class.
1408	0x0580	Invalid window, belongs to other thread.
1409	0x0581	Hot key is already registered.
1410	0x0582	Class already exists.
1411	0x0583	Class does not exist.
1412	0x0584	Class still has open windows.
1413	0x0585	Invalid index.
1414	0x0586	Invalid icon handle.
1415	0x0587	Using private DIALOG window words.
1416	0x0588	The listbox identifier was not found.
1417	0x0589	No wildcards were found.
1418	0x058A	Thread does not have a clipboard open.
1419	0x058B	Hot key is not registered.
1420	0x058C	The window is not a valid dialog window.
1421	0x058D	Control ID not found.
1422	0x058E	Invalid message for a combo box because it does not have an edit control.
1423	0x058F	The window is not a combo box.
1424	0x0590	Height must be less than 256.
1425	0x0591	Invalid device context (DC) handle.
1426	0x0592	Invalid hook procedure type.
1427	0x0593	Invalid hook procedure.
1428	0x0594	Cannot set non-local hook without a module handle.
1429	0x0595	This hook procedure can only be set globally.
1430	0x0596	The journal hook procedure is already installed.
1431	0x0597	The hook procedure is not installed.
1432	0x0598	Invalid message for single-selection listbox.
1433	0x0599	LB_SETCOUNT sent to non-lazy listbox.
1434	0x059A	This list box does not support tab stops.
1435	0x059B	Cannot destroy object created by another thread.
1436	0x059C	Child windows cannot have menus.
1437	0x059D	The window does not have a system menu.

1438	0x059E	Invalid message box style.
1439	0x059F	Invalid system-wide (SPI_*) parameter.
1440	0x05A0	Screen already locked.
1441	0x05A1	All handles to windows in a multiple-window position structure must have the same parent.
1442	0x05A2	The window is not a child window.
1443	0x05A3	Invalid GW_* command.
1444	0x05A4	Invalid thread identifier.
1445	0x05A5	Cannot process a message from a window that is not a multiple document interface (MDI) window.
1446	0x05A6	Popup menu already active.
1447	0x05A7	The window does not have scroll bars.
1448	0x05A8	Scroll bar range cannot be greater than 0x7FFF.
1449	0x05A9	Cannot show or remove the window in the way specified.
1450	0x05AA	Insufficient system resources exist to complete the requested service.
1451	0x05AB	Insufficient system resources exist to complete the requested service.
1452	0x05AC	Insufficient system resources exist to complete the requested service.
1453	0x05AD	Insufficient quota to complete the requested service.
1454	0x05AE	Insufficient quota to complete the requested service.
1455	0x05AF	The paging file is too small for this operation to complete.
1456	0x05B0	A menu item was not found.
1500	0x05DC	The event log file is corrupt.
1501	0x05DD	No event log file could be opened, so the event logging service did not start.
1502	0x05DE	The event log file is full.
1503	0x05DF	The event log file has changed between reads.
1700	0x06A4	The string binding is invalid.
1701	0x06A5	The binding handle is not the correct type.
1702	0x06A6	The binding handle is invalid.
1703	0x06A7	The RPC protocol sequence is not supported.
1704	0x06A8	The RPC protocol sequence is invalid.
1705	0x06A9	The string universal unique identifier (UUID) is invalid.
1706	0x06AA	The endpoint format is invalid.
1707	0x06AB	The network address is invalid.
1708	0x06AC	No endpoint was found.
1709	0x06AD	The timeout value is invalid.
1710	0x06AE	The object universal unique identifier (UUID) was not found.
1711	0x06AF	The object universal unique identifier (UUID) has already been registered.
1712	0x06B0	The type universal unique identifier (UUID) has already been registered.
1713	0x06B1	The RPC server is already listening.
1714	0x06B2	No protocol sequences have been registered.
1715	0x06B3	The RPC server is not listening.
1716	0x06B4	The manager type is unknown.
1717	0x06B5	The interface is unknown.
1718	0x06B6	There are no bindings.
1719	0x06B7	There are no protocol sequences.
1720	0x06B8	The endpoint cannot be created.
1721	0x06B9	Not enough resources are available to complete this operation.
1722	0x06BA	The RPC server is unavailable.
1723	0x06BB	The RPC server is too busy to complete this operation.
1724	0x06BC	The network options are invalid.
1725	0x06BD	There is not a remote procedure call active in this thread.
1726	0x06BE	The remote procedure call failed.
1727	0x06BF	The remote procedure call failed and did not execute.
1728	0x06C0	A remote procedure call (RPC) protocol error occurred.
1730	0x06C2	The transfer syntax is not supported by the RPC server.
1732	0x06C4	The universal unique identifier (UUID) type is not supported.

1733	0x06C5	The tag is invalid.
1734	0x06C6	The array bounds are invalid.
1735	0x06C7	The binding does not contain an entry name.
1736	0x06C8	The name syntax is invalid.
1737	0x06C9	The name syntax is not supported.
1739	0x06CB	No network address is available to use to construct a universal unique identifier (UUID).
1740	0x06CC	The endpoint is a duplicate.
1741	0x06CD	The authentication type is unknown.
1742	0x06CE	The maximum number of calls is too small.
1743	0x06CF	The string is too long.
1744	0x06D0	The RPC protocol sequence was not found.
1745	0x06D1	The procedure number is out of range.
1746	0x06D2	The binding does not contain any authentication information.
1747	0x06D3	The authentication service is unknown.
1748	0x06D4	The authentication level is unknown.
1749	0x06D5	The security context is invalid.
1750	0x06D6	The authorization service is unknown.
1751	0x06D7	The entry is invalid.
1752	0x06D8	The server endpoint cannot perform the operation.
1753	0x06D9	There are no more endpoints available from the endpoint mapper.
1754	0x06DA	No interfaces have been exported.
1755	0x06DB	The entry name is incomplete.
1756	0x06DC	The version option is invalid.
1757	0x06DD	There are no more members.
1758	0x06DE	There is nothing to unexport.
1759	0x06DF	The interface was not found.
1760	0x06E0	The entry already exists.
1761	0x06E1	The entry is not found.
1762	0x06E2	The name service is unavailable.
1763	0x06E3	The network address family is invalid.
1764	0x06E4	The requested operation is not supported.
1765	0x06E5	No security context is available to allow impersonation.
1766	0x06E6	An internal error occurred in a remote procedure call (RPC).
1767	0x06E7	The RPC server attempted an integer division by zero.
1768	0x06E8	An addressing error occurred in the RPC server.
1769	0x06E9	A floating-point operation at the RPC server caused a division by zero.
1770	0x06EA	A floating-point underflow occurred at the RPC server.
1771	0x06EB	A floating-point overflow occurred at the RPC server.
1772	0x06EC	The list of RPC servers available for the binding of auto handles has been exhausted.
1773	0x06ED	Unable to open the character translation table file.
1774	0x06EE	The file containing the character translation table has fewer than 512 bytes.
1775	0x06EF	A null context handle was passed from the client to the host during a remote procedure call.
1777	0x06F1	The context handle changed during a remote procedure call.
1778	0x06F2	The binding handles passed to a remote procedure call do not match.
1779	0x06F3	The stub is unable to get the remote procedure call handle.
1780	0x06F4	A null reference pointer was passed to the stub.
1781	0x06F5	The enumeration value is out of range.
1782	0x06F6	The byte count is too small.
1783	0x06F7	The stub received bad data.
1784	0x06F8	The supplied user buffer is not valid for the requested operation.
1785	0x06F9	The disk media is not recognized. It may not be formatted.
1786	0x06FA	The workstation does not have a trust secret.

1787	0x06FB	The SAM database on the Windows NT Server does not have a computer account for this workstation trust relationship.
1788	0x06FC	The trust relationship between the primary domain and the trusted domain failed.
1789	0x06FD	The trust relationship between this workstation and the primary domain failed.
1790	0x06FE	The network logon failed.
1791	0x06FF	A remote procedure call is already in progress for this thread.
1792	0x0700	An attempt was made to logon, but the network logon service was not started.
1793	0x0701	The user's account has expired.
1794	0x0702	The redirector is in use and cannot be unloaded.
1795	0x0703	The specified printer driver is already installed.
1796	0x0704	The specified port is unknown.
1797	0x0705	The printer driver is unknown.
1798	0x0706	The print processor is unknown.
1799	0x0707	The specified separator file is invalid.
1800	0x0708	The specified priority is invalid.
1801	0x0709	The printer name is invalid.
1802	0x070A	The printer already exists.
1803	0x070B	The printer command is invalid.
1804	0x070C	The specified datatype is invalid.
1805	0x070D	The Environment specified is invalid.
1806	0x070E	There are no more bindings.
1807	0x070F	The account used is an interdomain trust account. Use your global user account or local user account to access this server.
1808	0x0710	The account used is a Computer Account. Use your global user account or local user account to access this server.
1809	0x0711	The account used is a server trust account. Use your global user account or local user account to access this server.
1810	0x0712	The name or security ID (SID) of the domain specified is inconsistent with the trust information for that domain.
1811	0x0713	The server is in use and cannot be unloaded.
1812	0x0714	The specified image file did not contain a resource section.
1813	0x0715	The specified resource type can not be found in the image file.
1814	0x0716	The specified resource name can not be found in the image file.
1815	0x0717	The specified resource language ID cannot be found in the image file.
1816	0x0718	Not enough quota is available to process this command.
1817	0x0719	No interfaces have been registered.
1818	0x071A	The server was altered while processing this call.
1819	0x071B	The binding handle does not contain all required information.
1820	0x071C	Communications failure.
1821	0x071D	The requested authentication level is not supported.
1822	0x071E	No principal name registered.
1823	0x071F	The error specified is not a valid Windows RPC error code.
1824	0x0720	A UUID that is valid only on this computer has been allocated.
1825	0x0721	A security package specific error occurred.
1826	0x0722	Thread is not cancelled.
1827	0x0723	Invalid operation on the encoding/decoding handle.
1828	0x0724	Incompatible version of the serializing package.
1829	0x0725	Incompatible version of the RPC stub.
1898	0x076A	The group member was not found.
1899	0x076B	The endpoint mapper database could not be created.
1900	0x076C	The object universal unique identifier (UUID) is the nil UUID.
1901	0x076D	The specified time is invalid.
1902	0x076E	The specified Form name is invalid.
1903	0x076F	The specified Form size is invalid.
1904	0x0770	The specified Printer handle is already being waited on.
1905	0x0771	The specified Printer has been deleted.
1906	0x0772	The state of the Printer is invalid.
1907	0x0773	The user must change his password before he logs on the first time.
1908	0x0774	Could not find the domain controller for this domain.

1909	0x0775	The referenced account is currently locked out and may not be logged on to.
2000	0x07D0	The pixel format is invalid.
2001	0x07D1	The specified driver is invalid.
2002	0x07D2	The window style or class attribute is invalid for this operation.
2003	0x07D3	The requested metafile operation is not supported.
2004	0x07D4	The requested transformation operation is not supported.
2005	0x07D5	The requested clipping operation is not supported.
2202	0x089A	The specified username is invalid.
2250	0x08CA	This network connection does not exist.
2401	0x0961	This network connection has files open or requests pending.
2402	0x0962	Active connections still exist.
2404	0x0964	The device is in use by an active process and cannot be disconnected.
3000	0x0BB8	The specified print monitor is unknown.
3001	0x0BB9	The specified printer driver is currently in use.
3002	0x0BBA	The spool file was not found.
3003	0x0BBB	A StartDocPrinter call was not issued.
3004	0x0BBC	An AddJob call was not issued.
3005	0x0BBD	The specified print processor has already been installed.
3006	0x0BBE	The specified print monitor has already been installed.
4000	0x0FA0	WINS encountered an error while processing the command.
4001	0x0FA1	The local WINS can not be deleted.
4002	0x0FA2	The importation from the file failed.
4003	0x0FA3	The backup Failed. Was a full backup done before ?
4004	0x0FA4	The backup Failed. Check the directory that you are backing the database to.
4005	0x0FA5	The name does not exist in the WINS database.
4006	0x0FA6	Replication with a non-configured partner is not allowed.
6118	0x17E6	The list of servers for this workgroup is not currently available

Flags - Define Processing Tables (rcabe, dcabe)

The following flags can be used within filePro menus or from the command line with program "rcabe" or "dcabe".

Syntax: dcabe filename flags

-C	Tokenize the specified processing table(s) without going into interactive mode.
-CI	Same as -C, but the user is put into interactive mode if an error occurred.
-CA	Tokenize all .prc tables with one request.
-CIA	Tokenize all .prc tables, but the user is put into interactive mode if an error occurred.
-CS	With ABE set to ASCII, -cs will recompile all processing tables in ASCII. Automatic except when a creation password is required.
-T nnnnn	Sets scan/output processing tokenization table size.
-TY nnnnn	Sets automatic processing tokenization table size.
-Y name	Sets the automatic processing table name if other than "automatic" (UNIX) "auto" (DOS, Windows).

Flags - Directory (ddir & dprodir)

The following flags can be used within filePro menus or from the command line with program "dprodir".

Syntax: dprodir filename flags or ddir filename flags

-A	Delete entire file including formats & qualifiers.
-Ftype name name ...	deletes individual formats specified by format type & format name. (E)dit, (S)creen, (P)rocessing table, (O)utput, (B)rowse, (T)oken table, browse (L)ookup. NOTE: the L code deletes ALL of the browse lookup formats in the file.
-H "heading"	Screen heading (must be used with another flag).
-I n	Deletes an index. "n" is index letter or number.
-KA	Deletes all data in key & data segments; empties all indexes for the file.
-K	Deletes all data in key & data segments for main file only, empties indexes. (Use "-M qual" to delete a qualifier's key and data, and empty its indexes.)
-L	Deletes the lockfile.
-M qual	Sets the qualifier name.
-MD	Prompts user for a qualifier.
-MQ "msg"	Prompts user for qualifier using "msg".

Flags - Expand Files (dexpand)

The following flags can be used within filePro menus or from the command line with program "dexpand".

Syntax: dexpand filename flags

-H "heading"	Screen heading.
-M name	Uses the given qualifier.
-MD	Prompts user for a qualifier.
-MQ "msg"	Prompts user for qualifier using "msg".
Switch	Allows users to check free disk space and switch drives - must occur before the heading.
Nnn	Number of records to add.

Flags - Index Maintenance (dxmaint)

Syntax: dxmaint filename flags

The following flags can be used within filePro menus or from the command line with program "dxmaint".

-C comment	Adds a comment to an index
-E	Exit to user menu upon completion.
-H "heading"	Screen heading.
-KN	Save with rebuild. (default) Version 5.0.6 or greater
-KY	Save without rebuild. Version 5.0.6 or greater
-LN	Exclude from lists. Note: Version 5.0.6 or greater
-LY	Include in lists. Note: Version 5.0.6 or greater
-M name	Uses the given qualifier.
-MD	Prompts user for a qualifier.
-MQ "msg"	Prompts user for qualifier using "msg".
-MN	Hide [NONE] qualifier from the qualifier list.
-On	Indicates the output index name - "n" can be any automatic or demand index
-R	Rebuilds the index on the same key(s).
-RA	Rebuilds all automatic indexes in named file.
-RF n,l,o:n,l,o. ...	Rebuilds an index on a specified field, length & sort order (a=ascending, d=descending) where n = field Number, l = Length and o = Order of sort. Example: dxmaint filename -RF 1,10,d:2,5 -Oa Builds index a using field 1, length of "10" in Descending order and then on field 2, length of "5".
-RF @filename	Has been enhanced to allow you to specify a filename of a file that contains the sort order you want to use
-SX	(Version 5.8.03) - Sets the selection set for automatic indexes An example is: dxmaint testfile -RF 1,6 -E -SX "W and X":W,12,ne,98:X,12,ne,99 -OK
(Demand indexes only)	
-A	Selects all records.
-In	Builds index using index "n" as input.
-N	No sorting.
-S name	Use selection set "name".
-Xn	Use same sort criteria as index "n".
(*NIX only)	
-BG	Build index in the background.
-BS	Suppresses "background task completed" message when -BG option is being used.

Flags - IUA (rclerk, dclerk)

The following flags can be used within filePro menus or from the command line with program "rclerk" or "dclerk".

Syntax: dclerk filename flags

-B	Turns browse on - can be used with XI or XS.
-B name	Turns browse on using the given browse format.
-BD name	Like -B, but also allows switching formats & format maintenance (create,update,delete).
-BX	Turns browse off and removes option 5.
-D	Suppresses all bottom-of-the-screen messages.
-DB	Turns on processing debugger.
-DE	Turns off only the @entsel prompts, leaving the update mode prompts intact for GUI.
-DL	Forces filePro to display the prompts even if xkeys are included. Note: the brwlook xkey prompts have been changed back to the old behavior i.e. if any xkeys are listed, filePro does not put up the default brwlook prompts.
-DM	To remove Index Mode label from screen Version 6.0.02
-DU	Turns off only the update mode prompts, leaving the @entsel prompts intact for GUI.
-FP filename	Run a specified processing table without having an output format associated with it.
-FX	Disables the format selection window.
-H "heading"	Screen heading.
-J	Turns indexed scan on for one operation.
-JN	Turns indexed scan off for one operation.
-LX	Disables creation, deletion and modification of browse lookups.
-M name	Uses the given qualifier.
-MD	Prompts for a qualifier using default prompt.
-MQ "msg"	Prompts for qualifier using "msg" as prompt.
-MN	Hide [NONE] qualifier from the qualifier list.
-N	Puts user in update mode on each selected record - used with X flags.
-P name	Send printer output to "name".
-PC type	Sets the printer type.
-PN name	Sets the printer name.
-PQ	Prompts user for printing to printer, file or display.
-PV	Shows printer output on screen, one page at a time.
-R "value"	Sets value for system-maintained field @PM.
-RO	Designates files as read only.
-RW data	Works like the -R flag, i.e., passes a variable to a processing table. The processing table uses the @PW system-maintained field to retrieve the variable's contents.
-RX data	Same as -RW but uses the @PX system-maintained field.
-RY data	Same as -RW but uses the @PY system-maintained field.
-RZ data	Same as -RW but uses the @PZ system-maintained field.
-Sn	Screen to use.
-T nnnnn	Sets input processing tokenization table size.
-TF nnnnn	Sets output processing tokenization table size.
-TY nnnnn	Sets automatic processing tokenization table size.
-XA	User goes immediately into add records mode.
-XE	Skips IUA menu, goes directly to @ENTSEL processing in input processing.
-XI	User goes immediately to index-mode prompt.
-Xin	User goes immediately to index-mode on "index.n".
-XS name	User enters the file via given selection set.
-Y name	Choose alternate automatic processing table. Use nonexistent table name (or "") to skip automatic processing.
-Z name	Choose alternate input processing table. Use nonexistent table name (or "") to skip processing.

(UNIX/Linux only)

-PT

Print to local printer if codes are in termcap entry, otherwise print to the screen.

Flags - Request Output (rreport, dreport)

The following flags can be used within filePro menus or from the command line with program "rreport" or "dreport".

Syntax: dreport filename flags

-A	All records.
-AS n	Start processing beginning at record number n instead of record #1.
-DB	Activates the debugger for processing table.
-F name	Selects the output format.
-FP filename	Run a specified output-processing table without having an output format associated with it.
-FO path/filename	*report .out to use when not in the local directory
-H "heading"	Screen heading.
-In	Selects an index.
-M name	Uses the given qualifier.
-MD	Prompts user for a qualifier.
-MQ "msg"	Prompts user for qualifier using "msg".
-MN	Hide [NONE] qualifier from the qualifier list.
-N	No sorting.
-On	Saves the selected records as demand index n.
-P name	Selects the printer name.
-P file	Selects the file to put the output.
-PC type	Sets the printer type.
-PN name	Sets the printer name.
-PQ	Prompts user for printing to printer, file or display.
-PV	Shows printer output on the screen a page at a time.
-R "value"	Sets value for system-maintained field @PM.
-RF nnn,lll[,ad]	Specify a sort order to be used. You can either specify the sort order directly on the command line or point to a file that contains the sort order.
-RF @filename	
-RH	disable the automatic record number reporting in the middle of the screen Version 6.0.02
-RO	Designates files as read only.
-RW data	Works like the -R flag, i.e., passes a variable to a processing table. The processing table uses the @PW system-maintained field to retrieve the variable's contents.
-RX data	Same as -RW but uses the @PX system-maintained field.
-RY data	Same as -RW but uses the @PY system maintained field.
-RZ data	Same as -RW but uses the @PZ system maintained field.
-S name	Sets the selection set.
-SR n	Run on a single specific record number n.
-SX	Prevent user from saving revised selection sets to disk.
-T nnnn	Sets scan/output process tokenization table size.
-TY nnnnn	Sets automatic process tokenization table size.
-V name	Uses named sort/selection table.
-W	Prompt user between pages. (Multi-user needs -P also).
-X	Halt at sort screen, needs -F & either -S or -A.
-Y name	Choose alternate automatic processing table. Use nonexistent table name to skip automatic processing.
-Z name	Choose alternate output processing table. Use nonexistent table name to skip output processing.
	(LINUX/UNIX/XENIX/Network)
-U	Prevents users from being locked out of a file when "Request Output" is processing records. (LINUX/UNIX/XENIX/Network versions) Warning: Do not use when; "Request Output" is posting data; do not let a user and "Request Output" simultaneously update the same record.
	(UNIX only)
-BG	Generate report in the background.
-BS	Suppress the "background task completed" message when -BG option is used.

HTML Functions

Version Ref: 4.8 (not included in filePro Lite)

5.0 (enhancements)

Syntax:

HTML id :Tag_Code Tag_Value :Attribute Value	
"id"	can be used to create multiple HTML files.
"Tag_Code"	is a filePro code similar to a HTML "TAG".
"Tag_Value"	a value related to the Tag_Code.
"Attribute"	filePro code to identify HTML attributes.
"Value"	the attribute value.

Description:

Create HTML files using filePro HTML functions.

Example:

```
HTML "1" :CR "fp.html" :TI "filePro Page" :BI "fpback.gif"
```

The above example creates a single html document "fp.html" with a title of "filePro Page" with :BI attribute (background image) named "fpback.gif".

"1" - Create HTML id = "1" :CR - Create a document "fp.html"

:TI - Title "filePro Page" :BI - Background Image "fpback.gif"

The filePro HTML functions are presented in table format to show related functions and attributes that can be used.

<u>HTML Function</u>	<u>TAG</u>	<u>Tag Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u># of Values</u>
----------------------	------------	-----------------	------------------	-------------	--------------	--------------------

Terms Description

HTML Function	The standard HTML function name
TAG	HTML TAG returned for the filePro Tag Code.
Tag Code	filePro code related to a HTML function.
Tag Value	"Literal" or filePro variable.
Attribute	filePro code to specify HTML attributes.
Value	HTML attribute value as a "Literal" or filePro variable.
# of Val	Number of values allowed per attribute.

Legend for # of val (values)

0 - No values allowed.

1 - Exactly one value.

? - Single value optional.

+ - Text, comma-delimited list of values.

* - Text, comma-delimited list of values optional.

Note: filePro uses "Tag Codes" for functions with no corresponding HTML <TAG> such as :CR (Create file), :CL (Close file).

HTML(Address)

Version Ref: 5.0 (not included in filePro Lite)

Syntax:

HTML :AD

<i><u>HTML</u></i> <i><u>Function</u></i>	<i><u>TAG</u></i>	<i><u>Tag</u></i> <i><u>Code</u></i>	<i><u>Tag Value</u></i>	<i><u>Attr</u></i>	<i><u>Value</u></i>	<i><u>#of</u></i> <i><u>Values</u></i>
Address	<ADDRESS>	:AD	"address"	<		

Note: If value given, its text is used and <ADDRESS> is automatically closed.

HTML(Anchor)

Version Ref: 4.8 (not included in filePro Lite)

Syntax:

HTML:AN"Any Text" :HR

<u>HTML Function</u>	<u>TAG</u>	<u>Tag Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u>#of Values</u>
Anchor	<A>	:AN	Optional	:HR	Href	1
			Text	:TA	Table	1
5.0				:RL	Rel	1
5.0				:RV	Rev	1
5.0				:TI	Title	1
				:NA	Name	1
Close Anchor		:AN-	See Note			

Note:

If you do not include a "Tag_value" after "Tag_Code" :AN, you must close the anchor with an HTML :AN- command. When a Tag_value is used, filePro will automatically return the closing code after the "Tag_value".

HTML(Area)

Version Ref: 5.0 (not included in filePro Lite)

Syntax:

```
HTML:AR:HR"Image.bmp"
```

The :AR tag defines each "hotzone" region within tag :MA

<u>HTML Function</u>	<u>TAG</u>	<u>Tag Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u># of Values</u>
Area	<AREA>	:AR	<none>	:NA	Name	1
				:HR	HRef	1
				:SH	Shape	1
				:CO	Coord.	+
				:NO	No HRef	0
				:AL	Alternate	1

Description:

The :HR attribute specifies the URL for the destination that should be chosen if this area was selected. If you specify :NO instead, this area won't do anything.

:SH and :CO define the actual region.

:SH can be a rectangle, circle or polygon and :CO should contain a set of coordinates describing that shape. This is done with a comma separated list of numbers, enclosed in quotes. The syntax for :CO depends on what shape you choose.

rect - rectangle

A rectangle has four coordinates. The first specifies the top left corner, and the second the bottom right corner of the rectangle.

Example:

:SH="rect":CO="0,0,9,9" would specify a rectangle of 10x10 pixels, starting in the top left corner of the image.

circle - circle

A circle is defined by its center and radius. The :CO attribute first specifies the coordinates of the center, and then the radius of the circle, in pixels.

Example

:SH=circle:CO="10,10,5" would specify a circle with radius 5 at location (10,10) in the image.

poly - polygon

A polygon is built up by a list of coordinates. They are all connected in the order you present, and the last coordinate pair is connected to the first. This way you can build arbitrary figures.

Example:

:SH=poly:CO="10,50,15,20,20,50" would specify a triangle,

:AL is used by text browsers to present the URLs in the imagemap in a more readable fashion. If you leave those off, the browser can only display the "bare" URLs. The ALT text is required if you want your document to be valid.

HTML(Base)

Version Ref: 5.0 (not included in filePro Lite)

Syntax:

HTML :BA :HR <http://www.fptech.com>

<u>HTML Function</u>	<u>TAG</u>	<u>Tag Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u># of Values</u>
Base	<BASE>	:BA	<none>	:HR	Href	1

Description:

Can be used to record the document's location in the form of an absolute URL, which can be used to resolve a relative URL if the document is not accessed in its original location.

Example:

<IMG_SRC='../images/fplogo.gif'> would result in <http://www.fptech.com/././images/fplogo.gif>

Note: The relative URL is added to the BASE, so if you only need to specify what is to be added to the BASE for the relative URL's.

HTML(Blockquote)

Version Ref: 5.0 (not included in filePro Lite)

Syntax:

HTML :BQ

<i><u>HTML</u></i>	<i><u>TAG</u></i>	<i><u>Tag</u></i>	<i><u>Tag Value</u></i>	<i><u>Attr</u></i>	<i><u>Value</u></i>	<i><u># of</u></i> <i><u>Values</u></i>
<i><u>Function</u></i>		<i><u>Code</u></i>				
Blockquote		:BQ	Optional	<	<none>	

Description:

Allows you to include quoted text from another author.

Notes:

If a value is given, its text is used and </BLOCKQUOTE> automatically closes <BLOCKQUOTE>.

If you are quoting more than a few lines from a document, use a :BQ to indicate this. Block quotations are often rendered with indented margins, and possibly in italics, although a rendering with the standard quotation symbol for E-mail, ">", is of course also possible.

HTML(Body)

Version Ref: 4.8 (not included in filePro Lite)

Syntax:

HTML :BO "Any text" :BI "filepro.gif"

<u>HTML Function</u>	<u>TAG</u>	<u>Tag Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u># of Values</u>
/HEAD	<BODY>	:BO	<none>	:BI	BG Image	1
<BODY>		:BC	<none> <none>	<> :TE	BGColor Text Color	1 1
				:LI	Link Color	1
				:VL	Visit Link Color	1
				:AL	Active Link Color	1
				:FR	Frame	0
				:ZZ	Your Own Text	+

HTML (Caption, Frameset)

Version Ref: 5.0 (not included in filePro Lite)

Syntax:

HTML:TC

HTML:FS

<u>HTML Function</u>	<u>TAG</u>	<u>Tag Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u>#of Values</u>
Caption	<CAPTION>	:TC	Optional Text	:AL	Align	1
Frameset	<FRAMESET>	:FS	<none>	:RO :CO :FB :BO :BC	Rows Columns Frame Border Border Border Color	+ + 1 ? 1

HTML(Center)

Version Ref: 5.0 (not included in filePro Lite)

Syntax:

HTML :CE"Text"

<u>HTML Function</u>	<u>TAG</u>	<u>Tag Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u>#of Values</u>
Center	<CENTER>	:CE	Optional	<	<none>	

Note: If value given, its text is used, and </CENTER> automatically closed.

HTML (Comments)

Version Ref: 4.8 (not included in filePro Lite)

Syntax:

HTML:CO "This is a comment"

<u>HTML</u> <u>Function</u>	<u>TAG</u>	<u>Tag</u> <u>Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u># of</u> <u>Values</u>
Comments	<! ->	:CO	Optional text	◇		

HTML (Create)

Version Ref: 4.8 (not included in filePro Lite)

HTML(Close)

Syntax:

HTML :CR "test.html" :T1 "Test Page"

HTML :CL

<u>HTML Function</u>	<u>TAG</u>	<u>Tag Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u># of Values</u>
Create	<HTML> <HEAD>	:CR	"Title Text"	:T1	"Test Page"	1
Close	</BODY> </HTML>	:CL	<none>	<		
5.0				:DT	Document Type	1
				:TX	Text	+

Example:

:TA- (Closes a table)

Note: Many of the HTML commands allow you to explicitly CLOSE the command by using the command followed by using a "-" [dash].

HTML(Definition Term, Definition Data)

Version Ref: 5.0 (not included in filePro Lite)

Syntax:

HTML :LI

<u>HTML</u> <u>Function</u>	<u>TAG</u>	<u>Tag</u> <u>Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u># of</u> <u>Values</u>
Mark List		:LI	Optional	:TT :VA	Type Value	1 1

Description:

The LI tag is used to mark list items within a list. When the list used is an "ordered list", the :LI tag will be rendered with a number. The appearance of that number can be controlled with the :TY attribute. Similarly, inside an unordered list the type of bullet that is displayed can be controlled with :TY.

HTML(Divisions)

Version Ref: 5.0 (not included in filePro Lite)

Syntax:

HTML :DI "Text" :AL "center"

<u>HTML Function</u>	<u>TAG</u>	<u>Tag Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u># of Values</u>
Divisions	<DIV>	:DI	Optional	:AL	Align	1

Description:

The :DI tag is used to mark up divisions in a document. It can enclose paragraphs, headers and other block elements. Currently, you can only use it to set the default alignment for all enclosed block elements.

Note: If value given, its text is used, and </DIV> is automatically closed.

HTML(Font)

Version Ref: 5.0 (not included in filePro Lite)

Syntax:

```
HTML :FN "This is big text" :SI "+2" :CO "ffffff"
```

<u>HTML Function</u>	<u>TAG</u>	<u>Tag Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u># of Values</u>
Font		:FN	Optional Text	:CO	Color	1
				:SI	Size	1
				:TX	Any text	+

Description:

Controls font color and size.

Note: If the optional value is given, it is output as text, and a closing is automatically generated. The following generate identical output:

Examples:

Method 1

```
HTML :FN :SI "+2"
```

```
HTML :TX "This is big text."
```

```
HTML :FN
```

Method 2

```
HTML :FN "This is big text." :SI "+2"
```

Result

```
<FONT SIZE="+2">This is big text.</FONT>
```

HTML (Form)

Version Ref: 5.0 (not included in filePro Lite)

Syntax:

HTML:FO

<u>HTML Function</u>	<u>TAG</u>	<u>Tag Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u>#of Values</u>
FORM	<Form>	:FO	<none>	:AC	Action	1
5.0				:EN	Encode	1
				:ME	Method	1
				:NA	Name	1
				:ZZ	Other	+

HTML (Frame)

Version Ref: 4.8 (not included in filePro Lite)

Syntax:

HTML:FR

~~HTML:FR~~

HTML:NF

<u>HTML Function</u>	<u>TAG</u>	<u>Tag Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u>#of Values</u>
Frame	<FRAME>	:FR	<none>	:SR	Source	1
				:MW	Margin Width	1
				:MH	Margin Height	1
				:SC	Scroll Flag	?
				:FB	Frame Border	1
				:BC	Border Color	1
Close Frame	</FRAME>	:FR-				
No Frame	<NO FRAME>	:NF	<none>		<none>	

HTML (Header)

Version Ref: 4.8 (not included in filePro Lite)

Syntax:

HTML:TH

<u>HTML Function</u>	<u>TAG</u>	<u>Tag Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u>#of Values</u>
Header	<HEADER>	:TH	<none>	:AL	Alignment	1
				:VL	Vertical Alignment	1
				:NO		0
				:CO	Colspan	1
				:RO	Rowspan	1
				:WI	Width	1
				:BG	Backgmd Color	1

HTML (Heading Text)

Version Ref: 4.8 (not included in filePro Lite)

Syntax:

```
HTML :H1 "This is a header" :AL "CENTER"
```

<i><u>HTML Function</u></i>	<i><u>TAG</u></i>	<i><u>Tag Code</u></i>	<i><u>Tag Value</u></i>	<i><u>Attr</u></i>	<i><u>Value</u></i>	<i><u># of Values</u></i>
Header	<H1>	:H1	Optional Text	:AL	Align	1
	Thru <H6>	:H6				

Description:

Allows you to control header text alignment for six levels of header text.

Notes:

- 1) including the "Tag_value" will cause filePro to return </H1> at the end of the header line.
- 2) To include multiple lines in <H1> header, use the HTML :H1 command without the "Tag_value" and HTML :TX commands for the other lines followed by an HTML :H1- to close the <H1> Header.

HTML (Horizontal Rule)

Version Ref: 4.8 (not included in filePro Lite)

Syntax:

```
HTML:HR:SI "80"
```

<u>HTML</u> <u>Function</u>	<u>TAG</u>	<u>Tag</u> <u>Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u># of</u> <u>Values</u>
Horizontal Rule	<HR>	:HR	<none>	:AL	Align	1
				:SI	Size	1
				:WI	Width	1
				:NS		0

HTML(Image)

Version Ref: 5.0 (not included in filePro Lite)

Syntax:

HTML:IM:AL "CENTER"

<u>HTML Function</u>	<u>TAG</u>	<u>Tag Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u># of Values</u>
Image	<IMAGE>	:IM	<none>	:AL	Alignment	1
				:SR	Source	1
				:AT	Alternate	0
				:HT	Height	1
				:BO	Border	1
				:WI	Width	1
				:HS	Horizontal Space	1
				:VS	Vertical Space	1
				:US	USEMAP	1
				:IS	ISMAP	1

Description:

The :IM tag is used to insert images within text. These are often called "inline" images.

Note: The :IM tag is not a block tag by itself, so it must be used only within a block element. The location of the image file should be specified in the :SR attribute. It can be a relative or an absolute URL. When the image cannot be displayed, the browser should display the :AT text instead. The :WI and :HI attributes should contain the image's dimensions. This allows a browser to lay out the page in advance, as it now knows where the text below the image should be drawn.

:AL controls the alignment of the image with respect to the text.

:VS and :HS is a numeric value indicating the number of pixels that should be left free around the image.

:BO is used when the image is a link. It indicates that the browser should draw a border of the indicated size around the image to show that it is a link. It's most often used as :BO=0 to turn it off.

:IS and :US are used for imagemaps. The :IS attribute specifies that the link that this image is in goes to an imagemap program on the server, so the browser can send the coordinates of the selected location to the server. USEMAP is used for the client-side imagemap. It specifies the URL of the imagemap information. Support for this is limited, especially if the URL points to a different document rather than an inline anchor. See the section on the MAP tag for more information about client-side imagemaps.

HTML (Input)

Version Ref: 5.0 (not included in filePro Lite)

Syntax:

HTML:IN

<u>HTML</u> <u>Function</u>	<u>TAG</u>	<u>Tag</u> <u>Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u>#of</u> <u>Values</u>
INPUT	<INPUT>	:IN	<none>	:TY	Type	1
				:NA	Name	1
				:VA	Value	1
				:CH		0
				:SI	Size	1
				:ML	MaxLen	1
				:SR	Source	1
				:AL	Align	1
				:ZZ	Events	+

HTML(ISINDEX)

Version Ref: 5.0 (not included in filePro Lite)

Syntax:

HTML:IS :PR "Test Prompt"

<i><u>HTML Function</u></i>	<i><u>TAG</u></i>	<i><u>Tag Code</u></i>	<i><u>Tag Value</u></i>	<i><u>Attr</u></i>	<i><u>Value</u></i>	<i><u># of Values</u></i>
ISINDEX	<ISINDEX>	:IS	<none>	:PR	Prompt	1

Description:

<ISINDEX> was used before <FORM> became more popular. When inserted in a document, it will allow the user to enter keywords that are then sent to the server. The server then executes a search and returns the results. The :PR attribute can be used to override the default text in the dialog box.

HTML(Link)

Version Ref: 5.0 (not included in filePro Lite)

Syntax:

```
HTML:LN:HR "test.html"
```

<u>HTML Function</u>	<u>TAG</u>	<u>Tag Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u># of Values</u>
Link	<LINK>	Link	<none>	:HR	Href	1
				:RL	Rel	1
				:RV	Rev	1
				:TI	Title	1

Description:

:LN is used to indicate relationships between documents. There are two possible relationships.

The :RL attribute indicates a normal relationship to the document specified in the URL.

The :RV attribute is a reverse relationship. In other words, the other document has the indicated relationship with this one. The :TI attribute can be used to suggest a title for the referenced URL.

Usage:

Attrib	Value	Description
:RL	"copyright"	Indicates the location of a page with copyright information and such on the site.
:RL	"glossary"	Indicates the location of a glossary of terms for this site.
:RL	"help"	Indicates the location of a help file for this site. This can be useful if the data is complex or if the current document may require explanations to be used correctly (for example a large fill-in form).
:RL	"home"	Indicates the location of the homepage, or starting page in this site
:RL	"index"	Indicates the location of the index for this site. This doesn't have to be the same as the table of contents. The index could be alphabetical.
:RL	"stylesheet"	This indicates the location of the appropriate style sheet for the current document. The following link tags allow advanced browsers to automatically generate a navigational button bar for the site. For each possible value, the URL can be either absolute or relative.
:RL	"toc"	Indicates the location of the table of contents, or overview of this site.
:RL	"next"	Indicates the location of the next document in a series, relative to the current document.
:RL	"previous"	Indicates the location of the previous document in a series, relative to the current document.
:RL	"up"	Indicates the location of the document which is logically directly above the current document.
:RV	"made"	Indicates the creator of the document. Usually the URL is a "mailto: URL with the creator's address. Advanced browsers will now let the reader comment on the page with just one button or keystroke.

HTML(Map)

Version Ref: 5.0 (not included in filePro Lite)

Syntax:

```
HTML:MA :NA "imagemap"
```

<i><u>HTML Function</u></i>	<i><u>TAG</u></i>	<i><u>Tag Code</u></i>	<i><u>Tag Value</u></i>	<i><u>Attr</u></i>	<i><u>Value</u></i>	<i><u># of Values</u></i>
Map	<MAP>	:MA	<none>	:NA	Name	1

Description:

<MAP> identifies imagemap for showing information on the "hot spots"(clickable areas) in an image mentioned in an :AR tag.

Every selectable area should be mentioned in the :AR tag inside the :MA tag. The :NA attribute on the MAP tag assigns a name to the imagemap. The URL for the imagemap should point to this.

Example:

You have a map named "foo", you could reference to it with <IMG <../special/img.html SRC="map.gif" USEMAP="#foo"> if the image was on the same page.

HTML(Meta)

Version Ref: 5.0 (not included in filePro Lite)

Syntax:

HTML:MECO "Aprils Fools Day"

The :ME tag is used to convey meta-information about the document, but can also be used to specify headers for the document.

<u>HTML Function</u>	<u>TAG</u>	<u>Tag Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u>#of Values</u>
Meta	<META>	:ME	<none>	:HT	HTTP-EQUIV	1
				:NA	Name	1
				:CO	Content	1

Description:

You can use either :HT or :NA to name the meta-information, but :CO "Content" must be used in both cases. By using :HT, a server should use the name indicated as a header, with the specified content as its value.

Example:

HTML	:ME	:HT="Expires"	:CO="Sat, 01 Apr 2000 00:00 GMT"
HTML	:ME	:HT="Keywords"	:CO="April, Fools"
HTML	:ME	:HT="Reply-to"	:CO="Test@fpotech.com (Ernie)"

The server will include the following response headers when the document is requested:

Expires: Sat, 01 Apr 2000 00:00:01 GMT

Keywords: April, Fools

Reply-to: test@fpotech.com (Ernie)

Usage:

Common uses for META include:

META Name	Content	Description
"generator"	:CO="Some program"	This indicates the program used to generate this document. It is often the name of the HTML editor used.
"author"	:CO="Name"	This indicates the name of the author.
"keywords"	:CO="keyword"	Provides keywords for search engines such as Infoseek or Alta Vista. These are added to the keywords found in the document itself. If you insert a keyword more than seven times here, the tag will be ignored

HTML (Option)

Version Ref: 4.8 (not included in filePro Lite)

Syntax:

HTML:OP

<u>HTML Function</u>	<u>TAG</u>	<u>Tag Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u># of Values</u>
Option	<OPTION>	:OP	Optional Text	:VA :SE	Value Selected Flag	1 0

HTML(Ordered List, Unordered List, Definition List)

Version Ref: 5.0 (not included in filePro Lite)

Syntax:

HTML:OL

HTML:UL

HTML:DL

HTML:DT

HTML:DD

<u>HTML Function</u>	<u>TAG</u>	<u>Tag Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u># of Values</u>
Ordered List		:OL	<none>	:TY	Type	1
				:ST	Start	1
				:CO	Compact	1
Unordered List		:UL	<none>	:TY	Type	1
				:CO	Compact	1
Definition List	<DL>	:DL	<none>	:CO	Compact	1
Term	<DT>	:DT	Optional	<>		
Definition	<DD>	:DD	Optional	<>		

Description:

The :OL tag marks up an ordered list of items. Each item should be marked up with a :LI and it will be displayed with a number in front of it. The appearance of the number can be controlled with the :TY attribute:

I	Arabic numbers (default) (1, 2, 3, 4, ...)
A	Alphanumeric, lowercase (a, b, c, d, ...)
A	Alphanumeric, uppercase (A, B, C, D, ...)
I	Roman numbers, lowercase (i, ii, iii, iv, ...)
I	Roman numbers, uppercase (I, II, III, IV, ...)

The :TY attribute is the type of bullet. You have three possible styles: "disc" for a closed bullet, "square" for an open square and "circle" for an open bullet.

The :ST attribute indicates where the list should start.

The :CO attribute indicates that the list contains only short list items, so it may be rendered in a more compact way.

Notes:

If you want a list with bullets rather than numbers, use :UL. The :UL tag indicates an unordered list. Every item in a list is marked with and usually appears with a bullet of some sort in front of it. If you need numbering, use :OL for an ordered list.

HTML(PRE)

Version Ref: 5.0 (not included in filePro Lite)

Syntax:

```
HTML :PR "Optional text" :W "400"
```

<u>HTML</u> <u>Function</u>	<u>TAG</u>	<u>Tag</u> <u>Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u># of</u> <u>Values</u>
	<PRE>	:PR	Optional Text	:WI	Width	1

Description:

:PR is used to include text in which formatting is critical.

Notes: If value given, its text is used, and </PRE> automatically closed.

Unlike in the other HTML options, text with the PRE tag will only be wrapped at the line breaks in the source, and spaces will not be collapsed. You can even use tabs, although it is better to use multiple spaces since those will always be the right number.

Text inside this tag will be displayed in a mono-spaced font to retain the formatting. This is the reason you cannot include font-changing tags inside PRE text. Images are excluded because they can introduce problems with alignment. An image can't be translated to a certain number of characters. The optional :W attribute can be used to indicate how wide the text is (for example, :W=80 for a typical text file). This would allow the browser to pick a font that fits all the text in the current window.

HTML(Script)

Version Ref: 4.8 (not included in filePro Lite)

Syntax:

HTML:SC:SR "applhtmltest.scp"

<u>HTML Function</u>	<u>TAG</u>	<u>Tag Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u># of Values</u>
Script	<SCRIPT>	:SC	Text	:SR	Source	1
Language		:LA				

Note : If Flag :LA not used, the default language is JavaScript e.g. <script language=Javascript>. HTML 4.0 and later permit various Script Language values including JAVASCRIPT, JSCRIPT, VBSCRIPT, VBS.

HTML (Selection)

Version Ref: 4.8 (not included in filePro Lite)

Syntax:

HTML:SE

<u>HTML Function</u>	<u>TAG</u>	<u>Tag Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u># of Values</u>
Selection	<SELECTION>	:SE	<none>	:NA	Name	1
				:SI	Size	1
				:MU	Multiple Flag	0
				:ZZ	Events	+

HTML(Span)

Version Ref: 5.0 (not included in filePro Lite)

Syntax:

HTML :SP

<u>HTML</u> <u>Function</u>	<u>TAG</u>	<u>Tag</u> <u>Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u>#of</u> <u>Values</u>
Span		:SP	Optional Value	<	>	

Note: If value given, its text is used, and automatically closed.

HTML (Table, Data)

Version Ref: 4.8 (not included in filePro Lite)

Syntax:

HTML:TD

HTML:TA-

<u>HTML Function</u>	<u>TAG</u>	<u>Tag Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u># of Values</u>
Table Data	<TD>	:TD	<none>	:AL	Align	1
				:VL	Vertical Align	1
				:NO		0
				:CO	Colspan	1
				:RO	Rowspan	1
				:WI	Width	1
				:BG	Backgrnd Color	1
End Table	</TABLE>	:TA-	<none>		<none>	

HTML (Table, Table Row)

Version Ref: 4.8 (not included in filePro Lite)

Syntax:

HTML:TA

HTML:TR

<u>HTML Function</u>	<u>TAG</u>	<u>Tag Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u># of Values</u>
Table	<TABLE>	:TA		:CO	Columns	1
				:BO	Border	?
				:CS	Cell Spacing	1
				:CP	Cell Padding	1
				:WI	Width	1
				:ZZ		+
Table Row	<TR>	:TR	<none>	:AL	Align	1
				:VL	VerticalAlign	1

HTML (Text)

HTML (Paragraph), HTML (Break)

Version Ref: 4.8 (not included in filePro Lite)

Syntax:

HTML:TX

HTML:PA :AL "LEFT"

HTML:BR

<u>HTML Function</u>	<u>TAG</u>	<u>Tag Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u># of Values</u>
General Text		:TX	Text or memo field	<		
Paragraph	<P>	:PA	Optional Text	:AL	Align	1
Break	 	:BR	<none>	<		

HTML (Text area)

Version Ref: 4.8 (not included in filePro Lite)

Syntax:

HTML:TE

<u>HTML</u> <u>Function</u>	<u>TAG</u>	<u>Tag</u> <u>Code</u>	<u>Tag Value</u>	<u>Attr</u>	<u>Value</u>	<u># of</u> <u>Values</u>
Text Area	<TEXTAREA>	:TE	<none>	:RO	Rows	1
				:CO	Columns	1
				:NA	Name	1
				:WA		1

HTML (:ZZ)

Version Ref: 4.8 (not included in filePro Lite)

Syntax:

HTML:ZZ

The :ZZ flag is available for every HTML command, to allow you to add your own attributes to the command such as event processing statements etc. It takes a comma-separated list.

Example:

```
HTML "1" :AN:HR"#name" :ZZ "target=_top", "onMboseOver()=foo()", foo=bar"
```

Generates

If you do not specify optional text right after the :AN code, you must close the anchor with an HTML :AN- command. If you specify option text, filePro will automatically insert the code after the optional text. A simple processing table used to manually create a "H There document" follows:

HTML Sample

If: ' Create hithere.htm document

Then: HTML "1" :CR "hithere.htm" :T1 "H There"

If: ' Start Body

Then: HTML "1" :BO

If: ' Header1 Text Line

Then: HTML "1" :H1 "H There America"

If: ' Close Document

Then: HTML "1" :CL

JSON

Version Ref: 6.1 (USP6.1.01)

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

Export - Create a JSON file

Use these commands to create a JSON file using Processing.

```
JSON [id] :CR fname
```

Creates a JSON file with `fname` as the filename. 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 with `name` as the key.

```
JSON [id] :OB-
```

Closes an object.

```
JSON [id] :AR [name]
```

Starts an array in a JSON file with `name` as the key.

```
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: JSON files are created by appending one item at a time. When an array or object is started, any items added afterward will be added as direct "children" of the array or object until that array or object is closed.

Example - Create a JSON file

Processing:

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

Output (/tmp/myfile.json):

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

Import - Read a JSON file

Use these commands to read data from a JSON file.

```
JSON [id] :RO fname
```

Opens a JSON file for reading. The id is optional and defaults to "-" 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 (see **Key Syntax** below).

Key Syntax

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 - Read a JSON file

Input (/tmp/myfile.json):

```
{
  "name": {
    "first": "Tom",
    "last": "Anderson"
  },
  "age": 37,
```

```
"children": ["Sara", "Alex", "Jack"],
"fav.movie": "Deer Hunter"
}
```

Processing:

```
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
```

NOTE If a JSON file is missing, malformed, or broken upon attempting to open it, the file will not be opened and [HTMLERRNO\(\)](#) will contain a non-zero number:

```
1 -----
  . If:
  Then: JSON :RO "/tmp/mybrokenfile.json"
2 -----
  . If: HTMLERRNO() ne "0"
  Then: errorbox "Something is wrong with your JSON file!"; exit
3 -----
  . If: ' If no errors, we're okay to get values from the JSON
  Then: xx = JSON :GV "name.first"; msgbox "Hi," < xx
4 -----
```

XML

Version Ref: 6.1 (USP6.1.02)

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

Export - Create an XML file

Use these commands to create an XML file using Processing.

XML [id] :CR fname

- Creates an XML file with fname as the filename. 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 with name as the key.

XML [id] :EL-

- Closes an element.

XML [id] :AT name value

- Adds an attribute to an XML element with name as the key and value as the value.

XML [id] :TX text

- Adds a text element to an XML document with text being the value.

NOTE XML files are created by appending one item at a time. When an element is started, any items added afterward will be added as direct "children" of the element until that element is closed.

Example - Create an XML file

Processing:

```
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-
```

Output (/tmp/myfile.xml):

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

Import - Read an XML file

Use these commands to read data from an XML file.

XML [id] :RO fname

- Opens an XML file for reading with fname as the filename. 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 (see **Key Syntax** below).

Key Syntax

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 - Read an XML file

Input (/tmp/myfile.xml):

```
<EmployeeData>
  <employee id="21">
    <firstName>Tom</firstName>
    <lastName>Anderson</lastName>
  </employee>
  <employee id="99">
    <firstName>Tiffany</firstName>
    <lastName>Anderson</lastName>
  </employee>
</EmployeeData>
```

Processing:

```
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
```

NOTE If an XML file is missing, malformed, or broken upon attempting to open it, the file will not be opened and [HTMLERRNO\(\)](#) will contain a non-zero number:

```
1 -----
  . If:
  Then: XML :RO "/tmp/mybrokenfile.xml"
2 -----
  . If: HTMLERRNO() ne "0"
  Then: errorbox "Something is wrong with your XML file!"; exit
3 -----
  . If: ' If no errors, we're okay to get values from the XML
  Then: xx = XML :GV "name.first"; msgbox "Hi," < xx
4 -----
```

Simple Math

PLUS SIGN

MINUS SIGN

MULTIPLICATION SIGN

DIVISION SIGN

Description:

+	Adds fields or literals
-	Subtracts fields or literals
*	Multiplies fields or literals
/	Divides fields or literals
^	Raises to a number to a power of an exponent.

Exponents

Use the "^" caret symbol to raise a base number to the power of the exponent

Syntax

$$z = x ^ y$$

Return value

X raised to the power of Y. (Y can be fractional.)

Example:

MSGBOX "2 to the 10th power is" < "2" ^ "10")

AA = "10" ^ "0.5" ---- Same as AA = SQR("10")

Notes:

0^0 returns 1. Fractional powers of negative numbers and negative powers of zero return 0.

Math, Financial

Description

Financial Math Functions allow you to calculate the "Time-value-of-money".

Version Ref: 5.0

Formulas Description

$N = TVM_N(i, pv, pmt, fv)$	Calculates the number of compounding periods.
$I = TVM_I(n, pv, pmt, fv)$	Calculates the value of interest.
$PV = TVM_PV(n, i, pmt, fv)$	Calculates the present value.
$PMT = TVM_PMT(n, i, pv, fv)$	Calculates the payment value.
$FV = TVM_FV(n, i, pv, pmt)$	Calculates the future value.

The formula used is:

$$\frac{100}{(1 - sppv)^i} * pmt + pv = -fv * sppv$$

where "sppv" is the single payment present value:

$$sppv = \frac{100}{1 + i - n}$$

Note: It is assumed that payments are made at the end of each period.

Log Functions

Version Ref: 4.8

LOG(n)	Natural logarithm (base e)
LOG10(n)	Common logarithm (base 10)
EXP(n)	Exponent function (e^n)
EXP10(n)	Base 10 exponent (10^n)

Examples:

When na = "100"; nb = "4.6051702", nc = "2.000000"

xx = LOG(na) returns natural log value "4.6051702"

xx = LOG10(na) returns common log value "2.000000"

xx = EXP(nb) returns antilog(base e) value "100.00000"

xx = EXP10(nc) returns antilog(base 10) value "100"

Note:

Natural Logarithms (also called Napierian logarithms) are logarithms to the base 'e' where $e = 2.71828$ (5 dec. places).

Trig Functions

Version Ref: 4.8

Function	Description
ASIN(xx)	Returns Arcsine for angle in radians.
ACOS(xx)	Returns Arccosine for angle in radians.
ATAN(xx)	Returns Arctangent for angle in radians.
ATAN(ry,rx)	Returns Arctangent for sides in radians.
DASIN(xx)	Returns Arcsine for angle in degrees.
DACOS(xx)	Returns Arccosine for angle in degrees.
DATAN(xx)	Returns Arctangent for angle in degrees.
DATAN(dy,dx)	Returns Arctangent for sides in degrees.
DSIN(xx)	Returns sine for angle in degrees.
DCOS(xx)	Returns cosine for angle in degrees.
DTAN(xx)	Returns tangent for angle in degrees.
SIN(xx)	Returns sine for angle in radians.
COS(xx)	Returns cosine for angle in radians.
TAN(xx)	Returns tangent for angle in radians.
DTOR(xx)	Converts angle in degrees to radians.
RTOD(xx)	Converts angle in radians to degrees.
PI()	Returns the value of PI or 3.14159265

Note:

ATAN/DATAN functions can be used with a single or two parameters. Using ATAN(o,a) and DATAN(o,a) with two parameters provides for entering two sides of the right triangle in the function instead of entering the tangent value. For example, with a 3,4,5 triangle, where the two legs are 3 and 4, then entering ATAN(3,4) will return .6435011 and DATAN(3,4) returns 36.86989765. The functions are calculating the angle formed by the hypotenuse and the side opposite ("o" leg). To calculate the angle formed by the hypotenuse and side adjacent ("a" leg), you can switch the parameters.

Hyperbolic Functions

Hyperbolic Functions SINH/COSH/TANH/ASINH/ACOSH/ATANH

Version Ref: 5.0

Examples:

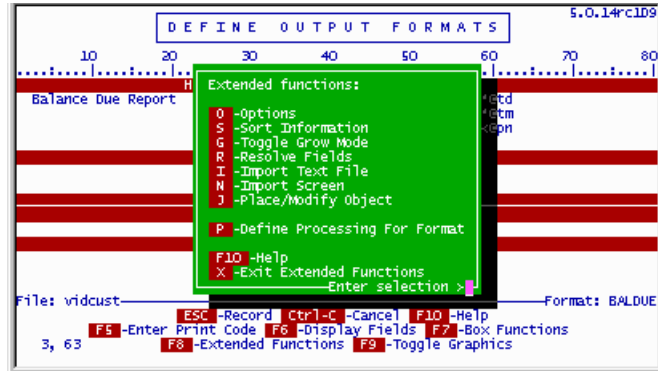
$$xx = \text{SINH}(yy)$$

$$zz = \text{ASINH}(xx)$$

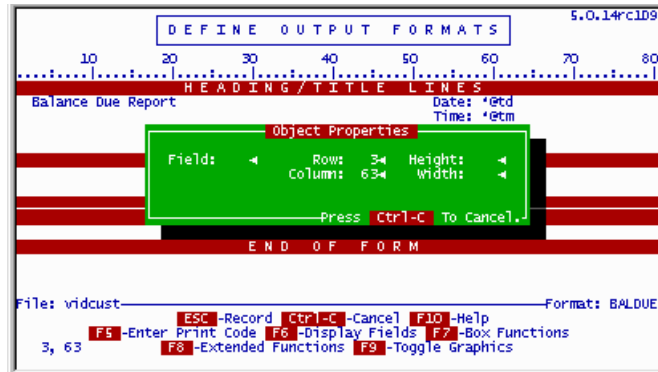
Objects

In order to accommodate memo printing, you can include fields in a report as objects. Although this feature was added in support of memo field printing, it is not limited to memo field types and can be used for any field.

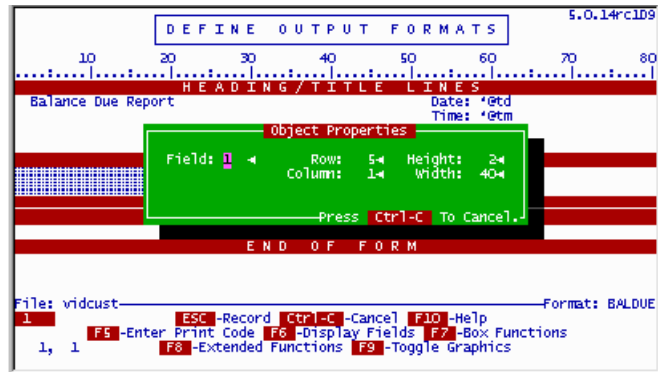
When using the " Define Output " option, an extended function option " J " allows you to place and modify an object.



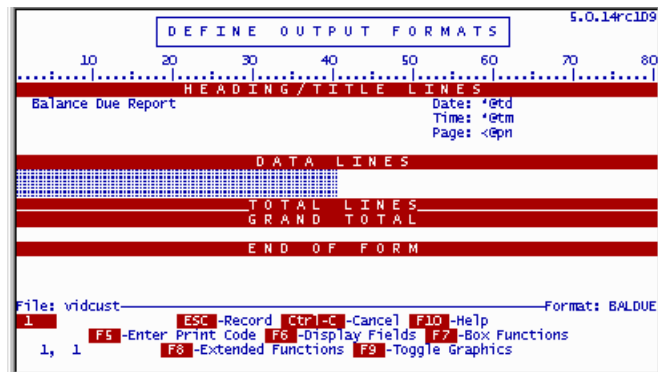
Press [J] - Place/ Modify Object



You can identify the Object Properties e.g. the field number and specify the area occupied by the Object on your output. Enter a field number and place the object by identifying the starting Row and Column, the number of rows in the Height parameter and the number of columns in the Width parameter.



In the above case, we have identified field number 1 as the object and specified an area starting in row 5, column 1 with a height of 2 rows and width of 40. When pressing [Save], you will see the placement of the object presented in your output format as follows.



As previously stated, objects can be used to control memo printing on your output but objects are not limited to memo fields and thus can be used for any real field or dummy variables defined in processing.

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. See [PDF Printing](#) for information on configuring an output destination to use a PDF format. This page will explain how to create basic form elements using this new feature.

There are four types of PDF Form Objects that can be used:

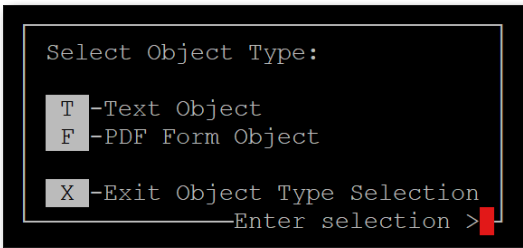
- Textbox
- Dropdown
- Checkbox
- Radio

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.

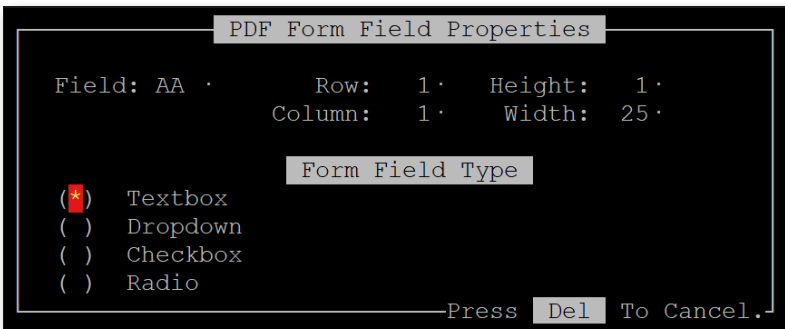
NOTE Using the new generation features in a report can lead to unintended results. Fields are shared across records and pages. Updating one field updates all matching instances of that field throughout the document. It is recommended to use output forms over output reports.

Placing a PDF Form Object on an Output Format

In Update Mode on an output format, press **F8** for Extended Functions, then **J** to place an object. From here, one can press **F** to place a PDF Form Object.

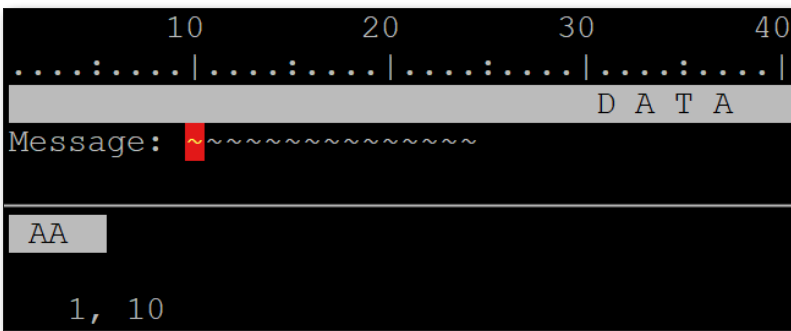


A window will appear with several options - which type of object to add, as well as this object's properties, such as width, height, and the row and column this field should be placed on.



Move the cursor to select one of the Field Type options and press **SPACE** to select one of the four field types. Configure the properties as desired. Press **RECORD** to place the field on the output format.

When the cursor is placed on an object, the object's Field property will be displayed in the bottom-left corner of the screen. The space occupied by an object is represented by ~ characters.



Form Object Properties

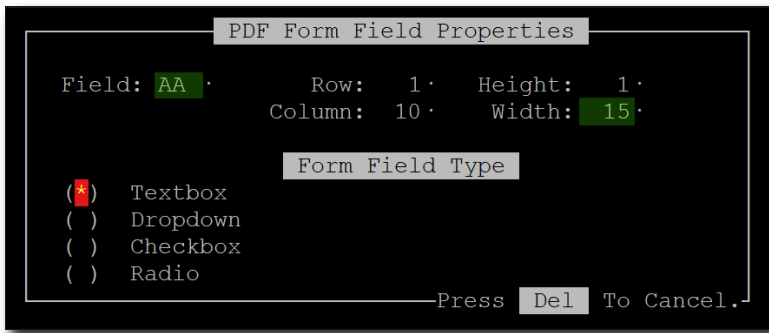
Refer to the table below for a breakdown of each object property. If an object cannot use a particular property, it will not be shown in the properties window.

Property	Description	Used By
Field	Placeholder data that goes in this object. Accepts real fields or dummy fields.	All Form Objects
Data	The comma separated list of options that can be selected when the dropdown arrow is used.	Dropdown
Row	The row of the output format to place this object on.	All Form Objects
Column	The column of the output format to place this object on.	All Form Objects
Height	The number of rows this object uses, starting from the Row and moving down.	Textbox
Width	The number of columns this object uses, starting from the Column and moving right.	Textbox, Dropdown

Textbox

A Textbox is a simple area where the user can input text, up to a certain length. Multiple rows can be included for large text areas spanning multiple lines. These are useful for things like names, addresses, and long descriptions.

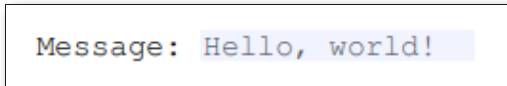
To set a placeholder using a field, create and set a dummy variable or use a real field in the format's Output Processing that has the same value as the object's Field property. For example, to set a Textbox object's value to "Hello, world!", create a dummy variable `aa(13,*)="Hello, world!"` and make sure the object's Field property is set to AA. Make sure the width is long enough to accommodate the data in the field, or the text will be chopped off!



Output Processing:

```
Then: aa(13,*)="Hello world!"
```

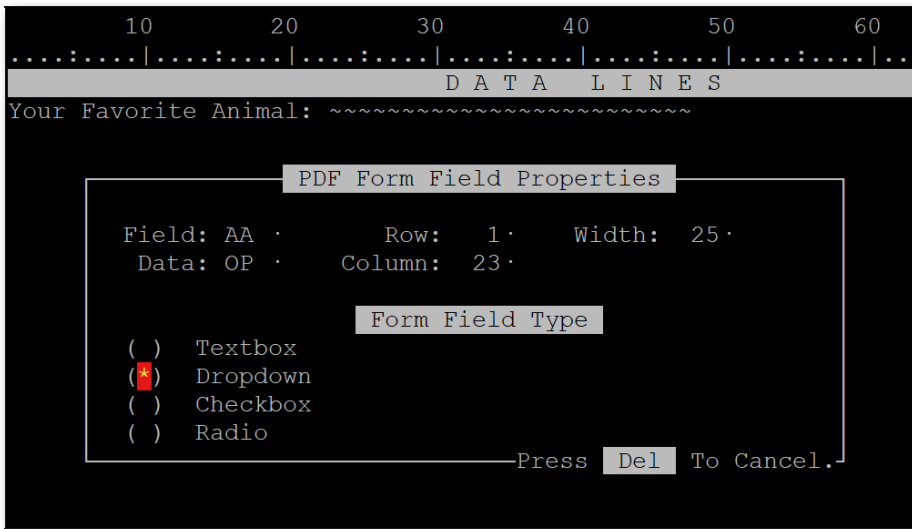
The resulting output using the example and images above looks like this:



Since this is a fill-in-the-blank form object, the "Hello, world!" can still be modified by the user - using a dummy variable simply sets the field's initial value.

Dropdown

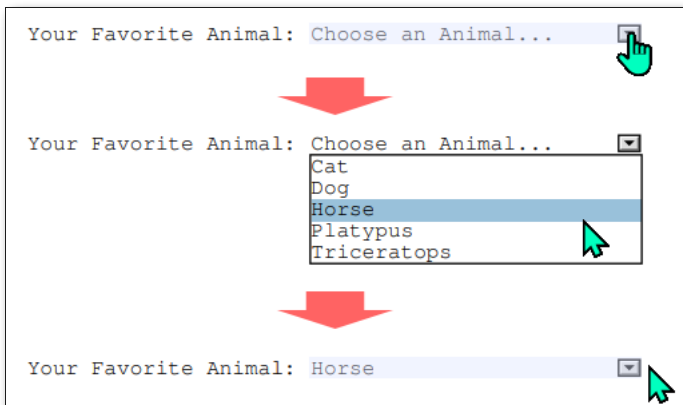
Dropdown objects have an additional Data property that refers to a field containing a list of comma separated entries that are predetermined and can be picked by the user. Similar to the Textbox, the optional Field property can be used to assign a default value to the field. The placeholder Field property does *not* have to match one of the options from the Data list field.



Output Processing:

```
1 -----
  · If: ' The default option ("Field" property)
    Then: aa(25,*)="Choose an Animal...";
2 -----
  · If: ' List of animals to pick from a dropdown ("Data" property)
    Then: op="Cat,Dog,Horse,Platypus,Triceratops";
```

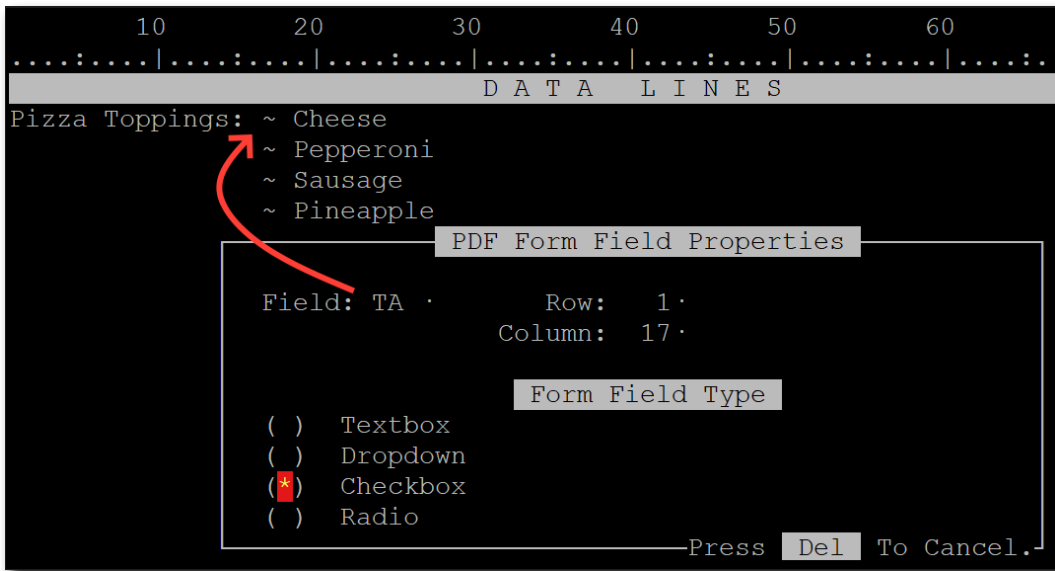
The resulting output using the example and images above looks like this:



Selecting an entry from the dropdown list will replace the default text.
NOTE: Once a selection is made, it is not possible to revert back to the default text.

Checkbox

The checkbox is a 1x1 box that can be toggled on or off. Each checkbox should have its own (1,yesno) field reference. Setting a field in Processing to "Y" will generate the field's checkbox as "checked".



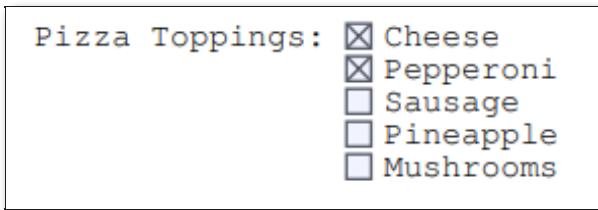
Each Field on the output is its own *yesno* checkbox object with fields TA, TB, TC, TD, and TE

Output Processing:

```

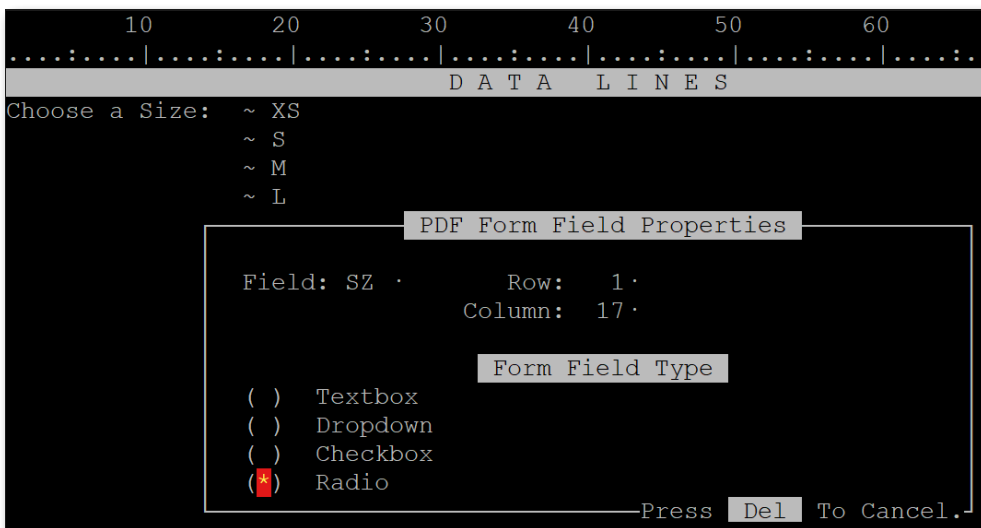
1 -----
  · If: ' Toppings A thru E
  Then: ta(1,yesno); tb(1,yesno); tc(1,yesno); td(1,yesno); te(1,yesno);
2 -----
  · If: ' Mark "Cheese" and "Pepperoni" as checked
  Then: ta="Y"; tb="Y";
  
```

The resulting output using the example and images above looks like this:



Radio

Radio buttons are similar to checkboxes, but are grouped together under the same field, meaning only one option in the radio button group can be selected. Instead of a (1,yesno) field, radio buttons are referenced by their *index number* and should use an appropriately sized numeric edit such as (1,.0).



Each Field on the output has the same Field property, SZ. This will group all the objects together in the same radio button "group".

```

1 -----
  · If: ' Mark the fourth shirt size, "L", as selected
  Then: sz(1,.0)="4";
  
```

The resulting output using the example and images above looks like this:

Choose a Size: XS
 S
 M
 L
 XL

Reading a Fill-In-The-Blank PDF Document

Processing can also be used to read in values from a PDF Document.

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

```
name = 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.

Differences between DOS/Windows and Unix

NOTE: This section refers to the old DOS file formats and does not apply to 32 bit Windows versions that support long filenames.

This Section Contains

Overview

The Unix and DOS filename limitations

Color

The USER Command

Case Sensitivity

The Slash and the Backslash in filenames

File Permissions (Attributes)

Filesystems

Overview:

There are several differences between these operating systems that affect filePro. The most important is the old filename limitation from early versions of DOS. When DOS was originally designed it allowed for 8 character filenames and a 3 character extension to that name (the extension follows a "." period). This naming convention has come to be known as 8.3, eight characters a period and three more characters. When you see a DOS directory, (by issuing the DIR command) as in:

DIR

You might see:

Volume in drive C has no label

Volume Serial Number is 3C5C-1EF1

Directory of C:\FILEPRO\STATES

.			<DIR>	03-18-97	3:48p	.
..			<DIR>	03-18-97	3:48p	..
DATA		0		12-04-89	4:55p	DATA
DEFAULT	OUT	1,344		12-30-88	5:01p	DEFAULT.OUT
INPUT	PRC	344		12-30-88	5:21p	INPUT.PRC
INDEX	A	923		01-11-89	2:15p	INDEX.A
KEY		3,618		11-08-88	6:30p	KEY
MAP		110		05-16-89	1:07p	MAP
MAP	TMP	110		05-16-89	1:07p	MAP.TMP
SCREEN	0	3,282		06-30-92	1:19p	SCREEN.0
	7 file(s)		9,387 bytes			
	2 dir(s)		57,999,360 bytes free			

The actual name of the files and directories appears on the right side of this display. The left shows a space when there is really a period in the name. If you wanted to erase a file, you could not type:

erase INDEX A

You would have to type:

erase INDEX.A

The Unix and DOS filename limitations

The 8.3 and 14 character limitations of the early versions of both these operating systems constrains your creativity just a bit... or challenges it. Because of this 8.3 file naming convention, filePro changed some of its component names from their original Unix(Xenix) names. Under Unix the name would be "out.default", under DOS it is "default.out". This is true of some other filePro filenames, processing tables are "prc.name" and "name.prc", selection sets are sel.name and name.sel, etc. This does not ever present a problem for you as a filePro user since filePro always handles all these names for you automatically. However, there is one caveat that is important. If you are writing filePro programs that you want to work both under Unix and DOS, you must make sure not to exceed the 8.3 filename constrictions when designing your application. This impacts most heavily on screens. Under Unix, screen names can be up to 7 characters long, under DOS the limitation is 3 characters.

Example:

Unix screen.bigname

DOS screen.big

This was the one filePro component which did not take advantage of reversing the order from Unix to DOS.

In any case, if you are going to want to run your apps under both DOS and Unix, remember to pick only eight character names for most filePro components and the name of the actual filePro directory, and only 3 character names for screens. In this way, you can transfer the application from one side to the other without too much problem at all.

Color

DOS supports full color screen design. The Unix system only supports full color on ANSI based terminals/consoles, however, Unix filePro will only display color screens correctly. That is, Unix filePro does not allow you to design a color screen, but it will display a color screen designed under DOS and transferred to Unix.

Unix does have six native color allocations. You can set these six colors (under SCO Unix) with the "setcolor" command. The six associations are for foreground and background text, foreground and background reverse video, and foreground and background graphic lines. These six global settings will be displayed by Unix filePro.

Login IDs (users)

The Unix O/S assigns a user name (and ID#) to every login session. Unix filePro can take advantage of this unique ID. There is no analog for this under DOS or Network-based

filePro..

The USER Command

Under Unix, filePro can hand values a "user-based" program that is executed by the operating system and gather back returned values from that program. This is not possible under DOS. However, Windows 7 and higher added the ability for the USER command to work so it was added to the Windows compiles.

Case Sensitivity

The older DOS operating systems were case-insensitive. This means that you could type in either lower or UPPER case and the system would convert your input to UPPER case automatically. If you executed the command:

```
ERASE truck
```

DOS would remove the filename TRUCK.

This is important because filePro uses the system to find files and is also case insensitive in this regard. If you ask filePro to look into a file called "truck" or one called "TRUCK" it will get the file called "TRUCK".

Under Unix, case is VERY important. If you ask to do something to a file called "truck" and there is only a file called TRUCK, Unix will say something like "truck not found". The following filenames are all completely different and unique under Unix:

```
filename  
FILENAME  
fileNAME  
FiLeNaMe  
(etc.)
```

The Slash and the Backslash in filenames

DOS uses the backslash "\", Unix uses the slash "/". An easy way to remember which is which: The slash is the one used in fractions, i.e., $\frac{1}{2}$, $\frac{3}{4}$, etc. The backslash is the other one. It is important to use the right one for each operating system, but more and more programs (including filePro) are allowing you to use either and making the necessary substitution for you automatically. However, at the command line, or in scripts (and batch files) the shells of either O/S will NOT make this substitution... you must get it right. Slash / for Unix, backslash \ for DOS.

File Permissions (Attributes)

Each operating system protects its files from being removed, overwritten and abused by unauthorized people. Under Unix, the system uses an elaborate permission scheme involving the owner of a file, his or her group and the rest of the world. The permissions can be set by the owner of the file to allow or disallow read, write and execute permission for this file to any of these users (owner, owner's group, world). The permissions also denote whether a file can be "executed", that means run as a program. You would not need to make a word processing document executable, however, you would want to make a script that performs some work for you executable. The "chmod" (change modification) command is used to set a file's permissions. See the index for more on Unix permissions and ownerships.

Unix also designates the "owner" of every file or directory in a filesystem. FilePro makes use of this security feature by naming all of its directories and files owned by the user "filepro". This user is installed on the system before filePro will run on a Unix system. The "chown" (change owner) command is used to change a file's ownership. See the index for more information.

DOS has a series of "attributes" for filenames and directories. These attributes denote a file's accessibility, whether it has been backed up since the last backup, etc. whether it is hidden or visible, readable, etc. These settings are not important to filePro as long as the file(s) needed are not "read-only". There might be some sharing problems across various networks, but that is something the network should handle correctly.

Filesystems

Filesystems under DOS are sometimes called "drives". C: is a filesystem, D: is another filesystem. The association is out of date now. It comes from a time when each drive was turned into one filesystem. Now a hard drive can be partitioned into many filesystems, they are still given letter names in DOS D:, E:, F: etc.

Under Unix, filesystems are not called "drives". They are still, however, a partition of any particular drive. Under Unix, they may have a name like "/u/hd2" or /u, or /u/appl.

A typical filesystem is organized into directories. Under DOS/Windows these directories are also called "folders". A directory may contain other directories and/or files. A file can be anything from a screen layout, an output format or a plain text file that describes the organization of your database. Files can be plain text files which can be read by humans or they can be binary files which can only be deciphered by the computer. A file can be a program, a document, device description, etc. The computer opens files to do all of its work, including loading its own brains into memory at startup... so it can do all the other things it does. The brains of the computer are also stored in a file somewhere on the hard disk. A built in "startup" routine within the computer's Basic Input Output System (its BIOS) tells it how to find this file and how to load it into memory so that it can be a computer and not a useless hunk of junk. Once the computer has loaded the files it needs into memory (for its own well being) then it can assist you in loading your files and programs.

FilePro is a collection of programs in a directory called "fp". This directory can sit anywhere (under any other directory or long chain of directories) as long as the program itself knows where this directory is located. The filePro applications that you write and use are also a collection of files and directories. Each filePro "file" is really a directory under the "filepro" directory. The name of the filePro file is the name of the directory. Under this directory are the various screens, output formats, processing tables and data which this file uses. The two directories "fp" and "filepro" are very important to filePro and it must know how to locate these files. Special environment variables are used by filePro to find these directories. If these environment variables that point to "fp" and "filepro" are empty, filePro uses default values.

DOS and Network Versions

filePro is compiled through version 4.8 for single user DOS and Networks. This version continues to use the DOS4GW extender software to make use of extended memory.

With filePro version 5.0, the compile was done on a Windows 32 bit systems and no longer required the DOS4GW extender. For newer Operating Systems it is important to be on supported versions of filePro 5.6 and higher

Environment

Unix, DOS, Windows, O/S2... all operating systems start you off at a "shell." A shell is a command interpreter. It takes your keyboard entries and hands them to the computer in a way it can understand and act upon. When you are positioned at a shell prompt (DOS=C:\, Unix=\$ or # or %, Windows=GUI Desktop interface) you are sitting in a special environment. There are two very important aspects of this environment:

The environment uses variables to control your operation.

You can change these variables to customize your environment to taste.

Environment Variables

This section contains

Setting Environment Variables (DOS)

Setting Environment Variables (UNIX/LINUX/XENIX)

Exporting Variables

Unsetting Environment Variables

Listing Environment Variables

- DOS/WINDOWS

Setting Environment Variables

This is an easy matter under DOS. You use the "set" command. The syntax is:

```
SET VARIABLENAME=VALUE
```

Example:

```
SET MINE=C:\WORDPERFECT\WPDOCS\MYFILES
```

Once a variable is set in DOS, you use it by obtaining its contents with a % sign directly in front of it and directly after it. For example, if you set a variable in the above manner, you could use it like this:

```
cd %MINE%
```

This tells the "cd" command (change directory) to change you to the long directory name contained in the variable MINE. A good shortcut. It is the very same thing as typing:

```
cd C:\WORDPERFECT\WPDOCS\MYFILES
```

Unsetting Environment Variables

To unset a variable in DOS, use the syntax:

```
set variablename=
```

By setting it equal to nothing, it goes away.

Listing Environment Variables

To list the environment variables in your environment (and their values) use the "set" command by itself. When you type "set" you might see a listing like this:

```
BLASTER=A220 15 D1 H5 P330 T6 E620
JAVA_HOME=C:\cafe\JAVA
MINE=C:\WORDPERFECT\WPDOCS\MYFILES
PATH=C:\cafe\JAVA;C:\WINDOWS;C:\WINDOWS\COMMAND;C:\FP
PFDSK=C
TEMP=C:\TEMP
```

To see the contents of just one environment variable, use the "echo" command, as in:

```
echo %MINE%
```

You will see:

```
C:\WORDPERFECT\WPDOCS\MYFILES
```

- UNIX / LINUX / XENIX

Setting Environment Variables

Under Unix, setting variables is even easier than under DOS. You do not even need the "set" command. The syntax is just:

```
variablename=value
```

Example:

```
mine=/usr/tracy/wp/myfiles
```

Exporting Variables

Once a variable is set in Unix, it is only useful to your current shell. That is to say, if you call up another program like filePro or WordPerfect, these variables are completely lost... they are simply not seen in that called environment. This is why you must always "export" any variable that you set under Unix. The syntax is very simple:

```
export variablename
```

So, you can consider that setting any environment variable under Unix is a two-step process. First set it, then export it. You only have to export a variable one time. After that you can change its value as many times as you like, it will remain exported and always send out the right value to programs that you call. The whole syntax is as follows:

```
variablename=value
export variablename
```

Example:

```
mine=/usr/tracy/wp/myfiles
export mine
```

To obtain a variable's contents just place a \$ directly in front of it. For example, if you set a variable in the above manner, you could use it like this:

```
cd $mine
```

This tells the "cd" command (change directory is the same command in both Unix and DOS) to change you to the long directory name contained in the variable mine. Again, it is the very same thing as typing

```
cd /usr/tracy/wp/myfiles
```

Unsetting Environment Variables

To unset an environment variable in Unix, use the "unset" command, as in:

```
unset variablename
```

This will make it go away.

You can also set the variable equal to nothing as in:

```
; e#=""
```

But this will not clear it from the environment, just set it equal to nothing.

Listing Environment Variables

To list the environment variables in your environment (and their values) use the "set" command by itself. When you type "set" you might see a listing like this:

```
PATH=/bin:usr/bin:/usr/SHOME/bin:/u/appl/fp
PFTMP=/usr/tmp
PS1=$
TZ=EST5EDT
```

If you want to see which of these variables is exported, use the export command by itself, as in:

```
; %aYou might see something like:
```

```
export ABE=ASCII
export PATH
export PFTMP
export PS1
export TZ
```

LINUX

filePro has been compiled to run on this more recent Operating system. Linux is a UNIX like system which can be obtained from REDHAT, Caldera, and other vendors.

Beginning with filePro version 5.6, support for FreeBSD was added.

Installation instructions and library requirements could be different depending on the NIX distribution used. filePro 32 bit requires the 32 bit compatibility libraries. Refer to the installation instructions provided with you media for the details of library installation.

Microsoft Windows

This version uses the native 32-bit code of this popular O/S from Microsoft Corp. and does not require any DOS extenders such as DOS4GW.

PATH

Of all the operating system and environment concepts that it is important for you to understand, PATH is probably the most important. It functions virtually the same under Windows and Linux/Unix. PATH is very simple. It is an environment variable that lists the directories in which the system looks to find the command you have just entered. If it does not find the command (or program) in any of these PATH directories, it returns an error and tells you it cannot find the command or program you requested. Most computer systems have hundreds or thousands of directories... it would be impossible (or at least VERY slow) to check in every single directory for the command you are calling. Instead, a few directories are specified (by convention) to hold your executable programs and scripts (batch files) and the O/S (actually your shell) looks in only these directories. This is why the computer responds so quickly to your orders.

A typical PATH in Linux/Unix might be:

```
/bin:/usr/bin:/usr/$HOME/bin
```

(HOME is just another environment variable that is substituted automatically when the PATH value is obtained. This variable may be set to "tracy", "bill", "root", etc. (See Unix=.profile, Windows = fpplus.bat, fulldev.bat, user.bat, etc.))

A typical PATH in Windows might be:

```
C:\WINDOWS;C:\WINDOWS\COMMAND
```

It is important that you know where your executable scripts (or batch files under Windows) live. If they are not inside a PATH directory, they will not be found unless you were to issue their "full" pathname either at the command line or within your executable scripts. However, if the executable program you want is within a PATH directory you can just call it by name, the system will automatically find it and execute it, as in:

```
sort filename
```

Under Unix the "sort" command would be found in /bin. Under DOS, it would be found under C:\WINDOWS\COMMAND, assuming these directories were in the PATH variable list, the system will easily find the command for you and execute it. Otherwise, to operate the program, you would have to issue the command as:

```
Linux or Unix /bin/sort
```

```
Windows C:\WINDOWS\COMMAND\SORT
```

The PATH variable is great shorthand for the user and the computer.

UNIX

filePro has been compiled to run on about all flavors of UNIX.

UNIXWARE7

This powerful O/S from SCO has now been tested and approved. The standard UNIX version of filePro runs fine on this operating system.

XENIX

There was a separate version compiled for XENIX up through version 4.5 but, since this O/S is not being maintained beyond year 2000, later versions have not been compiled. If you are currently running XENIX or any version of filePro prior to 5.0, you should contact sales to purchase an upgrade at a reduced price.

Alias Field Assignment

Syntax:

Then: aliasname:field

Description:

Alias field assignment substitutes an "alias" name for a field. The field may then be referred to by this alias as well as its number or letter(s).

Examples:

Simple Aliases

Then: BALANCEDUE:37

Then: YTDTOT:aa

Create and WordPerfect Merge File

File Name: Anyfile

Processing Table: wpmerge

Then: ' out.wpmerge - makes a file called /tmp/wpmerge.wp

Then: ' use this file to merge with wordperfect as the secondary file

Then: ' map significant fields to aliases (for clarity and ease of use)

Then: ' the array processing will 'close up' blank lines in the address

Then: FNAME:4 ; LNAME:3 ; COMPANY:2 ; ADDRESS1:8 ; ADDRESS2:9

Then: CITY:11 ; STATE:12 ; ZIP:13

Then: export wordperfect merge=/tmp/wpmerge.wp

Then: dim array[5](50) ; clear array

Then: i(1,0)="1"

If: FNAME ne "" or LNAME ne ""

Then: array[i] = "" { FNAME<LNAME ; i=i+1"

If: COMPANY ne ""

Then: array[i] = "" { COMPANY ; i=i+1"

If: ADDRESS1 ne ""

Then: array[i] = "" { ADDRESS1 ; i=i+1"

If: ADDRESS2 ne ""

Then: array[i] = "" { ADDRESS2 ; i=i+1"

If: CITY ne "" or STATE ne "" or ZIP ne ""

Then: array[i] = "" { CITY<STATE<ZIP ; i=i+1"

Then: merge(1)=array["1"]

Then: merge(2)=array["2"]

Then: merge(3)=array["3"]

Then: merge(4)=array["4"]

Then: merge(5)=array["5"]

Then: end

Alias Filename Assignment

Syntax:

Then: COMMAND alias = filename <options>

Description:

Commands that access files, such as IMPORT, EXPORT and LOOKUP, let you give these files "alias" (assigned) names. From the point that the assignment is made, the file (and its fields) MUST be referred to by the alias instead of the real filename.

Examples:

Then: lookup cust=customers r=free -e

The alias name "cust" must now be used to reference this file and the fields in it. The processing table will not recognize the real filename after an alias has been assigned.

Then: lookup cust=oldcusts k=1 i=a -nx

If not cust

Then: show "@ {Customer"<1<"is not in the archive file." ; end

The alias name (just like a real filename) may be used as a positive or negative test for whether any lookup finds a match.

Then: export ASCII epp="C:\tmp\epayplan" -X

The alias "epp" would now be used to reference this file.

IMPORTANT: If the line on which an assignment is being made has not been executed at least once, you can not refer to it elsewhere on the table. After the assignment line has been executed just once, you can refer to the assignment above or below that point on the table.

Assignment

= EQUAL SIGN

Syntax:

Then: f = v

"f" equals a real field, dummy field, or filename alias

"v" equals value to be assigned

"v" is an expression (only requires surrounding quotes around literal strings)

Description:

The equal sign operator (when used on a "then" line) assigns the value of the expression on its right to the field on its left. Assignments can be made to real field and dummy fields. Assignments can NOT be made to system maintained fields, only the system can do this.

NOTE: It is important to understand that the entire expression on the right of an equals sign will be calculated (fully processed to its actual value) before it is assigned to the field on the left.

In the case of numeric calculations, all operations are performed first (according to standard rules of precedence), then the assignment of the final value is made.

Examples:

Then: aa = "12"/"4"

In the above, field aa will be set equal to "3" not "12"/"4".

In the case of string operations, all manipulations and substitutions are made first and then the final value is assigned.

Then: aa=sales(12) { "," < sales(13) < sales(14)

In the above, the value of the three fields from the sales file will be joined in the designated manner and then assigned to the field aa.

VERY IMPORTANT: While it appears that filePro processing tables are working with algebraic equations, they are NOT. In algebra, the following would not be valid or correct:

Then: aa = aa + "1"

How could something be equal to itself plus 1? It can't. At least not in algebra and probably not too often in the real world either, but in filePro it makes perfect sense, and is perfectly legal. The above statement means take the current value of aa, add 1 to it and assign that newly calculated value to aa. This is changing the value of aa to something else, it is an assignment, not an equality.

NOTE: It is a quirk of filePro that the = operator is allowed on "if" lines. When used on "if" lines it is working as a comparison tool. In other words:

If: 4 = "Smith"

is the same as

If: 4 eq "Smith"

The above says "Does the value in field 4 match the value "Smith". If the fields/expressions being compared are numeric, the compare is a numeric compare. If the fields/expressions involved are strings an ASCII compare is performed.

HINT: It is STRONGLY suggested that you NEVER use the = (equals operator) on "if" lines. Always use "eq" instead. It will work for both numeric and string compares just as the equals operator, but you will never confuse it with = on a then line. You will never mistake it for an assignment. Your code will be easier to read. Incidentally, "eq" can never be used on a "then" line to replace the = operator. The strange behavior only works in one direction, this is another reason for not making use of the = operator quirk and use it only on "then" lines where it really belongs.

BITWISE is for just working on the bits that make up a number. It allows you to store multiple flags in the same number and mask them. It's useful in encryption, search algorithms, etc. You most likely would only really use them in something really complex. The average user probably won't get much use out of them but filePro has the function for those who may need it. A bitwise operation operates on one or more bit patterns or binary numerals at the level of their individual bits. It is a fast, simple action directly supported by the processor, and is used to manipulate values for comparisons and calculations.

Version 6.0.00

Sample Code

```
top::
::end:
@once::declare candelete(3,.0,g); declare canpop(3,.0,g); declare cansave(3,.0,g):
::candelete="1":
::cansave="2":
::candelete="1":
::declare flags(3,.0,g); flags="0":
:::
:' set candelete flag:flags=flags ~| candelete;:
:' set canpop flag:flags=flags ~| canpop:
:' set cansave flag:' flags=flags ~| cansave    ' not set:
::end:
@keyp:(flags ~& canpop) eq "0":msgbox "You are not authorized."; end:
::msgbox "Hello!": ::end:
@keys:(flags ~& cansave) eq "0":msgbox "You are not authorized to save."; end:
::msgbox "You can save": ::end:
@keyd:(flags ~& candelete) eq "0":msgbox "You are not authorized to delete."; end:
::msgbox "You can delete": ::end:
```

Examples

```
bitwise and (3 ~& 1)
(bin)11 (dec)3
(bin)01 (dec)1
result (bin)01 (dec)1
```

```
bitwise or (3 ~| 1)
(bin)11 (dec)3
(bin)01 (dec)1
result (bin)11 (dec)3
```

```
bitwise xor (3 ~^ 1)
(bin)11 (dec)3
(bin)01 (dec)1
result (bin)10 (dec)2
```

```
right shift (3 ~> 1)
(bin)11 (dec)3
result (bin)01 (dec)1
```

```
left shift (3 ~< 1)
(bin)11 (dec)3
result (bin)10 (dec)2
```

Expressions

Description:

Throughout the Processing Reference section, you will see "exp" or "expression". This means any legal filePro code that will eventually resolve to a number, date, time, or string-of-characters. Loosely translated, an expression is anything that can go on the right side of an equals sign on a "then" line.

Examples:

4

Z

Aa

3/"2"

3/aa

mid(@td,"1","2") & "/" & mid(4,"4","2") & "/" & mid(9,"7","2")

filename(n) This type of expression has some limitations when used in manyplaces where expressions would normally work.

array["2"]

dow(@td)

min(9)

ALIASNAME

Logic Operators

AND

OR

NOT

IMPORTANT: The logic operators can be used with selection set names, processing labels, and filenames.

Restrictions:

The logic operators can only be used on "if" lines of processing tables, and on the "sentence line" of selection sets.

Examples:

Selection Set:

If: not selsetname

Then: 15="O" ; end

Processing Label:

Secure If: @id eq "root" or @id eq "fred"

Then: x="OK" ; end

If: not secure

Then: x="BAD" ; exit("1")

Filename:

Then: lookup invoices k=ky i=a -ng

If: not invoices

Then: show "@There are no records in the invoice file.";end

Operator

Description

AND

The AND operator designates that more than one condition must be met.

OR

The OR operator designates that either condition must be met.

NOT

The NOT operator designates that the condition must not be TRUE.

Math Operators

- PLUS SIGN
- MINUS SIGN
- MULTIPLICATION SIGN
- DIVISION SIGN
- EXPONENT SIGN

Description:

+	Adds fields or literals
-	Subtracts fields or literals
*	Multiplies fields or literals
/	Divides fields or literals
^	Raises a number to the power of an exponent

EVALUATION OPERATORS

SEMICOLON
 COLON
 APOSTROPHE
 QUOTES
 PARENTHESES
 SQUARE BRACKETS
 SPACES
 CASE
 OTHER OPERATORS (" $<$ ", "{", "%") - see STRING/EXPRESSION MANIPULATION OPERATORS

EQ, GT, LT, GE, LE, CO

Syntax:

Left side of test... EVALUATOR ...Right side of test

Description:

These are relational operators allowed only on "if" lines. You may use them to test how one value relates to another. They stand for what they appear to "spell." EQ means "equal to," GT means "greater than," LT means "less than," GE means "greater than or equal to," LE means "less than or equal to," and CO means "contains."

IMPORTANT: Remember that the evaluation operators (EQ, NE, GT, GE, LT, LE, CO), are ALWAYS tests and may only be used on "if" lines. For example:

```
If: 3 eq "14"
```

will work properly, and:

```
Then: 3 eq "14"
```

has no meaning, and will generate a syntax error.

IMPORTANT: Numeric Compare vs. ASCII Compare

All of these evaluation operators perform the proper tests based on the type of data used in the evaluation. In other words, if you are testing one numeric field against another numeric field, the evaluation will be numerical. If you are testing ASCII values (non-numeric data), the comparison will be an ASCII comparison. This is an important concept to understand when performing tests.

@keyT If:

```
Then: aa(2)="bb"
```

```
If: aa gt "13"
```

```
Then: show "@Yes, bb is greater than 13" ; end
```

The test shown above will prove true, and this may not be what you would expect. If the dummy variable aa is filled with the literal string "bb", this test will prove TRUE. This is because the ASCII value of "bb" is higher than the ASCII value of "13". Just be careful to compare numbers against numbers if that is what you need to do.

Numeric compares (numeric data) will work the way you would expect. "3" will be less than "749". ASCII compares may not work the way you might expect, that is, until you get used to the ASCII sorting order employed by all data base applications.

To further explain this difficult concept, examine the following:

@keyT If:

```
Then: aa(2)="S"
```

```
If: aa gt "Broadway"
```

```
Then: show "@Yes, it is" ; end
```

This will test TRUE, simply because "S" is higher in ASCII value than "B", the first letter in Broadway. The "S" comes after (or higher than) "Broadway" in an ASCII sort. In the same way, the string "1111" would come before "2" in an ASCII sort (or be less than 2 in an ASCII compare). Think of the alphabet or the numbers, not the length, when you are thinking about an ASCII compare or an ASCII sort.

The EQ Operator

IMPORTANT: Evaluation vs. Assignment

The EQ operator is special in one regard. The equal sign maybe substituted for this operator. It is STRONGLY recommended that you do not use this convenience, as it will make your processing tables harder to read. If you do use the equal sign "=" in place of the EQ operator, you will be using the equal sign to perform double duty on your processing tables. Be aware of the significant difference between the two functions evaluation and assignment.

= Evaluates to the same value in a compare. (Only on "if" lines.)

or

= Assigns the value on the right side to the field on the left side.

In other words:

```
Then: aa="Hello"
```

the above code sets the value of field aa equal to "Hello", and:

```
If: 3 = "C"
```

tests for equality between the contents of field 3 and "C". Again, it is STRONGLY suggested that you do NOT ever do this. Instead, always use only the EQ operator to test for equality, as in:

```
If: 3 eq "C"
```

This will make your code much easier to read, since the EQ operator and other relational tests are NOT ever allowed on "then" lines, and you will know that every "=" means assignment.

The CO Operator

NOTE: The CO operator is very powerful and can be used in many different ways.

```
If: "TheForNotIfAnd" co 3
```

```
Then: show "@Yes, it does" ; end
```

If field 3 exactly contains any of the little words (The, For, Not, If, And), the test will prove TRUE. Of course, if field 3 is equal to "hef" or "rNo" or "fan", it will also prove true. Be careful when doing such "backward" compares.

IMPORTANT: All the evaluation operators do not take case into account. Therefore, "good", "Good", "gOOd", etc., are all the same for purposes of the compare.

S

; **U Description:**

Separates statements.

Examples:

```
Then: aa=min(2);screen "4" ; end
```

Use good judgment when separating statements on a line. If it makes more sense to put the statements on separate lines, there is no problem with doing this.

```
Then: a=4 ; aa=44 ; bb="fred" ; cc="temp" ; display ; RETURN
```

might be written as:

```
Then: a=4 ; aa=44 ; bb="fred" ; cc="temp" ; display
```

```
Then: RETURN
```

This places the RETURN command on a line by itself, and using upper case makes it stand out even more. Any tricks like this you can use to make your code easier to understand are usually desirable. As with everything else in programming, consistency is more important than the style content.

: **COLON**

Description:

Performs alias assignment. In other words, substitutes one way of referring to an item for another way of referring to that item.

Examples:

```
Then: BALANCE:14
```

The above sets the alias name BALANCE equal to field 14. BALANCE may then be used interchangeably with field 14 elsewhere on the table.

```
Then: dim TOTALS[10]:14
```

The above sets (or aliases) a 10 element array called TOTALS to overlay the file starting at field 14. The first element in the array would be equal to field 14, the second would be equal to field 15, etc., up to the 10th element.

' APOSTROPHE

Description:

An apostrophe placed anywhere on an "if" or "then" line, makes the rest of the line a comment or remark. These comments are ignored by filePro.

Restrictions:

You can NOT use a comment in a "label" field.

Examples:

```
If: tot(2) gt "4" 'is total of field 2 greater than "4".
```

It is STRONGLY suggested that you use comments liberally. Try to explain in words what the processing is going to accomplish. This will help you a great deal when reading old code, and it's invaluable to others who might have to read your code.

It is allowable to have only comments in a processing element.

```
If: 'The following lines get a free record from invoice file
```

```
Then: 'and post the customer code and new inv# to the new record
```

```
Then: lookup invoice r=free -e
```

Then: invoice(1)=CUSTCODE; invoice(2)=nn

Comments are what separate great programmers from good ones.

"" QUOTATION MARKS

Description:

Indicate a literal. Literals are any string of characters, that are not an expression for something else. They are meant to be used exactly as they appear between the quotes.

Examples:

Then: ab="Hello"

Then: xx=asc("J")

It is VERY IMPORTANT that you DONT use quotes around a filePro field, either real fields or dummy fields. It is equally IMPORTANT that you DO use quotes around numbers which are meant to be literal values.

Correct:

Then: 4="96" 'sets field 4 equal to the value "96"

Unusual (but acceptable):

Then: 4=96 'sets field 4 equal to the value found in field 96

It may be that you WANT to do the "Unusual" statement shown above for some reason. This is fine, as long as you are SURE you know the difference between these two very different lines of code.

Completely wrong:

Then: "4"=96 'this can never be done, it's illegal

() PARENTHESES

Description:

Parentheses perform several different functions.

a) Parentheses change operator precedence (as in algebra).

Then: aa=(ab + "1")/2

Expressions inside parentheses are resolved first. The above example would yield a much different result if written as:

Then: aa=ab + "1"/2

This is because multiplication and division are performed before additions and subtractions.

b) Parentheses hold arguments.

Then: aa=avg(14)

The value in field 14 on each record encountered is passed to the AVG function.

c) Parentheses hold variable filenames, screen names, and fields.

If: 3="V"

Then: ff="vendors"

If: 3="C"

Then: ff="customers"

Then: lookup (ff) r=free -ep

or

Then: ff="/tmp/file.asc"

Then: export ASCII (ff) -X

or

Then: screen (ff),(gg) 'puts cursor on screen "test" in field "2"

[] SQUARE BRACKETS

Description:

Square brackets have only one use on filePro processing tables. They may be used with the DIM command instead of parentheses. It is STRONGLY recommended that you do this to make your code easier to read. Every time you refer to arrays during their creation or use, employ square brackets.

Examples:

Then: `dim inv[10]`

Then: `inv[3] = 14`

The above are better than:

Then: `dim inv(10)`

Then: `inv(3) = 14`

this is because, it would be immediately obvious that `inv[3]` refers to an array rather than a lookup field. (NOTE: While it is true that the subscript of an array is "usually" in quotes, the above code is legal. Using a real field as a subscript of an array is valuable only in limited circumstances, and truly it is only then that an array element might look like a lookup field, but using square brackets for arrays nonetheless keeps them clearly set apart from lookup in general.)

SPACES SPACES ON PROCESSING TABLES**Description:**

SPACES have limited significance on processing tables. Unless they are a part of a literal (within quotes or a quoted string), or part of a command and its arguments, SPACES do not change the way a filePro processing table will compile.

Examples:

Then: `aa=24+"1"`

Then: `aa = 24+"1"`

Then: `aa= 24 + "1"`

The above are all identical and legal.

Then: `aa= 2 4 + "1"`

The above is not legal, and will yield a syntax error.

Of course, any processing table command or function, and any filename or alias must be followed by a space.

You can not use:

Then: `lookupfilenamek=li=a`

but you could use:

Then: `lookup filename k=li=a`

However, it is STRONGLY suggested that you do not use SPACES improperly. For example, one small change in the above code makes it not work.

Then: `lookup filename k=ii=a`

The above will not pass a syntax check. Even though there might be an `indexI` (capital I) and even though filePro normally ignores case, it will get confused by the above.

If you are unsure, it is usually best to use a SPACE when you are in doubt.

FilePro will not compile a faster table if you try to remove all the unnecessary spaces. Use them liberally to make your code easier to read.

If: `23co oc`

The above example will work, but it would be much better written as:

If: `23 co oc 'does field 23 contain the value in dummy OC?`

and

Then: `16=3*mid(1,"1","2")+aa/"-1"`

might be easier to read as:

Then: `16 = 3*mid(1,"1","2") + aa/"-1"`

Whichever way you decide to use SPACES on processing tables, the very best thing you can do is be consistent. If you put a SPACE before and after each semicolon, do it all the time.

Then: `4=b;aa="Ma" ; gosub xxx; k="" ; show "@hello"; end`

The above example would be much easier to read as:

Then: `4=b ; aa="Ma" ; gosub xxx ; k="" ; show "@hello" ; end`

or

Then: `4=b;aa="Ma";gosub xxx;k="";show "@hello";end`

or

Then: `4=b; aa="Ma"; gosub xxx; k="" ; show "@hello"; end`

Since SPACES can be used in such a variety of ways, it is simply a matter of good style to pick a convention and use it all the time. Your coding will be easier to read and debug if you use SPACES in a consistent manner.

NOTE: If you redirect processing with the GOTO command, anything following that command (even if separated by a semicolon) will NEVER be executed. The GOTO command is always the last executed command of any line.

Also, the file-handling functions LOOKUP and EXPORT can not have anything following them on the line.

IMPORTANT: When you use @wlf and @wef processing, there is a small set of commands, one of which MUST be used to close-off or end the @when processing. These commands are END, SCREEN, ESCAPE, RESTART and SKIP. @when processing MUST have one of these commands to end its work. Placing a semicolon and any commands after one of these closure commands while inside @when processing will result in these commands NEVER being executed. This is without doubt the major cause for improperly working tables among beginners and experts alike.

```
@wlf2 If:
```

```
Then: 5="OPEN" ; screen 2,7 ; gosub totals ; display ; end
```

The GOSUB, DISPLAY, and END commands will never be executed because the SCREEN command is one of the special ones that ends an @when routine.

Conversely, and equally IMPORTANT, the following code demonstrates a major source of problems:

```
@wlf2 If:
```

```
Then: 5="OPEN" ; popup update -,3
```

```
@wlf5 If: 5="CLOSED"
```

```
Then: aa=aa+"1" ; display ; end
```

Many filePro programmers first assume that since they have put the user onto SCREEN 3 with this code, the next time the user saves the record, the INPUT processing table will start at the top. It will not! It will start running right where it left off immediately after the POPUP UPDATE command. This is because the @when processing @wlf2 has not been properly closed with one of the special closure commands listed above. The code as shown here will incorrectly "fall through" to the @wlf5 processing that you do not want to run unless the user is actually leaving field 5.

CASE UPPER AND LOWER CASE

Description:

Upper and lower case letters are interchangeable on filePro processing tables. They have no special meaning except in literals, and filenames under Unix.

Examples:

```
Then: AA=aa+"1"
```

```
Then: Aa=aA+"1"
```

```
Then: aa=aa+"1"
```

The above are all identical and legal.

```
If: 3 eq "Mary"
```

```
If: 3 eq "mary"
```

The above are identical and test the same. This is because filePro ignores case when doing comparisons. (Unless you use the [COMPARE](#) function.)

However,

```
Then: show "Hello"
```

```
Then: show "HELLO"
```

will show exactly what is contained inside the quotes, respecting the case. This is how "literals" work. They mean "literally" what is contained within the quotes.

String/Expression Manipulation Operators

AMPERSAND
LESS THAN SIGN
LEFT FRENCH BRACE
PARENTHESES

& AMPERSAND

Description:

The & joins two fields, starting the second field at exactly the end of the defined length of the field on the left.

Examples:

Then: aa(10)="Smith" ; bb(10)="Joe" ; cc(20)=bb&aa ; show cc

would yield:

Joe Smith

because the fields are joined (concatenated) at the defined length of the field on the left, regardless of how much of that field is filled.

Then: aa(8,mdy)/=@td ; show aa&4

would yield:

10/04/97ACME (assuming field 4="ACME")

< LESS THAN SIGN

Description:

The < joins two fields by pushing the field on the right towards the left until there is exactly one space between the two fields. This operator is often called the "push left" operator.

Examples:

Then: aa(10)="Smith" ; bb(10)="Joe" ; cc(20)=bb<aa ; show cc

would yield:

Joe Smith

because the fields are joined, leaving 1 space between the data, regardless of how much of each field is filled.

Then: aa(10)="123" ; bb(10)=" XYZ" ; cc(20)=aa<bb ; show cc

would yield:

123 XYZ

Again, 1 and only 1 space is left between the joined fields, because that is the way the push left operator "<" works.

{ FRENCH BRACE

Description:

The { joins and pushes left, leaving no spaces. Because this operator works in the same manner as the "push left" operator but leaves no spaces between the joined fields, it is called the "squeeze left operator".

Examples:

Then: aa(10)="Joe" ; bb(10)="Smith" ; show aa{bb

would yield:

JoeSmith

() PARENTHESES

Description:

The set of () groups and orders evaluations for the purpose of clarity and precedence of operation.

Examples:

If: ((3 gt aa) and (4 eq "C")) or (BALDUE gt "0" or 19 co bb)

Then: show "@ Customer has outstanding balance."

Print Precedence Directives

The following listing shows, which print controls, have precedence. The highest on the list are PRINT commands on processing tables as these are the last thing to run before the output is sent wherever it is going, the spooler, a file, etc. The next highest priority is what is specified on the command line, i.e., "-p /tmp/filename" will override all settings (except print commands in processing). PFPRT will override any of the lower print directives, but will be overridden by the two above it, and so forth. Using this logic, you can set a group of defaults for your users and most standard reports, yet override these conventions at will by using any of the higher options. The best choice for most filePro output is to designate where it should "normally" go directly on the options page of the output format. User's can generally be well directed with a PFSPOOL command. Again, special print jobs can be directed with PFPRT/C or -p on the command line.



























- Print command in processing
- Command line options
- Printer designated on output format
- PFPRT
- PFPRINTER (default filePro printer)
- PRINTER1 (default filePro printer, if no default printer)
- PFSPOOL (default if no default printer destination)
- LPDEST

Default systemspooler "lp -s"

Special Key Labels

These Special keyboard keys are available to filePro for various uses. They can be tested using the @sk system maintained field. Testing for these keys with @wlf, @wef, @wuk processing is a valuable tool for building robust applications. Using these keys with PUSHKEY and within SHOW statements allows you to build very user-friendly programs.

Special Keys - For use with SHOW, @SK and PUSHKEY











Keyboard Key	Show Code	@sk TEST	Used in PUSHKEY	Description	Screen Display (PC/ansi)
F1	\K0	INSC	[INSC]	Insert character	
F2	\K1	DELC	[DELC]	Delete character	
F3	\K2	INSL	[INSL]	Insert line	
F4	\K3	DELL	[DELL]	Delete line	
ESC	\K4	SAVE	[SAVE]	Save	
F5	\K5	DUPL	[DUPL]	Duplicate	
PageUp	\K6	UTAB	[UTAB]	Page UP	
PageDown	\K7	DTAB	[DTAB]	Page DOWN	
Shift-Tab	\K8	LTAB	[LTAB]	Left Tab	
Tab	\K9	RTAB	[RTAB]	Right Tab	
Ctrl-End	\KA	CLEF	[CLEF]	Clear to end of field	
F6	\KB	DMAP	[DMAP]	Browse lookup	
Ctrl-L	\KC	DRAW	[DRAW]	Clear & redraw screen	
F7	\KD	PRTC	[PRTC]	Cursor to 1 after data	
F8	\KE	DPRT	[DPRT]	Tab cursor in field	
	\KF	CRON	[CRON]		
Alt-F9	\KG	RVON	[RVON]	Reverse video on	
F9	\KH	GRAF	[GRAF]	Cursor to end of data	
F10	\KJ	HELP	[HELP]	Help	
BREAK	\KY	BRKY	[BRKY]	Break	
ENTER	\KZ	ENTR	[ENTR]	Enter	
UParrow	\Ku	CRUP	[CRUP]	Up arrow	
DOWNarrow	\Kd	CDWN	[CDWN]	Down arrow	
LEFTarrow	\Kl	CLFT	[CLFT]	Left arrow	
RIGHTarrow	\Kr	CRGT	[CRGT]	Right arrow	
HOME	\Kh	HOME	[HOME]	Home	

(* NOTE: \Ku, \Kd, \Kl, \Kr, \Kh are CASE SENSITIVE.)

Note : INSC also applies for the **Insert Key**, and DELC also applies to the **Delete key**.

SHOWCODES

The following screen displays how special "show codes" can be used inside the message portion of the SHOW commands. (Show codes work in all formats of SHOW including: show popup, showctr, showtocol.)

SHOW CODES																					
Code	Action	Example processing	Displays																		
\r	reverse	show "\r Hello \r"	Hello																		
\Kn	Keylabel	show "Press \r \Kd \r"	Press DNarrow																		
\Xnn	Hex code	show "\XaBQuien es Se\Xa4or FilePro?"	¿Quien es Senfior FilePro?																		
\Gn	Graphics	show "Lower right corner \r \G3 \r"	Lower right corner 																		
<p>The graphic box characters match the keypad numbers in a one-to-one picture of a box with a center cross inside it. 0=horizontal, period=vertical.</p> <table border="0"> <tr> <td>7 8 9</td> <td>┌ ─ ┐</td> <td>0 -</td> <td>789</td> <td></td> <td>0 -</td> </tr> <tr> <td>4 5 6</td> <td>└ ─ ┘</td> <td>. </td> <td>456</td> <td></td> <td>. </td> </tr> <tr> <td>1 2 3</td> <td>└ ─ ┘</td> <td></td> <td>789</td> <td></td> <td></td> </tr> </table>				7 8 9	┌ ─ ┐	0 -	789		0 -	4 5 6	└ ─ ┘	.	456		.	1 2 3	└ ─ ┘		789		
7 8 9	┌ ─ ┐	0 -	789		0 -																
4 5 6	└ ─ ┘	.	456		.																
1 2 3	└ ─ ┘		789																		

The following table displays SHOW codes available only to systems that support color.

Color Systems Only.

(See next table for color values of "n")

\I	Toggle high- intensity on/off
\Ann	Set attribute (background+foreground))
\Fn	Set foreground color
\Bn	Set background color
\C	Display characters only (no color)

Color Attribute Code (\Ann, \Fn, \Bn) Values of "n" for above table.

Foreground/ Background	Color	Foreground Only	Color
"n"	Color	"n"	Color
0	black	8	gray
1	blue	9	bright blue
2	green	A	bright green
3	cyan	B	bright cyan
4	red	C	bright red
5	violet	D	bright violet
6	brown or yellow	E	yellow or bright yellow
7	bright gray or white	F	white or bright white

\Ann - Sets background and foreground colors. The first "n" is the background color; the second "n" is the foreground color. If only one "n" is used, it sets the foreground color, and the background color is set to black (0).

If you want to reset the background and foreground colors to the default colors, use a dash for "n" (\A-).

\Fn - Sets foreground color. Use values 8 to F for "n". If you want to reset the foreground color to the default color, use a dash (-) for "n".

\Bn - Sets background color. Use values 0 to 7 for "n". If you want to reset the background color to the default color, use a dash (-) for "n".

Blinking Colors

Using values 8 to F for the background does not affect the current background color, but will cause the foreground color to blink instead.

The following table displays key label codes (\Kn) values for "n", @SK labels, and DOS keys for SHOW codes.

(Note: Case is significant.)

"n"	@SK	DOS KEY	"n"	@SK	DOS KEY
0	INSC	Ins or F1	D	PRTC	F7
1	DELC	Del or F2	E	DPRT	F8
2	INSL	F3	F	CRON	N/A

3	DELL	F4	G	RVON	Alt-F9
4	SAVE	ESC	H	GRAF	F9
5	DUPL	F5	J	HELP	F10
6	UTAB	PgUp	Y	BRKY	Break
7	DTAB	PgDn	Z	ENTR	Enter
8	LTAB	Shift-tab	u	CRUP	Up arrow
9	RTAB	Tab	d	CDWN	Down arrow
A	CLEF	Ctrl-End	l	CLFT	Left arrow
B	DMAP	F6	r	CRGT	Right arrow
C	DRAW	Ctrl-L	h	HOME	Home

TCP/IP Functions (not included in filePro Lite)

handle2 = **ACCEPT**(handle)

Accept a connection on a socket. Returns a positive number containing the new handle, zero indicating no more sockets are available, or a negative number containing the error number.

status = **BIND**(handle,port [,address [,family]])

Binds a name to a socket. Note that the parameters are not identical to the C bind() function, but the port/address/family correspond to the struct sockaddr_in members sin_port, sin_addr, and sin_family, respectively. Port can be given by a number, or a name defined in /etc/services. Address can be an IP address, or any name known to the DNS server, or null for INADDR_ANY. Family defaults to AF_INET. Returns zero on success, or a negative number containing the error number.

status = **CONNECT**(handle,port,address [,family])

Initiate a connection on a socket. Note that the parameters are not identical to the C bind() function, but the port/address/family correspond to the struct sockaddr_in members sin_port, sin_addr, and sin_family, respectively. Port can be given by a number, or a name defined in /etc/services. Address can be an IP address, or any name known to the DNS server, or null for INADDR_ANY. Family defaults to AF_INET. Returns zero on success, or a negative number containing the error number.

name = **GETPEERNAME**(handle)

Returns the name of the connected peer, or "" on failure.

name = **GETSOCKNAME**(handle)

Returns the name of the socket, or "" on failure.

status = **LISTEN**(handle [,backlog])

Listen for connections on a socket. Backlog defines the maximum length to which the queue of pending connections may grow, and defaults to 1. Returns zero on success, or a negative number containing the error number.

status = **RECV**(handle,dest [,len [,flags [,noblock]]])

status = **RECVLINE**(handle,dest [,len [,flags [,noblock]]])

Receive a message from a connected socket. Fills in the field in "dest" with the result. Returns the number of characters received, or a negative number containing the error number. A maximum of "len" characters will be read. **RECVLINE**() will read up to len characters, or until a new line character is received. The new line character, if any, will be discarded for **RECVLINE**(). "Len" defaults to the length of the destination field. "Flags" defaults to zero and should currently not be given any other value. "Noblock" defaults to zero. If a non-zero value is passed, the function will return immediately, even if less than "len" bytes are received.

status = **SELECT**(nhandle [,read_array [,write_array [,except_array [,timeout]]])

where

nhandle = Maximum number of handles to use from each array. If the array is shorter than nhandle, then the entire array is used. Array entries of zero are ignored.

read_array = Name of array holding handles to check for read.

write_array = Name of array holding handles to check for write.

except_array = Name of array holding handles to check for exceptions.

timeout = Timeout, in microseconds.

Notes: If timeout is not specified, or is zero length, the function will not return until at least one handle meets the specified criteria. A timeout of zero means it will return immediately, regardless of the states of the handles. (This is one of the rare instances in filePro where a null numeric value is not the same as zero. This is part of the functionality of the select system call, which is duplicated in filePro's **SELECT** function.)

Only socket handles can be checked. While *nix systems allow any file handle to be passed to select(), Windows allows only socket handles. A future filePro update may allow non-socket handles to be passed to **SELECT**().

status = **@SELECT.READ**(handle)

status = **@SELECT.WRITE**(handle)

status = **@SELECT.EXCEPT**(handle)

After executing a **SELECT**(), these functions are used to determine if the specified handle satisfied the select criteria. The return value is either "0" for false, and "1" for true. For example, **@SELECT.READ(MySocket)** will return "1" if MySocket has data ready for reading. (Assuming, of course, that MySocket was in the read_array passed to the last **SELECT** function.)

Note that if the last **SELECT**() call did not get passed the corresponding array, the result of these functions are undefined. For example, a **SELECT**(nhandle,read_array,,except_array) call followed by **@SELECT.WRITE(x)** will be undefined, as no write_array was passed.

status = **SEND**(handle,value [,flags])

status = **SENDLINE**(handle,value [,flags])

Send a message to a connected socket. Flags defaults to zero and really shouldn't be used yet. **SENDLINE**() appends a new line character to the message. Returns the number of characters sent, or a negative number containing the error number.

handle = **SOCKET**([,family [,type [,protocol]]])

Creates a socket of the specified type. Returns a positive number containing the handle to the socket, zero if there are no more sockets available, or a negative number containing the error number.

Note: currently, the parameters are ignored, and the socket is always (AF_INET,SOCK_STREAM,0)

status = **SOCKETCLOSE**(handle)

Closes the socket. (Note: even though Unix socket handles and open file handles are interchangeable, this is not the case under Windows hence the requirement for a separate close function.)

As of 5.7.02, this command will now release the license used.

systemhandle = **SOCKETTOSYS**(handle)

Returns the corresponding system handle number for an open socket, or a negative error number if "handle" isn't a valid filePro socket.

errno = **SOCKETERROR**()

Returns the error code of the last socket-related error. The number is system-dependent.

handle = **SYSTOSOCKET**(systemhandle)

Given an operating system handle that corresponds to a socket, this creates a filePro handle to it. (Example: a server daemon has already listen()ed and accept()ed the connection, and spawned filePro to handle the connection.) Note that there is no check that the handle is really a valid socket. Returns zero if no filePro sockets are available.

SOCKETS Sample Applications (not included in filePro Lite)

Sample SOCKETS applications are included on the installation media.

The samples demonstrate use of most of the functions through a simple interactive application.

What is provided

menus filePro folder	~/filepro/menus	tcip Menu
TCPIP filePro folder	~/filepro/tcip	tcip sample
TCP/IP filePro folder	~/filepro/tcipphp	Contains this help.

Installing the Samples

Select the appropriate install for your system from the source media.

WINDOWS:

Click on Start, Run and d:\setup.exe where d: is your CD drive letter. After setup is complete, copy ~/filepro/menus/tcip to ~/fp/menus or other 'menus' directory as set by the pfmenu environment variable.

AIX/UNIX/LINUX:

```
tar xvf fTcip
```

After extracting the files, copy ~/filepro/menus/tcip to ~/fp/menus or other 'menus' directory as set by the pfmenu environment variable.

Run setperms script establish proper permissions.

Note: You can elect to install the TCP/IP Server and Client on different operating systems, on different machines with the same operating system or on a single machine.

Run the Menu

Run the tcip menu from the 'Run User Menu' option of the filePro main menu.

SOCKETS-Server

After completing the installation, you will need to know the IP address of the machine where you intend to run the TCP/IP Server.

The sample application should be run initially from the Menu but if you can also run from a prompt. The sample application uses the dclerk, input processing and screen.0

The menu provides options for launching TCP/IP Server, TCP/IP Client and this help. The same filePro file 'tcip', processing table and screen is used for the Server and Client. The menu simply uses a menu parameter to control which is launched for each option.

When selecting the TCP/IP Server from the menu, you will see a prompt "Enter port number to listen on:". Enter a port number ie. 4300 and then you should see message "Waiting for Connection on port nnnn ..."

Note: For Windows, you can use "localhost" or IP address "127.0.0.1".

SOCKETS-Client

After the TCP/IP server is listening, select the TCP/IP client option from the menu on the client machine or launch another session on the localhost machine if you are running in localhost mode. When prompted enter the Server IP address or localhost (127.0.0.1) if testing in local mode. Enter the port number you specified when launching the TCP/IP server.

If all has gone well, you should see the Server and Client connect. Once connected, you can send messages between the Server and Client. Received text is displayed in reverse video mode on both the Server and Client. You should be able to type anything into the screen on either machine and have it echoed on the other.

Press key [F1] to exit and shutdown the socket connection.

For more detail, look at the "input" processing for file 'tcip' and refer to function descriptions in the filePro TCP/IP help.

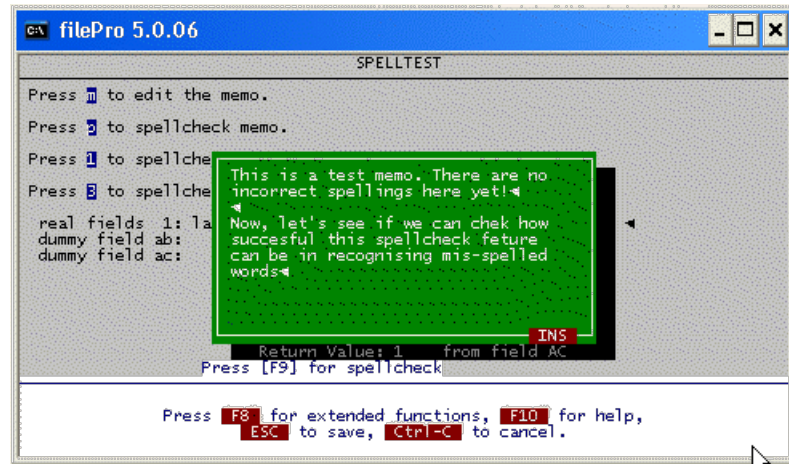
Licensing

Note that the "Sockets" functions are separately licensed as an option.

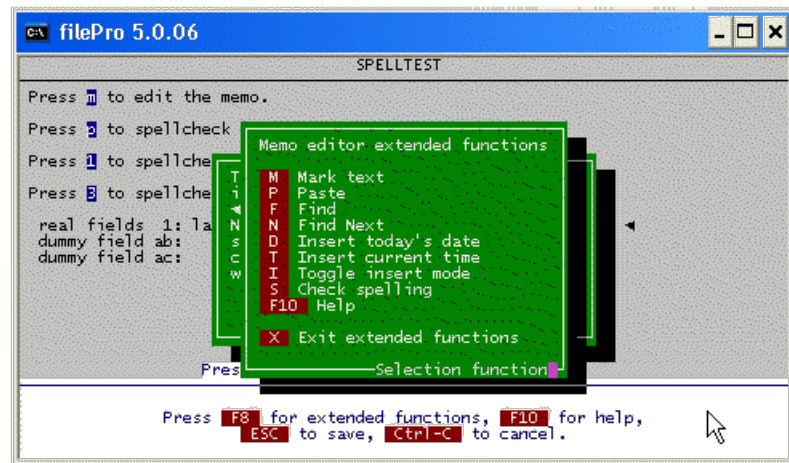
Contact our sales office by e-mail sales@fptech.com or (800) 847-4740 for a price quote on this option.

Spell Check - Memo Fields

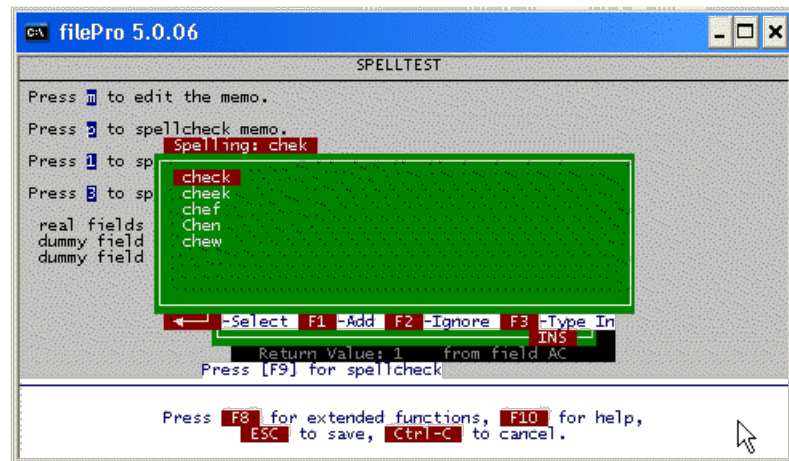
You can spell check memo fields through the Memo Editor extended functions [F8] [S] "Check Spelling" function of the memo editor. This will spell check the entire memo, and will use the chooser for any misspellings.



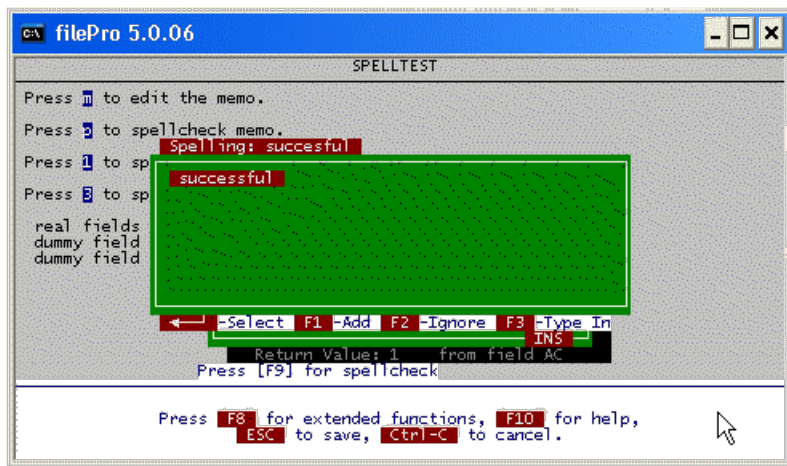
When the memo field is on the screen, Press [F8] to present the Memo editor Extended functions and then press [S] to check spelling.



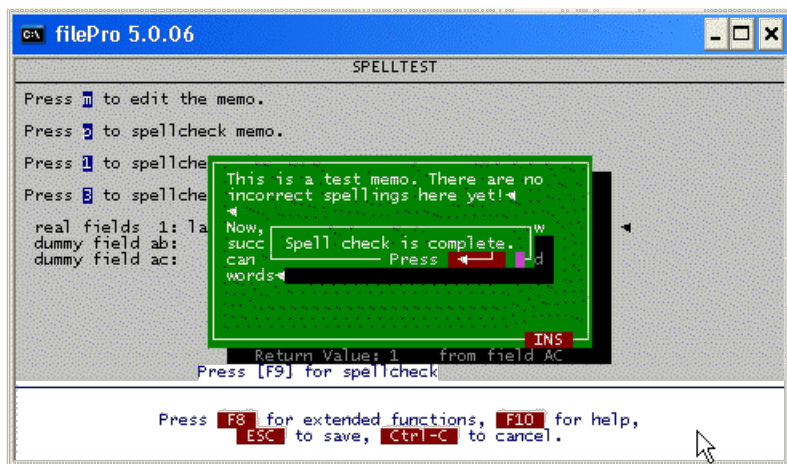
The first misspelled word is presented. Select the correct spelling and press [Return].



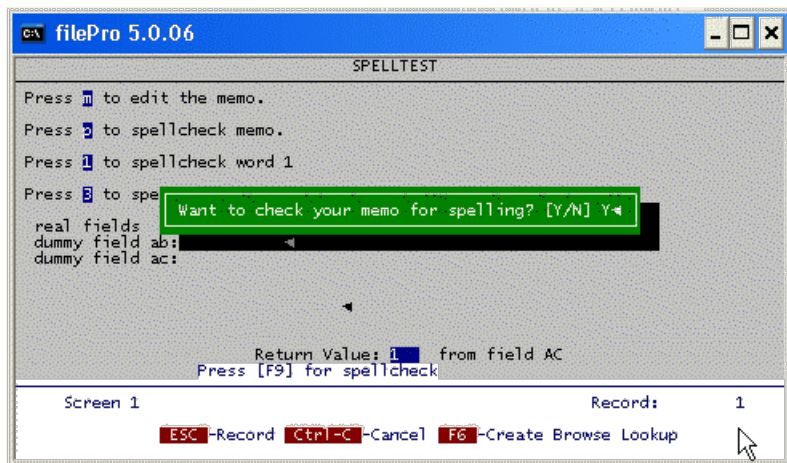
Each misspelled word will be presented.



If the word is not in the dictionary, you can elect to add to your personal dictionary with [F1], ignore by pressing [F2] or Type In the correct spelling by pressing [F3]. When spell check is complete, a message is presented to indicate this.



You can also add a message box or input statement to force spell check of a memo field.



Personal dictionaries are maintained in the ~/fp/spell directory by default as /fp/spell/userID.txt where user ID is the *NX login ID or Windows user name. You can override the default path and filename with the PFSPELLUSERLIST variable. Make sure that the path exists and that you properly set permissions if you override the default path. You can edit the personal dictionaries with the Spell Editor program included in the filePro Utilities menu.

Note: The F8 options of the memo editor have been tweaked as well. The "I"/"T" options for "insert time"/"toggle insert" have been swapped. Now, "D" inserts date, "T" inserts time, and "I" toggles insert mode. You can set PFMEMOEDITOLDKEYS=ON to revert to the old settings.

Spell Check - Using Processing

You can use spell checker in processing tables to check the spelling of any field including memo fields.

```
status = SPELLCHECK(field [ , chooser, row , col, height, width ] )
```

The return value is:

```
1 = good  
0 = bad  
negative = error
```

Except for "field", all parameters are optional. If "chooser" is non- zero, then misspelled words will cause the chooser to appear, where you can pick the correct spelling.

If "chooser" is zero, or not specified, then the spellcheck() function only checks if the words are spelled correctly or not and does not give you the option of correcting them

Example

- Clarify use of @fd e.g. aa = @fd
- SPELLCHECK(aa,"1")

Use SPELLCHECK to check the spelling of a memo.

The 64-bit versions of filePro now use the industry-standard "hunspell" spell-check library. As noted by the hunspell website:

"Hunspell is the spell checker of LibreOffice, OpenOffice.org, Mozilla Firefox 3 & Thunderbird, Google Chrome, and it is also used by proprietary software packages, like Mac OS X, InDesign, memoQ, Opera and SDL Trados."

<http://hunspell.sourceforge.net/>

System Maintained Fields

System maintained fields are a powerful feature of filePro that allow you to easily access information maintained by the O/S (Operating System) such as current date and time. These fields also include a myriad of filePro functions such as record creation date, date a record was last updated, cursor position on a screen.

Some of the system maintained fields are unique to specific operating systems i.e. "Name of the current user" was not available for Windows but only for UNIX or LINUX operating systems prior to the 5.0.12 release.

System maintained fields can be used on reports and screens just like real fields and dummy fields. The difference between system maintained fields and others is that they cannot be directly manipulated but are controlled by the system or filePro functions. Also, some system maintained fields are not available for all types of processing so check for any usage restrictions in the notes for the system maintained field that you decide to use.

@AF

Associated field instance.

@B4

4-digit year equivalent of @BD or last batch update. (MM/DD/YYYY)

Example:

02/24/2000

@BD

Last batch update. (MM/DD/YY)

Exit key for browse lookup, msgbox and errorbox.

Browse Lookup - When defining a browse lookup, you can specify the keys that the user can press to exit. The key pressed by the user is captured in the system-maintained field "@BK" and can be acted upon to control processing. You will want to identify a processing label, when defining the browse lookup, and normally test for both valid and invalid keys pressed by the user.

Example:

You have an invoicing application and need to add new customers, update the customer's address and phone number, or simply view more customer detail than will fit on a browse line.

When defining the browse lookup, specify the valid exit keys "xkeys" to perform each of your desired options. You would probably use "A" to Add a customer, "U" to update, "V" to view. These exit keys will be shown as "xkey=AUV" when defining the browse.

If the user presses [A], when testing the value in @BK on a condition line, you might execute processing to check if a customer record already exists, and if NOT, popup a blank customer data-entry screen.

If the user presses [U], display a screen with limited update capability for changing just the address and phone numbers.

If the user presses [V], popup a "View Only" screen in the customer file.

If one of the three valid keys are not pressed, remove the browse and continue.

@BK and special keys such as ESC, Enter, Break - Some special keys can be acted on just as any other key pressed by the user but you must use the key code that applies for the special keys e.g [SAVE], [ENTR], [BRKY] and determine the special key pressed by checking the value of @sk. Refer to the [Special Key Labels](#) topic in this manual for a complete list of Special Key Labels.

MSGBOX and ERRORBOX - These processing commands let you display a message in a popup window on the screen. It will display a string "message" in a popup window at a designated position on the screen until the user presses ENTER or a key from a specified keylist. The key pressed by the user is captured in the system-maintained field "@BK" and can be acted upon to control processing.

@BR

The currently highlighted browse lookup row.

System maintained fields, @sn, @fd and @br are for use mainly on the INPUT processing tables. However, they are available on output processing tables as well, but they will always be equal to null on these tables. This may be used to advantage.

Example:

You have an AUTOMATIC table that does things for you during Inquire, Update and Add, but it has no value when running output formats, and shouldn't be used. (Or it only does things related to INPUT processing and would just be a waste of time to run during an OUTPUT process. You can prevent the AUTOMATIC table from running during output processing by putting the following code on the first line of the AUTOMATIC table. No code after this line will run during OUTPUT, but there is always a "screen-name" during INPUT, so the rest of the AUTOMATIC table will run then.

```
If: @sn eq "" 'if there is no screen
Then: end      'don't go any further.
Then: 'other automatic table functions
```

@BT

The time of a record's last batch update. (HH:MM:SS)

Note: The time is stored in 2 second intervals

@C4

4-digit year equivalent of @CD or date that the current record was created. (MM/DD/YYYY)

Example:

02/24/2000

@CB

The name of the user that first created this record. Unix/Xenix only)

@CD

The date that the current record was created. This field is blank for new records. (MM/DD/YY)

@CO

Returns the column of the first position of the field just entered or left.

Version Ref : 5.0

Note: Only available on INPUT processing tables.

@CP

Provides the position of the cursor in the field you just left

Note: Only available on INPUT processing tables.

@CT

The time that the current record was created. (HH:MM:SS)

Note: The time is stored in 2 second intervals

@DT

The current date, spelled out. (e.g., "Mar 6, 1999")

@FD

Field cursor is in.

Note: Only available on INPUT processing tables.

@F

Provides the current file name

@FN

The current format name.

Note: Only available on OUTPUT processing tables.

@D

The name of the current user. (Unix only prior to version 5.0.12)

Note: @ID will only contain the first 8 characters of the username on Windows systems.

See P **FIDLEN=nnnn** Version reference 5.8.00

@LC

Line count. The number of lines currently printed on this page.

Note : Only available on OUTPUT processing tables.

@LI

Returns the current line number within processing.

Version Ref: 5.0

@OS

The operating system name. @OS values are "DOS", "UNIX", "HP-UX", "AIX", "DGUX" and "LINUX".

Notes: "DOS" is returned as the value for Windows operating systems. LINUX was added in version 5.0

@PC

Provides Current Printer Comment.

@PD

Provides Current Printer Destination. This also corresponds to PRINTERCOMMAND

Note: Prior to release 5.0.14 dscreen would resolve @PD to only 20 characters, when it's really 80.

@PM

Parameter passed to processing table with -r flag.

Usage:

```
/app/fp/dreport my_file -fp my_output -a -u -r "hello!"
```

In output processing, @PM now contains "hello!"

@PN

The current page number.

Note: Only available on OUTPUT processing tables.

@PR

Provides Current Printer Name

@PT

Provides Current Printer Type

@PW

Reads the contents of the -rw command line flag. The -rw flag can be used with *clerk and *report.

Usage:

```
/app/fp/dreport my_file -fp my_output -a -u -rw "hello!"
```

In output processing, @PW now contains "hello!"

@PX

Reads the variable contents of the `-rx` command line flag. The `-rx` flag can be used with `*clerk` and `*report`.

Usage:

```
/app/fp/dreport my_file -fp my_output -a -u -rx "hello!"
```

In output processing, `@PX` now contains "hello!"

@PY

Reads the variable contents of the `-ry` command line flag. The `-ry` flag can be used with `*clerk` and `*report`.

Usage:

```
/app/fp/dreport my_file -fp my_output -a -u -ry "hello!"
```

In output processing, `@PY` now contains "hello!"

@PZ

Reads the variable contents of the -rz command line flag. The -rz flag can be used with *clerk and *report.

Usage:

```
/app/fp/dreport my_file -fp my_output -a -u -rz "hello!"
```

In output processing, @PZ now contains "hello!"

@QU

Provides the current qualifier name

@FN

Provides the current record number

@RP

Provides Number of Records Processed

@PS

The number of records selected so far within this subtotal level.

Note: Only available on OUTPUT processing tables.

@SF

The contents of the subtotal field

Note: Only available on OUTPUT processing tables.

@SH

The name (heading) of the subtotal field.

Note: Only available on OUTPUT processing tables.

@SK

Special key can be used with INKEY, but avoid INKEY, it is CPU intensive, use WAITKEY instead.

"@sk" is used with when-processing. When-processing fills this field with the special key-label of the key last pressed.

Browse lookups will also pass SAVE, BRKY and ENTR to @sk if users press any of these 3 keys while the highlighted bar is visible.

@RO

The row/column of the first position of the field just entered or left.

Version Ref : 5.0

Note: Only available on INPUT processing tables.

@SN

The current screen name.

Note: Only available on INPUT processing tables.

@T4

4-digit year equivalent of @TD or the current date. (MM/DD/YYYY)

Example:

02/24/2000

@TD

The current date. (MM/DD/YY)

@TM

The current time. (HH:MM:SS)

@TN

The current processing table name.

@TS

The total number of records selected for this report.

Note: Only available on OUTPUT processing tables.

@U4

4-digit year equivalent of @UD or date the current record was last updated.

(MM/DD/YYYY)

Example:

02/24/2000

@UB

The name of the user that last updated this record. (Unix/Xenix only)

@UD

The date the current record was last updated. (MM/DD/YY)

@UT

The time that the current record was last updated. (HH:MM:SS)

Note: The time is stored in 2 second intervals

@VR

Version Ref: 4.8.04 (Run-time)

Version number of *clerk/*report being run. This is the full version number e.g. "4.9.01K3DN9" that can be moved into a (4,1) field to get the major version (i.e. "4.9") for comparisons.

Version Ref: 5.8.03 (Menus)

@VR is now available to use on menus. By entering in @VR into the version field of a menu, it will display the major version number of the release installed.

NOTE: For Windows, enter \$@VR and for NIX enter %@VR%

@VR2 (Menus only)

Version Ref: 5.8.03

Version number of *clerk/*report being run. This is the full version number e.g. "4.9.01K3DN9" that can be moved into a (4,1) field to get the major version (i.e. "4.9") for comparisons.

@DV (Menus only)

Version Ref: 5.8.03

Version number of *clerk/*report being run. This is the full version number e.g. "4.9.01K3DN9" that can be moved into a (4,1) field to get the major version (i.e. "4.9") for comparisons.

@GUI

x=@GUI.PAUSE()

Pauses automatic screen updating while in GI/Web.

x=@GUI.RESUME()

Resumes automatic screen updating while in GI/Web.

These functions are useful for hiding the output of some system commands while running processing. Output can sometimes cause undesired effects on a filePro screen or prevent some commands from updating the screen correctly. This should correct some text being displayed while outside of filePro's messaging.

Example:

Then: x=@GUI.PAUSE()

Then: SYSTEMcommand

Then: x=@GUI.RESUME()

@VF

This is the @welf function to trap the Virtual Keys available in fileProWeb
See fileProWeb documentation for more details.

@WORDWRAP[]

Returns text for the most recent WORDWRAP() call.

Syntax:

```
xx = @WORDWRAP[linenum]
```

Returned value:

Text is returned for the specified line in "linenum" and based on the WORDWRAP() option used.

Example:

```
then: dim wraptext(999)
then: xx = WORDWRAP(9,"70","1"); c(3,.0)="1"
loop
if: c lt xx
then: wraptext(c) = @WORDWRAP(c); c=c+"1"; goto loop
then: end
```


SYSINFO Version 5.7.00

xx = @SYSINFO.TIME-- Returns information about the current time, in several formats:

@SYSINFO.TIME.RAW()

Returns the "raw" time, as the number of seconds since "the epoch", which is defined as "1970-01-01 00:00:00 UTC".

@SYSINFO.TIME.UTC()

@SYSINFO.TIME.LOCAL()

Returns the current time, either from UTC, or the local timezone, formatted as:

YYYY:MM:DD:HH:MM:SS:w:yyy:d

Where:

YYYY - The year

MM - The month, as a number from 01 to 12

DD - The day of the month (01-31)

HH - The hour of the day (00-23)

MM - The minute of the hour (00-59)

SS - The second of the minute (00-60)

w - The day of the week (1-7)

yyy - The day of the year (001-366)

d - "1" if daylight savings time is in effect, else "0"

@SYSINFO.TIME()

Same as @SYSINFO.TIME.LOCAL()

xx = @SYSINFO.WIDTH

The width of the filePro "screen". Currently always "80".

xx = @SYSINFO.HEIGHT

The height of the filePro "screen". Currently always "24".

Other systeminfo may be added in future updates.

System Arrays

Version Ref : 5.0

@ALLFILES[]

Contains a list of all files open by filePro, even those closed due to dynamic open/close. Some entries may be blank, representing an unused (i.e. closed) file.

Note: also known as "@OCFILE[]"

@CMDLINE[]

Contains the command line passed to *clerk/*report.

Note: also known as "@ARGV[]"

@DIRLIST[]

Description:

Contains the full directory line as returned by NEXTDIR() including the following:

Description	Length	Starting Position
Format	32	01
Space	01	33
Extension	10	34
Space	01	44
File Size	14	45
Space	01	59
File Date	10	60
Space	01	70
File Time	8	71
A or P (am/pm)	01	79
Space	01	80
Full Name	32	81

Table 1 - NEXTDIR Line

Example:

```
dl = @dirlist(n)
```

where dummy field "dl" is set to the value for @dirlist array subscript number "n".

@CMDLINE[]

Contains the command line passed to *clerk/*report.

(Note: also known as "@ARGV[]")

@DIRLIST_EXT[]

Contains the 10 character extension name (as returned starting in column 34 by NEXTDIR). See table 1.

@DIRLIST_FILENAME[]

Contains the 32 character full name (as returned in the last column by NEXTDIR). See table 1

@DIRLIST_NAME[]

Contains the 32 character format name (as returned in the first column by NEXTDIR). See table 1

@FPFILES[]

Contains a list of all open filePro files. Some entries may be blank, representing an unused (i.e. closed) file.

Note : @FPFILES["1"] is always the main filePro file.

@FSTAT[]

Contains information about the last successful EXISTS()

Function. (See your O/S manual for a description of each entry.) All entries are numeric, except for the date/time entries, which are returned as (10,mdyy/) and (8,TIME) respectively,

- [0] = length of array (currently always 14)
- [1] = st_dev

- [2] = st_ino
- [3] = st_mode
- [4] = st_nlink
- [5] = st_uid
- [6] = st_gid
- [7] = st_rdev
- [8] = st_size
- [9] = st_atime_date (date portion of st_atime)
- [10] = st_atime_time (time portion of st_atime)
- [11] = st_mtime_date
- [12] = st_mtime_time
- [13] = st_ctime_date
- [14] = st_ctime_time

Notes:

If no call to EXISTS() has been executed, or the last EXISTS() failed, then the contents of @FSTAT[] are undefined.

Array subscript "0" contains the length of the array. Out-of-bound subscript references return null.

System arrays are read-only.

@FUZZY[]

Contains information about the most recent fuzzy browse lookup.

Currently, the only info available is the record number, via @FUZZY.RECNO [subscript]

@LICENSE[]

5.6 Contains license information.

Subscripts:

- [1] = A comma-separated list of licensed features.
- [2] = Name
- [3] = Serial number
- [4] = Platform
- [5] = Product name
- [6] = Version
- [7] = User count
- [8] = License start date (YYMD format)
- [9] = License end date (YYMD format)

Example

```
msgbox "Licensed user Name"<@license["2"]
```

Note: License start date and License End date are null if the license does not specify a start or end date.

@SYSFILES[]

Contains a list of all open files. Some entries may be blank, representing an unused (i.e. closed) file.

@UNAME[]

Returns the systemuname() information. The meaning of each entry is system-specific.

- [1] = sysname
- [2] = nodename
- [3] = release
- [4] = version
- [5] = machine

For Window's systems, the following values are returned.

Sysname: Win9x, WinNT, or Windows (if no specific type).

Nodename: The value from the C programming function "GetComputerName()".

Version: Value of szCSDVersion from the C programming function "GetVersionEX()".

Machine: blank

Trigger Processing

User definable "trigger" processing can be written to occur when the designated event happens. In filePro, these types of processing are given labels and that is where the processing starts up when the trigger is pulled. This is called "when processing" and spoken aloud as "when enter selection", "when entering field nnn", "when leaving field nnn", and sometimes "@when leaving field n" or "@key X", etc. The @starting all these labels may be considered to mean "at", or "at the moment when this trigger is activated".

Triggers Used Only on Input Processing Tables

@keyX

When user presses key "x" (or "X", case is not significant) while sitting at the "Enter Selection >" prompt.

@key can be used with any plain text character key on the keyboard. Special keys (TAB, ENTER, SAVE, etc.) are not usable with @key.

This processing can be activated by the user while sitting on a record with the cursor at "Enter Selection" prompt (or custom prompt).

Pressing the designated key locks the record (multi-user), runs the AUTOMATIC table first, and then runs the @key processing, starting at its label following the table directions until an END is encountered. At this point, the AUTOMATIC table is run again and the process is finished.

IMPORTANT: The AUTOMATIC table's first job is to clear all regular dummy variables. If you set a variable inside your @key processing and expect it to retain that value, you must make this variable "global". The AUTOMATIC table does not clear global variables.

NOTE: Be sure that there is always a "then" line above your @key process which will not allow the INPUT table processing or other @when processing to "fall through" to this @key code. This will, of course, give you undesirable results. If there is no INPUT processing and you want to build an @key routine, put an END statement on line 1's "then" line, and build the @key starting from line 2. This way, when the user SAVES the record, the @key process will not be run as if it were the INPUT table processing.

IMPORTANT: On multi-user and network systems, @key processing "locks" the current record. If other users are running reports or processing that select this record, they will be stopped in their tracks until the lock is removed. For this reason, you should not put up processing MENUS with @key. Also, be careful not to put up message boxes or the like which require that the user press a key to bring them down. More likely than not, someone will inadvertently leave such a message on the screen and go to lunch. This hangs everyone else up who needs access to that record. A better idea would be to use the SLEEP command and display the message for a reasonable amount of time and then bring it down within your own @key code.

Example:

```
@keyT if:
then: show "@The time is"<@tm ; end
```

The T is capitalized only for clarity when reading the code. (Processing tables are insensitive to cAsE)

The above will certainly work, but it forces the user to press ENTER to finish the @key process. Instead, try:

```
@keyT if: 'on Unix, use sleep "3"
then: show "The time is"<@tm ; sleep "3000" ; end
```

This will show the time for 3 seconds and then automatically end the processing, leaving nothing up to the user.

Of course, if the @key process is supposed to put the user into UPDATE mode so he can add to or modify the record or perform some other task, this is perfectly acceptable and a great use for @key processing. You have to assume that the user will finish such a task before leaving for lunch.

@key processing can be tricked into working from other parts of the INPUT processing table. This is done with

PUSHKEY. As in:

```
if:
then: pushkey "T" ; end
```

When this code is encountered, filePro waits for control to be returned to the user, and just before it accepts input from the keyboard, it generates the designated key (in this case T). Since the @keyT is triggered by this event, it happens just as if the user had activated it himself. If there is no @keyT processing on the INPUT table, nothing will happen.

Processing to do after record is displayed, but just before the prompts are displayed and the cursor is placed at the "Enter Selection" prompt.

This processing does NOT lock the current record. This means you can not write to real fields within an @entsel routine.

A common use for @entsel is to combine it with the -d flag of clerk. This flag tells filePro to clear the standard prompts off the bottom of the record screens. This allows you to put up custom prompts of your own. Using @entsel will put these custom prompts up at the appropriate times.

For example: The following command line and code will display a custom set of prompts. It would be up to you to make sure there are appropriate @keys for each of the displayed prompts. Just changing the prompts will not stop filePro from intercepting the keys it normally does like (H)ardcopy, (B)rowse, (U)pdate and so forth. They will still be active unless you build @keyH @keyB @keyU processing of your own. Any combination of filePro (factory installed @keys) and custom @keys of your own is acceptable, just remember that yours will override the factory supplied @keys.

```
rclerk filename -sl -d
@entsel if:
then: show "\r P \r-Print, \r U \r-Update, \r M \r-Modify, \r X \r-Exit"
if:
then: end
```

IMPORTANT: Remember that @entsel processing does NOT lock the record on multi-user systems, so you can not make assignments to real fields within the @entsel code. In other words, a real field can not be the left side of an equals statement.

@menu

Processing done just before the clerk menu appears.

@menu processing can only be used on INPUT tables. It is activated just before the clerk menu appears on the screen and, this is **IMPORTANT**, it is activated EVERY TIME the clerk menu is just about to appear on the screen. This can be very confusing at first. More than likely, you will want the @menu code to work just once, the FIRST time the user ENTERS this file. You probably do not want to run it again when the user is LEAVING the file. This can be accomplished by setting global dummy variables to control the processing.

Let's assume that you want the @menu processing to bring the user to "his" record in a file. You do this by asking for his initials and then moving him directly to that record with "lookup -". This can be done as follows:

```
@menu if:
then: input popup in(3,allup) "What are your initials? "
err if: in eq "" 'no answer,
then: errorbox "Sorry, those initials not on file." ; exit
if:
then: lookup - k=in i=A-nx
if. not -
then: in="" ; goto err
```

This code will allow the user to put in a good set of initials (on file) and take him directly to that record. Blank initials or not on file initials will kick him out of the program. However, there is one glitch. When the user is done with the task on his record and wants to exit the program, the @menu program keeps asking for his initials, it doesn't know he wants out! Of course, users could press ENTER and the program would throw them out for blank initials, but there is a better way, and it is the key to making @menu work in most other situations as well. Set a "flag" to tell @menu what to do on the way in and on the way out. Do this with a global dummy variable. It means adding 2 lines above what is already shown, and something like this should probably be in most @menu routines.

```
@menu if: feq "1"
then: exit
if:
then: f(1,,g)="1"
if:
then: input popup in(3,allup) "What are your initials? "
err if: in eq "" 'no answer,
then: errorbox "Sorry, those initials not on file." ; exit
```

```

if:
then: lookup - k=in i=A -nx
if: not-
then: in="" ; goto err

```

This flag will be empty the first time into clerk, so the user can get to his record. But, this time when he tries to exit the program, the value of f will be "1" and it will happen.

If you want the user to be able to stay in the program and make use of the clerk menu, this EXIT could be an END statement. This would leave the user at the clerk menu. Remember, @menu processing executes all the @menu code it encounters and then puts up the clerk menu. If all it encounters is END, it does nothing and puts up the menu.

There are a variety of things you can do with @menu in terms of taking users in and out of files and records. A powerful combination is to use @menu with PUSHKEY. Examine the following:

```

@menu if: f eq "1"
then: exit if:
then: f(1,,g)="1"
askin if:
then: input popup in(1) "1) Indexes\n2) BALDUE selection set\n====> "
if: @sk eq "BRKY" or @sk eq "ENTR"
then: exit
if: in ne "1" and in ne "2"
then: goto askin
if: in eq "1"
then: pushkey "4" ; end
if: in eq "2"
then: pushkey "22LBALDUE[ENTR][ENTR]" ; end

```

This asks the user whether he wants to go into the file using indexes, or to go directly to the first record that matches the criteria on the BALDUE selection set. This is NOT very robust or useful code, but it shows how PUSHKEY and @menu can work together.

NOTE @menu processing is performed while the user is standing on record 0. There are no real fields available.

@update

Starts input processing before user is put into update mode. This processing allows you to do things AFTER the user has pressed "U" to UPDATE the record (or whatever key is assigned to perform this function) and BEFORE the cursor hits the first field on the cursor path. (Or before it hits any @wef processing which may be attached to that first field).

@wefNNN When entering field "nnn".

@wlfNNN When leaving field "nnn".

@wblNNN

Responds to browse lookup key (keylabel DMAP) when in field "nnn". If there is an existing browse lookup for this field, it is not performed.

@whpNNN

When the user presses help key (keylabel HELP) when in field "nnn".

@wukNNN

When the user presses any of keylabels (DPRT, GRAF, INSL, DELL) when in field "nnn". Once the user presses one of these user keys, you can use @sk to determine which of the 4 were pressed. The default user key is DPRT.

@w??*

The "*" character can be used in place of the "nnn" field number in WEF, WLF, WBL, WHP, WUK.

Triggers Used Only on Output Processing Tables

@done

The output processing subroutine labeled by @DONE will be executed when the output processing is done, even if no records are selected. Executes after @WGT label and after all output has been sent to the printer/spooler.

@wbrkN

When breaking a report for subtotal "n".

@wgt

When breaking a report for a grand total.

@done

Description:

The output processing subroutine labeled by @DONE will be executed when the output processing is done, even if no records are selected. Executes after @WGT label and after all output has been sent to the printer/spooler.

@ONCE

Description:

Executed once at the beginning of the output phase in *report, before the first record is processed with output processing. Also, will now execute in *clerk, prior to the first execution of @MENU.
Note: doing a "lookup-" in @ONCE will cause you to jump to that record and skip the normal *clerk menu, just as a "-" lookup-" in @MENU will do.

Version Ref: 5.0

Note: Although @ONCE in *report is documented as being run prior to any output being done, it was run while sitting on the last record read during the sort/select process prior to release 5.0.14. Some people thought that this meant that it was sitting on a selected record.

@ONCE has now been fixed to be not sitting on any record. However, some people depend on their incorrect interpretation of the old behavior, so setting FFOLDONCE=ON will "revert back" to a modified version of the old behavior, where it will now be run while sitting on the last record _selected_ during the sort/select process.

Version 6.0.00 - @ONCE now works from all processing tables.

@entsel

Description:

Tells an "Input Processing Table" what to do after record is displayed, but just before the prompts are displayed and the cursor is placed at the "Enter Selection" prompt.

This processing does NOT lock the current record. This means you can not write to real fields within an @entsel routine.

A common use for @entsel is to combine it with the -d flag of clerk. This flag tells filePro to clear the standard prompts off the bottom of the record screens. This allows you to put up custom prompts of your own. Using @entsel will put these custom prompts up at the appropriate times.

For example: The following command line and code will display a custom set of prompts. It would be up to you to make sure there are appropriate @keys for each of the displayed prompts. Just changing the prompts will not stop filePro from intercepting the keys it normally does like (H)ardcopy, (B)rowse, (U)pdate and so forth. They will still be active unless you build @keyH, @keyB @keyU processing of your own. Any combination of filePro (factory installed @keys) and custom @keys of your own is acceptable, just remember that yours will override the factory supplied @keys.

```
rclerk filename -s1 -d
```

```
@entsel If:
```

```
    Then: show "\r P \r-Print, \r U \r-Update, \r M \r-Modify, \r X \r-Exit"
```

```
    Then: end
```

IMPORTANT: Remember that @entsel processing does NOT lock the record on multi-user systems, so you can not make assignments to real fields within the @entsel code. In other words, a real field can not be the left side of an equals statement.

Note : @entsel trigger is used only on INPUT processing table.

@key

Description:

When user presses key "x" (or "X", case is not significant) while sitting at the "Enter Selection >" prompt.

@key can be used with any plain text character key on the keyboard. Special keys (TAB, ENTER, SAVE, etc.) are not usable with @key.

This processing can be activated by the user while sitting on a record with the cursor at the "Enter Selection" prompt (or custom prompt).

Pressing the designated key locks the record (multi-user), runs the AUTOMATIC table first, and then runs the @key processing, starting at its label following the table directions until an END is encountered. At this point, the AUTOMATIC table is run again and the process is finished.

IMPORTANT: The AUTOMATIC table's first job is to clear all regular dummy variables. If you set a variable inside your @key processing and expect it to retain that value, you must make this variable "global". The AUTOMATIC table does not clear global variables.

NOTE: Be sure that there is always a "then" line above your @key process which will not allow the INPUT table processing or other @when processing to "fall through" to this @key code. This will, of course, give you undesirable results. If there is no INPUT processing and you want to build an @key routine, put an END statement on line 1's "then" line, and build the @key starting from line 2. This way, when the user SAVES the record, the @key process will not be run as if it were the INPUT table processing.

IMPORTANT: On multi-user and network systems, @key processing "locks" the current record. If other users are running reports or processing that select this record, they will be stopped in their tracks until the lock is removed. For this reason, you should not put up processing MENUS with @key. Also, be careful not to put up message boxes or the like, which require that the user press a key to bring them down. More likely than not, someone will inadvertently leave such a message on the screen and go to lunch. This hangs everyone else up who needs access to that record. A better idea would be to use the SLEEP command and display the message for a reasonable amount of time and then bring it down within your own @key code.

Example:

```
@keyT  If:
      Then: show "@The time is"<@tm ; end
```

The T is capitalized only for clarity when reading the code. (Processing tables are insensitive to cAsE)

The above will certainly work, but it forces the user to press ENTER to finish the @key process. Instead, try:

```
@keyT  If: 'on Unix, use sleep "3"
      Then: show "The time is"<@tm ; sleep "3000" ; end
```

This will show the time for 3 seconds and then automatically end the processing, leaving nothing up to the user.

Of course, if the @key process is supposed to put the user into UPDATE mode so he can add to or modify the record or perform some other task, this is perfectly acceptable and a great use for @key processing. You have to assume that the user will finish such a task before leaving for lunch.

@key processing can be tricked into working from other parts of the INPUT processing table. This is done with PUSHKEY. As in:

```
      Then: pushkey "T" ; end
```

When this code is encountered, filePro waits for control to be returned to the user, and just before it accepts input from the keyboard, it generates the designated key (in this case T). Since the @keyT is triggered by this event, it happens just as if the user had activated it himself. If there is no @keyT processing on the INPUT table, nothing will happen.

Note: @keyX trigger is used only on INPUT processing table.

@menu

Description:

Processing done just before the clerk menu appears.

@menu processing can only be used on INPUT tables. It is activated just before the clerk menu appears on the screen and, this is IMPORTANT, it is activated EVERY TIME the clerk menu is just about to appear on the screen. This can be very confusing at first. More than likely, you will want the @menu code to work just once, the FIRST time the user ENTERS this file. You probably do not want to run it again when the user is LEAVING the file. This can be accomplished by setting global dummy variables to control the processing.

Let's assume that you want the @menu processing to bring the user to "his" record in a file. You do this by asking for his initials and then moving him directly to that record with "lookup -". This can be done as follows:

```
@menu If:
      Then: input popup in(3,allup) "What are your initials? "
err   If: in eq "" 'no answer,
      Then: errorbox "Sorry, those initials not on file." ; exit
      Then: lookup - k=in i=A-nx
      If: not -
      Then: in="" ; goto err
```

This code will allow the user to put in a good set of initials (on file) and take him directly to that record. Blank initials or not on file initials will kick him out of the program. However, there is one glitch. When the user is done with the task on his record and wants to exit the program, the @menu program keeps asking for his initials, it doesn't know he wants out! Of course, users could press ENTER and the program would throw them out for blank initials, but there is a better way, and it is the key to making @menu work in most other situations as well. Set a "flag" to tell @menu what to do on the way in and on the way out. Do this with a global dummy variable. It means adding 2 lines above what is already shown, and something like this should probably be in most @menu routines.

```
@menu If: f eq "1"
      Then: exit
      Then: f(1,,g)="1"
      Then: input popup in(3,allup) "What are your initials? "
err   If: in eq "" 'no answer,
      Then: errorbox "Sorry, those initials not on file." ; exit
      Then: lookup - k=in i=A-nx
      If: not -
      Then: in="" ; goto err
```

This flag will be empty the first time into clerk, so the user can get to his record. But, this time when he tries to exit the program, the value of f will be "1" and it will happen.

If you want the user to be able to stay in the program and make use of the clerk menu, this EXIT could be an END statement. This would leave the user at the clerk menu. Remember, @menu processing executes all the @menu code it encounters and then puts up the clerk menu. If all it encounters is END, it does nothing and puts up the menu.

There are a variety of things you can do with @menu in terms of taking users in and out of files and records. A powerful combination is to use @menu with PUSHKEY. Examine the following:

```
@menu If: f eq "1"
      Then: exit
      Then: f(1,,g)="1"
askin If:
      Then: input popup in(1) "1) Indexes\n2) BALDUE selection set\n====> "
      If: @sk eq "BRKY" or @sk eq "ENTR"
      Then: exit
      If: in ne "1" and in ne "2"
      Then: goto askin
      If: in eq "1"
      Then: pushkey "4" ; end
      If: in eq "2"
      Then: pushkey "22LBALDUE[ENTR][ENTR]" ; end
```

This asks the user whether he wants to go into the file using indexes, or to go directly to the first record that matches the criteria on the BALDUE selection set. This may NOT be very robust or useful code, but it shows how PUSHKEY and @menu can work together.

Notes :

@menu trigger is used only on INPUT processing table.

@menu processing is performed while the user is standing on record 0. There are no real fields available.

When using a lookup-dash in @MENU routine, it will override -X flags.

@update

Description:

Starts input processing before user is put into update mode. This processing allows you to do things AFTER the user has pressed "U" to UPDATE the record (or whatever key is assigned to perform this function) and BEFORE the cursor hits the first field on the cursor path. (Or before it hits any @w ef processing which may be attached to that first field).

Note : @update trigger is used only on INPUT processing table.

@wbl

Description:

Responds to browse lookup key (keylabel DMAP) when in field "nnn". If there is an existing browse lookup for this field, it is not performed.

Notes: The "*" wild card character can be used in place of a particular field number when using @wbl, i.e., @wbl*. @wbl trigger can only be used on the INPUT processing table.

Field-specific events (@wbl5, @wblA, etc.) take precedence over associated field events (@wbla1, @wblb2, etc.), and events including the wild card (@wbl*). filePro will execute the field-specific event first, then associated field events and finally non-specific "*" events.

@wbrk

Description:

When breaking a report for subtotal "n".

Note : @wbrk trigger can only be used on the OUTPUT processing table.

@wef

Description:

When entering field "nnn".

Note: The "*" wild card character can be used in place of a particular field number when using @wef, i.e., @wef*. @wef trigger can only be used on the INPUT processing table.

Field-specific events (@wef5, @wefA, etc.) take precedence over associated field events (@wefa1, @wefb2, etc.), and events including the wild card (@wef*). filePro will execute the field-specific event first, then associated field events and finally non-specific "*" events.

@wgt

Description:

This trigger occurs when the report process breaks for grand totals. If the environment variable FFWGT0=ON, FilePro will execute the @WGT processing even if no records are selected.

Note : @wgt trigger can only be used on the OUTPUT processing table.

User definable "trigger" processing can be written to occur when the designated event happens. In filePro, these types of processing are given labels and that is where the processing starts up when the trigger is pulled. This is called "when processing" and said as "when Enter Selection", "when entering field nnn", "when leaving field nnn", and sometimes "@when leaving field n" or "@key X", etc. The @ starting all these labels may be considered to mean "at", or "at the moment when this trigger is activated".

There are important rules to follow for each of these types of processing.

@whp

Description:

When the user presses help key (keylabel **H**ELP) when in field "nnn".

Note: The "*" wild card character can be used in place of a particular field number when using @whp, i.e., @whp*. @whp trigger can only be used on the INPUT processing table.

Field-specific events (@whp5, @whpA, etc.) take precedence over associated field events (@whpa1, @whpb2, etc.), and events including the wild card (@whp*). filePro will execute the field-specific event first, then associated field events and finally non-specific "*" events.

@wlf

Description:

When leaving field "nnn".

Note: The "*" wild card character can be used in place of a particular field number when using @wlf, i.e., @wlf*. @wlf trigger can only be used on the INPUT processing table.

Field-specific events (@wlf5, @wlfA, etc.) take precedence over associated field events (@wlfA1, @wlfB2, etc.), and events including the wild card (@wlf*). filePro will execute the field-specific event first, then associated field events and finally non-specific "*" events.

v6.1 (USP 6.0.02)

You can now use: @wlf<letter>* ex. @wlfT* This will apply to any dummy/associated field that begins with 'T' Overrides any other @wlf*

@wuk

Description:

When the user presses any of keylabels (DPRT, GRAF, INSL, DELL) when in field "nnn". Once the user presses one of these user keys, you can use @sk to determine which of the 4 were pressed. The default user key is DPRT.

Note: The "*" wild card character can be used in place of a particular field number when using @wuk, i.e., @wuk*. @wuk trigger can only be used on the INFUT processing table.

Field-specific events (@wuk5, @wukA, etc.) take precedence over associated field events (@wuka1, @wukb2, etc.), and events including the wild card (@wuk*). filePro will execute the field-specific event first, then associated field events and finally non-specific "*" events.

@exit (Version 6.0.00.01)

@exit label to *clerk processing.

This is executed whenever a record is exited or broken out of.

Events that trigger this are 'X' while not in update mode, 'BRKY' while not in update mode, and 'exit' in processing. It is the opposite of @entsel, and is the last thing executed when leaving a record.

NOTE: Assignment of real fields is not allowed, this is similar to @once in that the processing that is executed is NOT sitting on a record, but rather record '0'.

Commands

This provides the filePro version number that added or modified the command. If you are using an older version than identified, the command will either not work or may work differently than advertised. Items with version number 3.x were included in version 3.1 or earlier. (* designates not included in filePro Lite)

Version No.	Command	Description
4.1	ABS()	Returns the absolute value of a number.
5.6	ACCEPT()	Accept a connection on a socket.
4.8	ACCESS()	Returns file access information.
4.8	ACOS()	Returns Arccosine of an angle in radians.
5.0	ACOSH()	Hyperbolic cosine value.
4.8	ADDMONTH()	Add a specified number of months to a date.
6.0	ARCHIVE	Copies a lookup record with all system maintained fields unchanged..
3.x	ASC()	Convert character to ASCII code.
4.8	ASIN()	Returns Arcsine of an angle in radians.
5.0	ASINH()	Hyperbolic Arcsine value.
4.8	ATAN()	Returns Arctangent of an angle in radians.
5.0	ATANH()	Hyperbolic Arctangent value.
3.x	AVG()	Find averages at subtotal/total levels.
5.0	BACKGROUND	Turn ON or OFF the ability to enter background mode with "IG".
4.5	BASE()	Converts between different bases.
3.x	BEEP	Sound the speaker.
5.6	BIND()	Binds a name to a socket using IPv4
5.7	BIND2()	Binds a name to a socket using IPv6
5.0	BLOB	Binary Large Object functions.
4.5	BOM()	Beginning of month.
4.5	BOQ()	Beginning of quarter.
4.5	BOY()	Beginning of year.
3.x	BREAK	Turns BREAK key on or off.
4.0	CALL name	Calls a processing table as subroutine.
4.8	CALL pathname	Calls a processing table in another file.
5.0	CALL NOAUTO	Ignores dummy fields define in automatic processing.
4.5	CEIL()	Performs the ceiling function.
3.x	CHAIN name	Chain to a processing table.
4.8	CHAIN pathname	Chain to a processing table in another
4.5	CHDIR	Change the current directory.
3.1	CHR()	Convert ASCII code to character.
4.1	CLEAR	Sets each element of "array" to blank.
4.1	CLEARB	Removes a lookup window from screen.
4.1	CLEARP	Removes a popup window from screen.
4.5	CLEARS	Clear a SHOW POPUP window.
3.x	CLOSE	Close all files opened with lookups.
3.x	CLOSE file	Close indicated lookup file.
3.x	CLOSE export	Close indicated export.
3.x	CLOSE()	Close a file
4.8	CLOSEDIR()	Close a OPENDIR() system directory for reading.
3.x	CLS	Clear entire screen.
4.5	COMPARE()	Compares two values, with case sensitivity.
5.6	CONNECT()	Initiate a connection on a socket using IPv4
5.7	CONNECT2()	Initiate a connection on a socket using IPv6
3.x	COPY	Copy record to lookup file.
3.x	COPYIN	Copy record from lookup file.
4.8	COPY TO	(enhanced) Copy record from one lookup file to another.
4.8	COS()	Returns cosine for angle in radians.
5.0	COSH()	Hyperbolic cosine.
4.5	CREATE()	Create a new file, and open it.
5.8	CRYPTERROR	gets error code for ENCRYPT() / DECRYPT() failure

5.0	<u>CURSOR</u>	Forces cursor ON, OFF or to default behavior.
5.0.6	<u>CURSORPATH</u>	Allows turning off forced cursor path - fileProGL. Same as MOUSEPATH.
4.8	<u>DACOS()</u>	Returns Arccosine of an angle in degrees.
4.8	<u>DASIN()</u>	Returns Arcsine of an angle in degrees.
4.8	<u>DATAN()</u>	Returns Arc tangent of an angle in degrees.
4.8	<u>DCOS()</u>	Returns cosine for angle in degrees.
3.x	<u>DEBUG</u>	Turn debugger on or off.
4.8	<u>DECLARE</u>	Global and Local dummy variables long names.
5.7	<u>DECODE()</u>	Converts text in a string that was ENCODEd
5.6	<u>DECRYPT()</u>	Decrypt ciphered field data created by ENCRYPT().
3.x	<u>DELETE</u>	Delete record in current file.
3.x	<u>DIM()</u>	Define an array of "n" fields.
1.x	<u>DISPLAY</u>	Redisplay current screen (no update).
4.1	<u>DLEN()</u>	Returns the display length of a string.
4.5	<u>DOEDIT()</u>	Apply an edit to the contents of a field
5.6	<u>DOKEY</u>	Execute a keystroke as if the user pressed it
4.5	<u>DOM()</u>	Day of month.
4.5	<u>DOQ()</u>	Day of quarter.
4.1	<u>DOW()</u>	Returns day of week for a date expression.
4.5	<u>DOY()</u>	Day of year.
4.1	<u>DROP</u>	Drop records from browse lookup
4.5	<u>DROP ALL</u>	Drop all remaining records in browse lookup.
4.8	<u>DSIN()</u>	Returns sine for angle in degrees.
4.8	<u>DTAN()</u>	Returns tangent for angle in degrees.
4.8	<u>DTOR()</u>	Converts angle in degrees to radians.
3.x	<u>EDIT()</u>	Returns the edit-type of a field.
5.7	<u>ENCODE()</u>	Converts the contents of a field
5.6	<u>ENCRYPT()</u>	Encrypt field data.
5.7	<u>ENCRYPTED()</u>	Returns true if a lookup file was ENCRYPTed
3.x	<u>END</u>	Stop processing.
4.5	<u>EOM()</u>	End-of-month.
4.5	<u>EOQ()</u>	End-of-quarter.
4.5	<u>EOY()</u>	End-of-year.
4.1	<u>ERRORBOX</u>	Error box position.
4.5	<u>ERRNAME()</u>	Return the name of the specified system error.
3.x	<u>ESCAPE</u>	Save current screen; use with @w ef/@w lf.
4.1	<u>EXISTS()</u>	Returns error level for files.
4.5	<u>EXIT</u>	Exit filePro, and set the exit value.
4.8	<u>EXP()</u>	Anti-log (e^n) for natural log function.
4.8	<u>EXP10()</u>	Anti-log (10^n) for common log function.
4.8	<u>EXPORT</u>	(Enhanced) Create spin-off files for other programs.
4.5	<u>FIELDEDIT()</u>	Return edit name of field.
4.5	<u>FIELDSLEN()</u>	Return the length of the specified field.
4.5	<u>FIELDNAME()</u>	Return the name of the specified field.
5.6	<u>FIELDNUM()</u>	Returns the field number of a FIELDNAME in a lookup
4.5	<u>FIELDVAL()</u>	Returns the value of a specified field.
5.0	<u>FILENAME()</u>	Return the filename of specified handle.
4.5	<u>FILESIZE()</u>	Return the number of bytes in a file.
4.5	<u>FLOOR()</u>	Performs the floor function.
4.8	<u>FLUSHKEY</u>	Clears pending keystrokes.
3.x	<u>FORM</u>	Print a form (auto & input proc only)
4.0	<u>FORMM</u>	Print a form without closing spooler.
4.1	<u>FRAC()</u>	Return the fractional portion of a number.
4.8	<u>FREESPACE()</u>	Returns the amount of free disk space if less than 2GB

5.7	FSTAT()	Fills @FSTAT[] system array with open file info
5.6	GET16() / GET32()	Reads data from the buffer of a binary file
4.5	GETCWD()	Return the current directory name.
4.1	GETENV()	Return value of environmental variable.
3.x	GETNEXT	Used with LOOKUP for repetitive lookups.
5.6	GETNONCE()	Gets the "nonce" last used with ENCRYPT().
5.6	GETPEERNAME()	Returns the name of the connected peer (sockets).
5.6	GETPID()	Returns the process ID of the current process.
5.6	GETPPID()	Returns the parent process ID of the current process.
3.x	GETPREV	Used with LOOKUP for repetitive lookups.
5.6	GETSOCKNAME()	Returns the name of the socket.
3.x	GOSUB	Branch to subroutine label; used with RETURN.
4.5	GOSUB OF / GOTO OF	Gosub/goto one of a list of labels.
3.x	GOTO	Branch to element label.
5.0	GUI	Test if you are running a fFclient session e.g. fileProCl.
3.x	HARDCOPY	Print the current screen.
5.7	HASH()	Applies the HASH encryption to a field
3.x	HELP	Display help screen of section "name".
4.8	HTML	Create HTML files using filePro HTML functions.
5.0	HTML:xxx	HTML functions added and enhanced.
4.8	HTMLERRNO()	Return error code for the last HTML/JSFILE element.
3.x	IMPORT	Open a non-filePro file for data importing.
3.x	INKEY	Get next keystroke, if available.
3.x	INPUT	Prompt for user input.
4.0	INPUTPW	Input password (puts # when typing).
4.1 / 5.0	INPUT POPUP	Input using a popup window .
4.1	INPUTPW POPUP	Input a password in popup window .
4.1	INSTR()	In-string search.
4.1	INT()	Return the integer portion of a number.
4.5	ISLEAP()	Determine if a given year is a leap year.
4.8	IXCOMMENT()	Return the comment for the index specified.
4.8	IXSORT()	Return sort information for a specified index.
4.8	JSFILE	Create sequential ASCII files.
3.x	LEN()	Find length of field or expression.
4.1 / 5.0	LISTBOX()	Display a popup window containing a list.
5.6	LISTEN()	Listen for connections on a socket.
4.8	LOCKED()	Tests if the lookup failed due to a locked record.
4.8	LOG()	Natural or naperian log function.
4.8	LOG10()	Common or base 10 log function.
4.5	LOGTEXT	Writes text to a file specified by PFLOGFILE
3.x	LOOKUP	Get data from another file.
4.0		Qualifier Enhancement.
4.5		-s switch. Do not display "No Records Exist ..." message
4.0		BRW=lines,row,col size and location of window .
4.0		SHOW=KEEP; Keep window after selecting record.
4.0		SHOW=ONLY; only show the browse lookup.
4.1		SHOW=PKEEP; keep window & highlight bar.
4.0		POP=screen; Popup "screen" when user presses 'V'.
4.0		XKEY=keys; keys which will take browse down.
4.1		FILL=top/bot; Direction to fill browse window .
4.1		PRC=label; prc label to goto at each record.
4.0		number of records for fuzzy search.
4.8		Selection Lookup enhanced to allow use of expressions including the new extended long name variables.

4.1	MAX()	Enhanced to return max value from a list of values.
4.5	MDAY()	Returns the number of days in a given month.
5.0	MEMO	Memo & plain text BLOB functions.
3.x	MENU	Create menu in processing.
5.7	MERGEVAL	result = MERGEVAL (importname, field number
4.1	MESGBOX	Display a popup message box
5.0.6	MESSAGE	Refer to fileProGI Developer Toolkit.
3.x / 5.0	MID()	Find and copy the middle of a field.
4.1	MIN()	Enhanced to find minimum value of field from a list of values.
4.1	MOD()	Return the remainder of a expression.
5.0.9	MOUSE PATH	Force screen cursor path ON or OFF when using a mouse with fileProGI client. Default is ON. Same as CURSORPATH
4.1	MSGBOX	Display a popup message box.
4.8	NEXTDIR()	Return system directory information.
4.8	NOT HTML	Tests if last HTML/JSFILE element failed.
4.5	NUMFIELD()	Return the number of fields in a file.
4.5	NUMRECS()	Return the number of records in a file.
4.5	OPEN()	Open a file.
4.8	OPENDIR()	Return the number of files or list of files
4.0	OUTS	Sends data to a serial port.
3.x	PAGE	Force end of page when printing reports.
4.8	PI()	Returns the value of PI or 3.14159265
4.0	POPUP	Displays screen of a lookup file.
4.0	POPUP UPDATE	Allows update of a screen of a lookup file.
3.x	PRINT	Print current record.
3.x	PRINTER "command"	Send output to "command". (LINUX/UNIX)
3.x	PRINTER FILE	Send output to file "filename".
5.0	PRINTER FLUSH	Flushes all printer buffers
3.x	PRINTER LOCAL	Send output to terminal printer.
4.0	PRINTER NAME	Send output to printer "name".
3.x	PRINTER RESET	Reset the printer.
4.0	PRINTER TYPE	Sets printer type.
4.0	PUSHKEY	Places "keystrokes" into the keyboard queue.
5.6	PUT16() / PUT32()	Writes binary data to a buffer
4.5	PUTENV	Store a value in the environment.
4.1 / 5.0	RAND()	Return a pseudo-random number.
4.5	READ()	Read from a file.
4.8	READBROWSE()	Returns browse format as a data string
4.5	READLINE()	Read a line of text from a file.
4.8 / 5.0	READOUTPUT()	Reads the text portion of a report.
4.8 / 5.0	READSCREEN()	Reads text portion of a screen.
4.8	RECLEN()	Record length of the specified file.
5.6	RECV()	Receive a message from a connected socket.
5.6	RECVLINE()	Receive a message from a connected socket.
4.5	REMOVE()	Remove a file from the disk.
4.5	REPEAT()	Repeat a string of characters.
3.x	RESET	Reset selected records @wgt.
5.0	RESET @PN	Reset Page Number back to 1
3.x	RESTART	Restart processing from start of table.
3.x	RETURN	End subroutine; go to element after GOSUB.
4.8	RTOD()	Converts angle in radians to degrees.
5.0	SAVE	Set the save option ON or OFF.
4.5	SCREEN	(enhanced) Allows a cursor position to be specified.
4.5	SEEK()	Set the current location within an opened file.

3.x	<u>SELECT</u>	Select record.(sort/selection processing)
5.6	<u>SELECT()</u>	Select a socket handle.
5.6	<u>SEND()</u>	Send a message to a connected socket.
5.6	<u>SENDLINE()</u>	Send a message to a connected socket.
4.5	<u>SET</u>	Fills an array with a specified value
3.x	<u>SHOW</u>	Show a message.
4.5	<u>SHOW POPUP</u>	Display a message in a popup window .
5.0.6	<u>SHOW RAW</u>	Show a message without filePro interpretation.
4.5	<u>SHOWCTR()</u>	Center a message on the screen.
4.5	<u>SHOWTOCOL()</u>	Specify show ending column for display.
4.5	<u>SIGN()</u>	Return the sign of a number.
4.8	<u>SIN()</u>	Return sine for angle in radians.
5.0	<u>SINH()</u>	Hyperbolic sine value.
3.x	<u>SKIP</u>	Skip to next field; used with @wef only.
4.5	<u>SLEEP</u>	Stop processing for a specified amount of time.
5.6	<u>SOCKET()</u>	Creates a socket of the specified type.
5.6	<u>SOCKETCLOSE()</u>	Closes the socket.
5.6	<u>SOCKETTOSYS()</u>	Returns the corresponding system handle number for an open socket.
5.6	<u>SOCKETERROR()</u>	Returns the error code of the last socket-related error.
3.x	<u>SORT</u>	Control sorting level and method.
4.8	<u>SORTINFO()</u>	Sort information for a specified output.
4.1	<u>SOUNDEX()</u>	Return the soundex code for the string "exp".
5.6	<u>SPELLCHECK</u>	Check spelling of memos and other fields.
4.1	<u>SQRT()</u>	Returns the square root of "n".
5.0	<u>STATUS</u>	Retrieves the status of a handle
4.8	<u>STRTOK()</u>	First occurrence of any string.
4.5	<u>SWITCHTO</u>	Switch to a different screen.
4.5	<u>SYNC</u>	Flush any disk writes pending on a file.
3.x	<u>SYSTEM</u>	Execute a OS command from processing.
5.6	<u>SYSTEM()</u>	Returns the exit value of command issued.
4.1	<u>SYSTEM NOREDRAW</u>	No screen redraw while processing.
5.6	<u>SYSTOSOCKET()</u>	Given an operating system handle that corresponds to a socket, this creates a filePro handle to it.
4.8	<u>TAN()</u>	Returns tangent for angle in radians.
5.0	<u>TANH()</u>	Hyperbolic tangent value.
4.5	<u>TELL()</u>	Get the current location in the file.
4.8	<u>TOHTML()</u>	Converts filePro HTML characters to HTML equivalents.
3.x	<u>TOT()</u>	Get a total or subtotal on field.
5.0	<u>TVM_I()</u>	Time-Value-Money - Interest.
5.0	<u>TVM_N()</u>	Time-Value-Money - Number of periods.
5.0	<u>TVM_PV()</u>	Time-Value-Money - Present Value.
5.0	<u>TVM_PMT()</u>	Time-Value-Money - Payment.
5.0	<u>TVM_FV()</u>	Time-Value-Money - Future Value.
3.x	<u>UPDATE</u>	Put user in update mode.
3.x	<u>USER</u>	Send/receive data to/from user program
4.5	<u>VIDEO</u>	Turn video updates on and off.
3.x	<u>WAITKEY</u>	Wait for next keystroke.(See also INKEY)
4.5	<u>WOM()</u>	Week of month.
4.5	<u>WOQ()</u>	Week of quarter.
5.6	<u>WORDWRAP()</u>	Gets wrap information for memo & non-memo fields,
4.5	<u>WOY()</u>	Week of year.
4.5	<u>WRITE</u>	Write data to a file.
4.5	<u>WRITE()</u>	Write to a seek position in a file.
4.5	<u>WRITELINE()</u>	Write a line of text to an external file.
4.5	<u>XLATE()</u>	Translate characters.

Environment Variables

This provides the filePro version number that added or modified the environment variable. If you are using an older version than identified, the variable will not work or may work differently than advertised. Items with version number 3.x were included in version 3.1 or earlier.

Version No.	Variable	Description
1.x	ABE=ASCII	Save processing tables in ASCII format.
4.8	COMSPEC	MS-DOS standard variable for path to COMMAND.COM
4.1	DIALOGINVERSE=0xNN	Sets the foreground and background inverse colors of filePro dialog boxes.
4.1	DIALOGNORMAL=0xNN	Sets the foreground and background colors of filePro dialog boxes.
3.0	ERRORINVERSE=0xNN	Controls colors of error boxes.
3.0	ERRORNORMAL=0xNN	Controls colors of error boxes.
3.0	HELPINVERSE=0xNN	Sets the foreground and background inverse for help.
3.0	HELPNORMAL=0xNN	Sets the foreground and background normal for help.
3.x	INSTDRV	Part of MDCONFIG. Allows installation from B:
4.5	LOGAPPEND	Append LOGFILE instead of overwriting. Also PFLOGAPPEND.
4.5	LOGFILE	Sets filename for LOGTEXT command.
3.0	MENUBORDER	Sets foreground and background colors of borders.
3.0	MENUINVERSE	Sets inverse colors of menus.
3.0	MENUNORMAL	Set normal colors of menus.
4.5	PF64K=ON	Turns of size warning in cabe.
4.5	PFADDWP	Control .wp extension for Import/Export word.
4.8	PFAUTOKSIZE	Sets default tok size for auto processing. If not set, default value is "20000" prior to 5.6.0 and "100000" with 5.6.0 and later.
4.8	PFBACKGROUND	Turns off background processing.
4.8	PFBIXBLANK	Controls how filePro treats a null lookup key.
4.8	PFBIXBUILD=2	Use 4.1 style sorting for indexes.
4.8	PFBIXNODESIZE	Overrides index rebuild nodesize.
4.8.8	PFBLANKOV	Causes date math with blank dates to return "/OV".
4.5	PFBLDFREE	Freechain build message. Default is OFF.
4.5	PFBREAK=OLD	Processing tables continues when break key pressed.
4.5	FBRWM=ON	Strip trailing blanks from browse lookup.
4.8	PFBRWSLASH	Controls backslash handling for browse lookups.
6.01	PFCATCHSIGPIPE	ON changes the way SIGPIPE works with user commands. Default=OFF
4.5	PFCHECKLOCK	Warning if attempt to modify lookup w/o -p flag
5.0	PFCHECKLOCKPOPOP	Controls logging of non-protected lookups.
4.5	PFCLKBREAK	Return to last function when break key pressed.
4.8	PFCLOCK	Enables or disables clock displayed in menus.
5.0.15	PFCLOSEPENDWARNING=OFF	Disable the warning if you attempt to close an HTML tag when it was not open.
4.5	PFCMARK	Century mark.
4.8	PFCONFIG	Overrides default path of /fp/lib/config.
4.8	PFCURSOR	Sets the size of the cursor.
1.x	PFDATA	Drive letter for the "/filePro" directory.
5.0	PFDIALOGPROMPT	Controls positioning of system message prompts.
1.x	PFDIR	Path for the "/filePro" directory.
4.8	PFDIRFILTER	Verifies that only /filepo/directories in filename list.
4.1	PFDLDIR	Sets path to downloadable printer file.
4.5	PFDLGENTER	Enter key acts like a save key.
4.1	PFDROPSHADOW	Turns the drop shadow on or off.
3.0	PFDSK	Identifies data drives. Overrides PFIGN
4.8.9	PFDFAILBOX	Causes "edit failed" messages to appear in a popup box, rather than flash at the bottom of the screen.
5.6.2	PFENTSELDISABLE=list	Disables (and remove prompts for) a set of default behavior of *clerk at the "Enter Selection" prompt.
5.0	PFEOF	Sets End-of-Field character to use with filePro's internal memo editor.
5.0	PFEOP	Sets End-of-Paragraph character to use with filePro's internal memo editor.
5.0.14	PFEEXPORTALL=ON	Use to "revert back" to old behavior.
4.5	PFERRKEY	Key to return from a filePro/system error.

4.8	PFF6PROMPT	Controls F6 with @wbl.
4.5 (Obsolete)	PFFILES=mn	Override default of 20 file handles.
5.0.9	PFFIXEDLISTSIZE=ON	Prevents filePro from shrinking selection lists. This allows screen readers for the blind to be programmed with fixed screen locations for such lists. Default: OFF
5.0.9	PFFIXNOLOCK=OFF	Turns off a change in behavior related to how filePro handles posting to a lookup that does not have a "-p" to lock the record.
5.0.6	PFFORCECURSORPATH	Turn OFF forced cursor path for mouse clicks in fileProGl.
4.8	PFFORMTOKSIZE	Sets default tok size for "FORM" command. If not set, default value is "20000" prior to 5.6.0 and "100000" with 5.6.0 and later.
4.1	PFGLOB	Override path "/fp/lib/edits"(global edits).
4.1	PFHCF	Page eject after "H" for "hardcopy".
5.0	PFHLPAUTOGOTO	Forces F9 index search upon entering help.
5.0.5	PFHLPDIR	Sets alternate path to filePro help files.
1.x	PFIGN	Drives to ignore. (DOS only)
4.1	PFIMPBUF	Increase import record length. Default value increased from "1024" to "10000" in 5.6.0
5.0.9	PFINSERTMODE=ON	Set insert mode on by default in *cabe/*clerk.
4.8	PFXGT	Allows clerk to do next gt if no exact match.
3.0	PFXS	Turns on or off "Index Scan" feature.
4.8	PFKEPIXVAL	Keep last index used in clerk.
6.0.02	PFKEYLOGGER	ON to log all keystrokes
6.0.02	PFKEYLOGGERMB=n	Where n is the number of megabytes allowed
4.5	PFKEYTAB	Changes filePro key table as specified.
4.5	PFLABEL	Allow invalid characters in a prc tables.
4.5	PFLANG	Sets the sort collating sequence for different languages.
4.5	PFLBSIZE	filePro label table size. PFLBSIZE defaults to 1000, but can be set to any from 100 to 32,767.
5.0.15	PFLICDEBUG=path	Set path & filename for creating a license debug file.
ODBC 1.0.01 5.0.15	PFLICFILE	Override the default license path. Default path is %\pfprog%\fp\lib\licfp.dat
4.8	PFLISTSLASH	Controls backslash handling in list boxes.
4.5	PFLKNL	Lookup "-n" finds last match instead of 1st.
4.5	PFLOCKBOX	Flashes "record is being updated" message.
5.0.15	PFLMHOST:IP_addr:port	Identify where the license manager resides.
5.0.14	PFLOGAPPEND	Append LOGFILE instead of overwriting. Use LOGAPPEND prior to version 5.0.14
5.0.14	PFLONGVARDOT=OLD	Allow dots in declared variable names.
5.0	PFLOOKUPNOFILE	Allows you to check syntax on a prc table without it checking for valid filenames in any lookups.
5.0.6	PFLOOKWIZPROT	Change default for lookup wizard's "protect record".
4.8	PFLX	Globally disable creation of browse lookup.
4.5	PFMAXALLOC	Max # of sort buffers to allocate for indexing.
4.5	PFMAXASIZE	Max sort buffer size for indexing.
4.5	PFMAXTEMP	Maximum virtual memory size when sorting.
4.5	PFMAXTRIL	Maximum virtual memory files used for sorting.
3.x	PFMBTO	Sets "messagebox" timeout in seconds.
1.x	PFME	Waits for Return key press for next field.
5.0.10	PFMEMOINSERTMODE=ON	Sets default memo editor insert mode to on.
5.0.15	PFMEMOEDITOLDKEYS=ON	Revert to old keys T = Toggle Insert and I = Insert Time for the memo editor.
4.5	PFMENBRK=OLD	Restores 4.1 behavior for break in a menu.
4.1	PFMENU	Overrides default path for user menus.
4.8	PFMISSINGARG=OLD	Revert to old method for missing arguments.
4.1	PFMONO	Use monochrome screens with a color video card.
5.0.15	PFMSBBLINK=ON	Restore the old behavior of MSB meaning "blink".
4.5	PFMU	Turns off "protect lookup" in cabe lookup.
1.x	PFNAME	Same as using Set/Change filename from menu.
3.0	PFNB	Turn banner printing off.
3.0	PFNET	Use Network calls. (DOS only)
4.8	PFNEWNTCONSOLE	Create new console for Java RunMenu.
5.0.6	PFNEWSK	Allows new @sk values.

4.1	PFNOBOX	Eliminates the boxes around menus, headers etc.
4.5	PFNODF	Disables ddefine, dexpand free-disk space check.
4.1	PFNOHELP	Displays "No Help Available" if ON.
5.6.2	PFNODFMSG=OFF	Turns off ddefine's "FFNODF=ON" notice. (Default:ON)
5.0	PFNOXHIDE	Disables/enables Index hiding.
5.0	PFNOQUAL	Turns off "NONE" from qualifier list.
4.8	PFNOTRAP	Controls filePro trapping of SIGBUS and SIGBUSV. (filePro internal debug)
4.8.10	PFNTPRTRERR	Message box for NT internal errors.
5.0	PFNULLIXSORT	Enables index sort without a major key.
4.5	PFNUMIXBUILD	Number of index blocks cached.
4.5	PFNUMIXBUF	Number of index blocks to buffer.
5.0.14	PFODBCCOMMITTYPE	Selects the open-commit-type to use for high-level ODBC e.g. " 0 ", " 1 ", or " 3 " .
4.5	PFOLDIX	Builds old style(4.1) indexes.
5.0.14	PFOLDONCE=ON	PFOLDONCE=ON will "revert back" to a modified version of the old behavior.
4.5	PFONEHEAD	Report prints header lines only once.
4.1	PFOUTS	Specifies serial communication parameters.
5.0	PFPERL	Provides the path for PERL executable.
4.8	PFPOSTPRINT	Executes command line variable when printing to a file.
4.8	PFPRINTER	Set the printer type and destination (Enhanced to allow setting to LOCAL and SCREEN)
4.5	PFPRINTERx	Set printer characteristics for printer 1-9.
1.x	PFPROG	Path to the "/fp" directory.
3.0	PFPRT	Directs output to a device or filename.
4.1	PFPRTC	Set the printer type.
4.0	PFPT	Controls local printing (AIX/LINUX/UNIX).
4.0	PFPTO	Wait time in sec. for "Printer Ready".
3.0	PFQUAL	Qualified data set to use.
5.0	PFQUALMMSG	Override default qualifier message.
4.8	PFQUIT	Disables CTRL \ for UNIX users.
4.8	PFREADONLY	Forces read-only attribute.
5.0.6	PFREFRESHRATE	Sets the screen refresh rate during sort/select in dxmaint/*reporty to once every "nnn" seconds.
4.5	PFRETRY	Number of retries for locked read.
5.0.12	PFROOTFIX=OFF	Turns off fix for UID with root login on *NIX systems if not required.
4.8	PFSCC	Enables "lscc" shell-escape within clerk.
6.02	PFDFEFCOLOR	PFDFEFCOLOR=ON/OFF (default ON) to force monochrome in ddefine.
4.8.8	PFSEVR/ROOT	Implied root for HTML and JSFILE
3.0	PFSHADOWCOLOR	Foreground and background colors for drop shadows.
5.0	PFSHOWF6ARROW	Shows a down-arrow as EOF marker for F6 popups.
5.0.9	PFSHOWROWCOL=OFF	Turns off the row/column display in programs like dscreen, drpedef, and *cabe. It can confuse screen readers for the blind, as the numbers are read every time you press a key.
5.0.6	PFSHOWWINERROR	Show specific Windows error codes.
5.0	PFSEXHEX	Turns on display of HEX value for @SK.
4.8	PFSKIPLOCKED	Skip locked records nnn seconds.
4.8	PFKIPPEDLOG	Log records skipped by PFSKIPLOCKED.
4.5	PFSP	Overrides the site password stored in fppath.
5.6	PFSPELLPATH=path	Path for the spellchecker dictionary.
5.6	PFSPELLUSERLIST=path	Override default path for personal dictionaries.
3.0	PFSPPOOL	Selects spooler/printer attached to spooler.
4.5	PFSSYNC	Sync after expanding files or writes.
4.8	PFSSYSEJID	If OFF, then SYSTEM command will be executed without the filePro setuid. Default is ON
4.8	PFSSYSYR4	@BD, @CD, @TD, @UD to returns 4 digit years.
4.0	PFSTERM	Type of terminal being used.
4.0	PFTIMEOUT	Same as PFPTO. Default 10 seconds.
1.x	PFTMP	Identifies where to place temporary files.
4.1	PFTOKSIZE	Sets the default token table size. If not set, default value is "20000" prior to 5.6.0 and "100000" with 5.6.0 and later.

1.x	PFVER	Show individual filePro program version.
4.8	PFWGT0	Totals without selected records.
3.0	POPUPNORMAL	Color code for popup windows foreground.
3.0	POPUPINVERSE	Color code for popup windows background.
5.0	PFUFLAG=ON	Same as -U flag for reports. (jumpstart)
4.8.9	PFUMASK=0nnn	Sets Unix "umask" value.
4.5	PFXFERDOS=OLD	Use old syntax for "doscp" in SCO Unix.
4.8	PSI	Standard Unix variable for the shell command prompt.
1.x	TERM	Type of terminal being used.
1.x	TERMCAP	Filename overrides "/etc/termcap" file.
3.0	TEXTINVERSE	FG and BG colors of menus, prompts etc.
3.0	TEXTNORMAL	FG and BG colors of menus, prompts etc.
6.1	PFERRSUPPRESS PFPWAUDIT	Password auditing also requires a ./fp/logs/pwaudit.cfg file. Same structure as servlog.cfg. Any error that would be sent to mail will still be mailed on unix/linux based systems. Errors reported in the background will still be suppressed. Including the program name. Invalid password and license errors will still be reported. Password errors omit the filename. dcabe and rcabe are exempt from the error suppression.
6.1	PIGNTMEDS	Ignore "Too many edits" error message. Default OFF.
6.2	PFMENUVER=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.
6.2	PFXLASCII=OFF	If enabled, any non- printable ASCII characters will be automatically stripped from data when inserted into an XLSX document.

System Maintained Fields

This provides the filePro version number that added or modified the system maintained field. If you are using an older version than identified, the system maintained field will not work. Items with version number 3.x were included in version 3.1 or earlier.

Version No.	Field	Description
3.x	@AF	Associated field instance.
4.8	@B4	4 digit year equivalent of @BD
3.x	@BD	Last batch update. (MMDDYY)
4.0	@BK	Exit key for browse lookup.
3.x	@BR	The currently highlighted
6.1	@BT	Last batch update time. (HHMMSS)
4.8	@C4	4 digit year equivalent of @CD
3.x	@CB	The name of the user that first created the record.
3.x	@CD	The date that the current record was created.
5.0	@CO	The column of field left or entered.
4.8	@CP	Position of the cursor within the field you just left.
6.1	@CT	The time that the current record was created. (HHMMSS)
3.x	@DT	The current date, spelled out.
4.1	@FD	Field cursor is in.
4.8	@FI	Current file name
3.x	@FN	The current format name.
3.x	@ID	The name of the current user.
3.x	@LC	Line count.
5.0	@LI	The current line number within processing.
4.1	@OS	The operating system name ("DOS", "UNIX", "LINUX", etc.)
4.8	@PC	Current printer comment
4.8	@PD	Current printer destination
3.x	@PM	Parameter passed to ".prc" browse lookup row.
3.x	@PN	The current page number.
4.8	@PR	Current printer name
4.8	@PT	Current printer type
4.8	@PW	Parameter passed to "prc" with -rw flag.
4.8.5	@PX	Parameter passed to "prc" with -rx flag.
4.8.5	@PY	Parameter passed to "prc" with -ry flag.
4.8.5	@PZ	Parameter passed to "prc" with -rz flag.
4.8	@QU	Current qualifier
3.x	@RN	The current record number.
5.0	@RO	Row position of field.
4.8	@RP	Number of records processed.
3.x	@RS	Number of records selected so far.
3.x	@SF	The contents of the subtotal
3.x	@SH	The name of the subtotal
4.1	@SK	Special key (used with INKEY, when-processing only)
3.x	@SN	The current screen.
4.8	@T4	4-digit-year equivalent of @TD
3.x	@TD	The current date. (MMDDYY)
3.x	@TM	The current time. (HHMMSS)
3.x	@TS	The total number of records selected for this report.
3.x	@UB	The name of the user that last updated this record.
4.8	@U4	4 digit year equivalent of @UD
6.1	@UT	The time the current record was last updated. (HHMMSS)
3.x	@UD	The date the current record was last updated. (MMDDYY)
4.8.4	@VR	Version number of *clerk or *report being run.
5.8.3	@VR2	Version number of *clerk or *report being run for menu use

System Arrays

Version No.	Array Name	Description
5.0	@CMDLINE	Command line passed to *clerk or *report.
5.0	@DIRLIST	List of filenames returned by NEXTDIR.
5.0	@DIRLIST_EXT	Contains the extension as returned by NEXTDIR.
5.0	@DIRLIST_FILENAME	Contains the full filename as returned by NEXTDIR.
5.0	@DIRLIST_NAME	Contains the format name as returned by NEXTDIR.
5.0	@SYSFILES	All system files opened.
5.7	@SYSINFO	Returns information about the current time, in several formats.
5.0	@ALLFILES	All filePro files Open or Closed.
5.0	@FPFILES	Open filePro files.
5.0	@FSTAT	Information for the last successful EXISTS().
5.6.4	@FUZZY	Contains info about the most recent fuzzy browse lookup.
5.0.15	@LICENSE	Information related to the filePro license.
5.6	@WORDWRAP[]	Returns text for the most recent WRAPINFO call.

Trigger Fields

This provides the filePro version number that added or modified the trigger field. If you are using an older version than identified, the trigger field will either not work, or work differently than advertised. Items with version number 3.x were included in version 3.1 or earlier.

Version No.	Field	Description
4.8	@DONE	Executes processing after @WGT.
4.1	@ENTSEL	Processing tables to do after record is displayed, but before the prompts are displayed.
3.x	@KEYx	When user presses key "x".
4.1	@MENU	Processing to do before the clerk menu. (Inquire/Update/Add)
5.0	@ONCE	Executes once before records processed.
4.1	@UPDATE	Starts input prc before user is put into update mode.
3.x	@WFFnnn	When entering field "nnn". (Inquire, Update, Add)
3.x	@WGT	When breaking for a grand total. (Request Output)
4.1	@WBLnnn	Responds to browse lookup key (KB) when in field "nnn"; existing browse lookup is not performed.
3.x	@WBRKn	When breaking for subtotal "n". (Request Output)
4.1	@WHPnnn	When user presses help key (KJ) in field "nnn". (Inquire, Update, Add)
3.x	@WLFnnn	When leaving field "nnn". (Inquire, Update, Add). See also @WFF.
4.1	@WUKnnn	When user key (KE) is pressed in field "nnn".
4.5	@W??*	The "*" character can be used in place of the "nnn" field number in WFF, WLF, WBL, WHP, WUK

The filePro Plus software and the documentation provided with it are protected under United States Copyright Laws and is provided subject to the terms and conditions of the filePro License Agreement.

PLEASE NOTE the support and fax phone numbers listed in this readme file. Open new support incidents on our website.

WWW http://www.fpotech.com
Support support@fpotech.com
Sales sales@fpotech.com
Management filepro@fpotech.com

To submit bug reports

1. Login to your account portal on our website <http://www.fpotech.com/fpotech/login.php> and then go to the Support Incident Menu and submit an incident request.
2. EMail them to support@fpotech.com including the text "Bug Report" with the version # and your filePro License # in the subject line
3. FAX them to (813) 354-2722 clearly marking them as bug reports and be sure to reference your filePro License #
4. Call the customer support number (800) 847-4740

A special thank you to Jim Asman for his contribution to the functionality of our printer tables. Jim was a good friend to filePro and is dearly missed.

Contact Information

Surface Mail

fP Technologies, Inc.
432 W. Gypsy Lane Road
Bowling Green, OH 43402

Phones

Support (800) 847-4740
Sales (800) 847-4740
Fax (813) 354-2722

Email

Support support@fpotech.com
Sales sales@fpotech.com
Management filepro@fpotech.com

It's important that you clearly describe a suspected bug and include the filePro version number. If the programmer has trouble figuring out what you meant, you might as well not have reported the bug. Be very specific. For example, if you are reporting a bug concerning a Browse, identify if it is a lookup browse or browse created by using the [F6] key. A screen shot is very helpful and sometimes better than more than 1000 words.

Describe exactly how to duplicate the bug. Although it's sometimes difficult to create a working sample to demonstrate the problem, make every effort to trim down your code and provide a working sample application with test data. You may even discover that what you thought to be a bug is due to a coding error or the bug may only occur with lots of data or large processing tables.

Take good notes as to any error messages and under what circumstances the error message is presented. It never hurts to provide more information rather than not enough. This is particularly true when the programmer asks for additional information. Rather than responding with a single sentence, be verbose since this may shed some light on the bug or what you may be doing wrong in your code.

Read what you wrote. Closely read your bug report before submitting to make sure it's clear and complete. If you have listed steps for duplicating the bug in a sample, exercise the sample with the listed steps to make sure you haven't missed a step.

filePro and filePro Plus are registered
trademarks of fP Technologies, Inc.

=====
Bug fixes are below the New Items.
=====

Version 6.1.02.RR New USP Only Items

=====

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.

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.

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 :".

Added new option to ENCODE() and DECODE(), "URL", to handle URL percent encoding. Failure to decode will return an empty string.

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

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.

Added READMAP(file) function. 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:"

Added a new function 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.

Examples:
Given a print code table containing the following entries:
+-----+
+- Number -- Sequence ----- Description -----+
| 1 %2 %3 Initialize printer |
| 2 <page> New Page |
| 3 Set Font |
+-----+
if: ' x will contain '<page> '
then: x = PRINTCODE("1")
if: ' x will contain '<page> '
then: x = PRINTCODE("1","0")
if: ' x will contain 'New Page'
then: x = PRINTCODE("2","1")

Added x=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
```

Added 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 retrieve 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.

Added 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.

Added 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.

Added 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.

Added s = GETQUAL(fname) function. GETQUAL() will return 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 :"
```

Added n = GETQUAL(array, fname) function. GETQUAL() will return 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
```

Added new XLSX functions: XL_FREEZEPANE, XL_FREEZEPANE2, XL_SPLITPANE

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.

Added new environmental variable PFXLASCII, default OFF. If enabled, any non-printable ASCII characters will be automatically stripped from data when inserted into an XLSX document.

Enhanced DIM to allow IMPORT and EXPORT commands to 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 commands to support arrays. 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
```

Added `x = 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 "")
```

Added XML import and export code.

`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

Added LOOP commands.

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
```

=====
Version 6.1.01.RR New USP Only Items
=====

Added JSON import and export code.

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

```

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. 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.

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")

Added QRCODE FPML print code.

<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 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.

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.

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"

Added initial support for multi-dimensional arrays.

DIM array[n1,n2,...,n8] (1,e)
Multi-Dimensional array of fields with length "1" & 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"]

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 average 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.

=====
END OF NEW USP ITEMS
=====

6.1.XX.08 NEW ITEMS
=====

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.

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.

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.

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

=====
6.1.XX.07 NEW ITEMS
=====

Added a F7 last record option to clerk.

Added new system controlled fields for creation time (@CT), update time (@UT), and batch time (@BT) per record. Note: The time is stored in 2 second intervals.

=====
6.1.XX.06 NEW ITEMS
=====

Added a new option to show a stacktrace on a runtime error if PFERRTRACE is set. Default OFF.

Dxmaint will now always show qualifier if PFQUAL is set.

=====
6.1.XX.04 NEW ITEMS
=====

Added PFOLDCHAIN to allow CHAIN to return to the top of processing when a record is saved and the chain was performed inside of an event.

Added basic reconnect functionality into ODBC mirroring upon communications link failure.

Updated Fuzzy search screen in clerk to be larger and show correct button prompts.

```
n = STACKTRACE(array)  
Fill an array with a processing trace, listing the current and past processing tables and their line numbers to the current line being executed. This will show lines "jumped" from gosubs and follow calls and functions.
```

Returns the number of elements that could fit into the array.

Added new debugger option "T" to show the current stacktrace while debugging.

=====
6.1.00.03 NEW ITEMS
=====

Updated all programs to no longer require unixODBC by default. unixODBC will now only be required when an ODBC related function is used. If unixODBC is not found when an ODBC function is required, a filePro error will be returned.

Added the ability to assign directly to a longvar when creating it.
e.g.
declare myvariable(32,*)="Hello, World!"

Reworked tokenization engine to no longer require setting PFTOKSIZE or related

variables. Variable will now be silently ignored.

Added PFPDFAUTOBREAK=ON (default OFF) to allow PDFs to automatically break pages based off of selected paper type.

Added menu letter to menu script editor.

=====

6.1.00.00 NEW ITEMS

=====

You can now use: @wlf<letter>*
ex. @wlfT*
This will apply to any dummy/associated field that begins with 'T'
Overrides any other @wlf*

Added logging to ddefine.
ddefine can now optionally track changes made to filePro file layouts. This includes the name of the file, who changed it, and what fields were changed. Requires a logging configuration file to be added under the ./fp/logs directory named 'ddefine.cfg'.
Format of the config file is the same as the servlog.cfg file that comes shipped with filePro.
Example ddefine.cfg:
 ROLLING,DEBUG,ddefine.log,60000

xx=FORMERROR
syntax: xx=FORMERROR()
returns: errno from last FORM or FORMM command.
e.g. 2=file not found, 13=permission error

Validate menu script before prompting for removal

Added new option 'C' to F8 Extended Functions for dmoedef to show a list of all print codes on an output format. Selecting an item from the list will jump the editor to it.

TRIM command to remove spaces
aa=ltrim(fld)
 left trim
aa=rtrim(fld)
 right trim
aa=trim(fld)
 trim both left and right

PFIXGT can now be set in dxmaint F8 options.
This is backwards compatible, so if PFIXGT is still set in config, then it is honored by clerk *if true*. If false, the index header is checked for the flag.

Windows fPTransfer now will accept wildcards.

A compress-filePro file routine
fppack

Function:
Remove deleted records from a filePro file, and then (optionally) rebuild all automatic indexes.

Syntax:
fppack [filename | -] [-H heading] [-E] [-R] [-X] [-EX] [-C] [-M name | -MD | -MQ mesg | -MA] [-BG] [-BS]

- H "heading" custom title to display in box.
 - E don't actually pack the records, just give statistics.
 - R rebuild the automatic indexes even if no records were deleted.
 - EX skip statistics
 - C skip continue and finished prompts

 - X skip rebuilding the auto indexes.
 - M name qualifier file name to use.
 - MD ask for qualifier with default prompt.
 - MQ "mesg" ask for qualifier with "mesg" as the prompt.
 - MA use all qualified files & main file.
- UNIX/XENIX only:
- BG work in the background.
 - BS suppress "completed in background" message.

Added various enhancements to PDF engine.
See on-line or ~/fp/docs PDF documentation.

Added optional error message suppression and basic password auditing to filePro.

PFERRSUPPRESS=ON, default OFF
PFPWAUDIT=ON, default OFF

Password auditing also requires a ./fp/logs/pwaudit.cfg file. Same structure as servlog.cfg.
Any error that would be sent to mail will still be mailed on unix/linux based systems.
Errors reported in the background will still be suppressed.

Including the program name.
Invalid password and license errors will still be reported. Password errors omit the filename.
dcabe and rcabe are exempt from the error suppression.

These functions lock or unlock bytes of the file specified by handle.

```
x=lock(handle,how[,nbyte])
  handle - an open handle to a file
  how    - U|0 : unlock bytes
          L|1 : lock bytes
          N|2 : lock bytes non-blocking
  nbyte  - How many bytes in the file to lock, if omitted, lock
          the billionth byte in the file (file does not have to be
          that large)
```

```
x=unlock(handle[,nbyte])
  handle - an open handle to a file
  nbyte  - How many bytes in the file to unlock, if omitted,
          unlock the billionth byte in the file (file does not
          have to be that large)
```

(returns "1" on success and returns negated system error on error)

ddefine will now create new screens the same as dscreen does instead of just mono.

NEW command OPENDIR2 to handle long-named files and paths.

```
Syntax:
N = OPENDIR2(mask, path, fmt_sz, ext_sz, nam_sz)
All arguments are optional.
  Format Length
  Extension Length
  Fullname Length
```

*cabe lookup wizard will now honor PFQUAL and show qualified indexes

Added new FPML commands to control the appearance of underlines. (See PDF Docs)

New RINSTR, and INSTR now allows negative positions for working backwards.

New GIadmin that will count GUI (GI or Web) sessions, ease of system and user configuration files and additional security.

Added PDF syntax as an option for printer maintenance (pmaint): Windows only

Lookup Wizard in cabe now allows long vars as key.

Added alias and arrays to F6-D-L display in *cabe.

Updated color with new routines and corrected the shell escape codes.

Automated processing table backups.

```
CABEBACKUP ON|OFF (on by default)
CABEBACKUPMINS n (minutes between backups)
CABEBACKUPCT n (backup files per process)
```

Menu maintenance (makemenu) now asks if you wish to remove an unused menu script if the menu item is not used.

*report now allows one to use .outs from a pathed directory library

SCREEN command can switch fields in a POPUP UPDATE -, provided no screen name is passed to the SCREEN command.

MEMO EDIT now accept maxsize to limit the number of characters that can be entered into a memo field.

```
memo NNN edit (row,col,lines,width,startLine,startcol,maxSize)
(Text mode only)
```

Added option 7 to dxmaint to clear qualifier

New -SE *report flag to allow report to edit/save a selection set.

Added @EXIT label to *clerk processing. This is executed whenever a record is exited or broken out of. Events that trigger this are 'X' while not in update mode, 'BRKY' while not in update mode, and 'exit' in processing. It is the opposite of @entsel, and is the last thing executed when leaving a record. Assignment of real fields is not allowed, this is similar to @once in that the processing that is executed is NOT sitting on a record, but rather record '0'.

Partial lookup flag added to *cabe lookup wizard.

-O on an exact lookup now does partial key matching. This kills a lookup once the beginning of the key value no longer matches the lookup key value.

BUSYBOX

```
BUSYBOX "my message"
BUSYBOX("10","10")
BUSYBOX("10","10") "my message"
```

Added PFFPPFULLPATH as an enhancement to PDFPOSTPRINT
and added an PFNEWPOSTPRINT alias to name to PDFPOSTPRINT
Added PFFPPFULLPATH to augment the filename passed to the post print
handler, default ON, this causes the filename passed to the postprint
script to contain the full path to the file, not just the file name.
Set to OFF to revert to old behaviour. PFFPOSTPRINTnnn will now work
with normal file destinations. Same rules as the old global PFFPOSTPRINT
but also supports PDF files.

Clerk will now allow a full path to a form when using
the FORM and FORMM command in processing.

User defined functions

Forward declare functions to be used:
(function|func) [file.]name([dim|var] var1, [dim|var] var2, ...)

e.g.

```
function fplib.showlock(var pid)
function fplib.log(file, line, what)
function somefunc(dim myarray)
```

Call a function:

```
[x=][file.]name(var1, var2, ...)
```

Return a value from a function:

```
return(value)
```

Can pass fields: real, dummy, longvar

Can pass arrays: Alias and system arrays are copied to a non-aliased
array. Non-aliased arrays are passed by reference.

Function names must be at least 3 characters in length.

Functions cannot modify values outside of its scope.

Functions do not call automatic processing.

Functions cannot modify real fields.

Functions cannot be called unless they are declared.

Functions can pass values by reference (changes made to the value will
carry back out of the function, only to arrays).

Functions can optionally return a value.

Parameter names must be at least 3 characters in length.

Parameters will be passed to the function using the name they were defined
with in the declaration statement.

Environment variables:

PFFUNCDBG=(ON|OFF), default OFF.

If ON the debugger will be allowed to continue into the function
call. If OFF the debugger will skip over user defined functions.

NOTE: Debug statements inside of functions will still be able to
be activated. If debug is set inside of a function, it will
continue even after the function is left.

Example:

Processing table for fibonacci:

```
If:                ' Declare for future use
Then: function fibonacci(nval)
  If:                ' Get the parameter
Then: declare extern nval
  If: nval le "1"    ' Return the result
Then: return(nval)
  If:                ' Return the result
Then: return(fibonacci(nval-"1")+fibonacci(nval-"2"))
```

Usage:

```
If:                ' Declare for future use
Then: function fibonacci(nval)
If:                ' Call the function
Then: n=fibonacci("9")
If:                ' Display the result
Then: msgbox ""{n    ' Prints "34"
```

EXTERN and GLOBAL arrays

DIM GLOBAL name(size)

DIM EXTERN name

Only non-aliased arrays can be declared GLOBAL/EXTERN.

Functions similar to GLOBAL/EXTERN longvars.

New compare condition for Associated Fields

Added new selection set relational operators:

```
AEQ - Associated field, all equal
ANE - Associated field, all not equal
ACO - Associated field, all contain
```

These require ALL components of an associated field to match the
comparison being done, rather than just one of its component fields.

New functions for creating XLSX documents from filePro.

e = XL_OPEN(file [, name])

Start building an XLSX output file.

Parameters -

file : Path to the file to create. If no full path is given the
generated file will be placed in the PFTMP or equivalent
directory.

name : The name for the default sheet that will be created. Defaults to Sheet1.

If the filename does not end in ".xlsx" it will be added on creation.

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

Note: Only one XLSX file can be created at a time.

```
e = XL_SAVE([password])  
Save the current XLSX file.
```

Parameters -
password : If specified, encrypt the XLSX output file using Agile encryption (AES128).

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

Note: Encrypted XLSX files cannot be opened with most third party programs such as LibreOffice and OpenOffice. They are fully supported by Excel however. The documents are saved in an encrypted CFB file.

```
handle = XL_ADDSHEET([name])  
Add a new sheet to the XLSX document.
```

Parameters -
name : The name for the sheet to be created. Defaults to auto naming the sheet based on the Sheet1, Sheet2, ..., SheetN template.

Returns a handle to a new sheet object on success and "-1" on error. XL_ERROR() can be called to return the last error.

```
e = XL_ADDCELL([data [, style [, sheet [, row [, col]]]])  
Add a new cell to the XLSX document.
```

Parameters -
data : Data to be inserted into the document. A cell starting with '=' will be treated as a formula.
style : Handle to style to be used for this cell. Use blank to use the default style.
sheet : Handle to sheet to insert the cell on. Use blank, "0", or "-1" to use the default sheet.
row : Row to place the cell (0 indexed).
col : Column to place the cell (0 indexed).

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

Note: Using an empty or negative row/column value will cause the cell to be added using the auto counter in the sheet, incrementing the column value after the cell is added. Specifying a location will reposition the auto counter. Formulas can be used as part of the data as well by prefixing the string with '='.

```
e = XL_ADDCELL2([data [, style [, sheet [, cell]]])  
Add a new cell to the XLSX document.
```

Parameters -
data : Data to be inserted into the document. A cell starting with '=' will be treated as a formula.
style : Handle to style to be used for this cell. Use blank to use the default style.
sheet : Handle to sheet to insert the cell on. Use blank, "0", or "-1" to use the default sheet.
cell : The Excel style cell to insert the cell. e.g. "A1" "D6" "F6".

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

Note: Using an empty cell number will cause the cell to be added using the auto counter in the sheet, incrementing the column value after the cell is added. Specifying a location will reposition the auto counter. Formulas can be used as part of the data as well by prefixing the string with '='.

```
handle = XL_FORMAT(format)  
Create a new format to use with the XLSX document.
```

Parameters -
format : Excel format string to use to format the a style. e.g.
"\$ #,###,nnn.nn"
"% ##n.n"
"m/d/yyyy"

Returns a handle to a new format object on success and "-1" on error. XL_ERROR() can be called to return the last error.

e = XL_COLWIDTH(width, firstcol, lastcol [, sheet])
Change the default column width for a sheet between a range.

Parameters -

width : Width of the column(s). e.g. "24" "12.5", "11"
firstcol : Zero based column index or column letter to set from.
lastcol : Zero based column index or column letter to set to.
sheet : Handle to sheet to change the cell widths.

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

handle = XL_FONT(font, [size [, attr [, color]])
Create a new font to use with the XLSX document.

Parameters -

font : Name of the font to use.
size : Point size of the font. e.g. "11" "8.42" "12", default "11.0"
attr : List of attributes to apply to this font, separated by commas.
e.g. "bold,italic"

Values:

- "bold"
- "italic"
- "underline"
- "strike"
- "unlocked"
- "hidden"
- "wrap"
- "shrink"
- "fill"
- "left"
- "center"
- "right"
- "justify"
- "top"
- "bottom"
- "vjustify"
- "vcenter"

color : The RGB Hex value to set the font color.
e.g. "000000" "ADD8E6"

Returns a handle to a new font object on success and "-1" on error.
XL_ERROR() can be called to return the last error.

handle = XL_BORDER(borderstyle [, color])
Create a new border to use with the XLSX document.

Parameters -

borderstyle : The style to use with this border. Must be one of the following values:
"thin"
"medium"
"dashed"
"dotted"
"thick"
"hair"
"medium_dashed"
"dash_dot"
"medium_dash_dot"
"dash_dot_dot"
"medium_dash_dot_dot"
"slant_dash_dot"

color : The RGB Hex value to set the border color.
e.g. "000000" "ADD8E6"

Returns a handle to a new border object on success and "-1" on error.
XL_ERROR() can be called to return the last error.

handle = XL_FILL(bg [, fg [, fill]])
Create a new fill to use with the XLSX document.

Parameters -

bg : The RGB Hex value to set the background fill color.
e.g. "000000" "ADD8E6"
fg : The RGB Hex value to set the foreground fill color.
e.g. "000000" "ADD8E6"

fill : The fill pattern to use, defaults to "solid" fill. Value must be one of the following.
"solid"
"medium_gray"
"dark_gray"
"light_gray"
"dark_horizontal"
"dark_vertical"
"dark_down"
"dark_up"
"dark_grid"
"dark_trellis"

```
"light_horizontal"  
"light_vertical"  
"light_down"  
"light_up"  
"light_grid"  
"light_trellis"  
"gray_125"  
"gray_0625"
```

Returns a handle to a new fill object on success and "-1" on error.
XL_ERROR() can be called to return the last error.

```
e = XL_ADD_DT(date, time [, style [, sheet [, row [, col]]]])  
Combine two fields into a single spreadsheet datetime field and insert it as  
a new cell in the XLSX document.
```

Parameters -

```
date : filePro date field.  
time : filePro time field.  
style : Handle to style to be used for this cell. Use blank to use the  
default style.  
sheet : Handle to sheet to insert the cell on. Use blank, "0", or "-1"  
to use the default sheet.  
row : Row to place the cell (0 indexed).  
col : Column to place the cell (0 indexed).
```

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

```
e = XL_ADD_DT2(date, time [, style [, sheet [, cell]]])  
Combine two fields into a single spreadsheet datetime field and insert it as  
a new cell in the XLSX document.
```

Parameters -

```
date : filePro date field.  
time : filePro time field.  
style : Handle to style to be used for this cell. Use blank to use the  
default style.  
sheet : Handle to sheet to insert the cell on. Use blank, "0", or "-1"  
to use the default sheet.  
cell : The Excel style cell to insert the cell. e.g. "A1" "D6" "F6".
```

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

```
handle = XL_CHART(type [, title [, xname [, yname [, row [, col [, stylenum  
[, sheet [, xoff [, yoff [, xscale [, yscale]]]]]]]]]]])  
Add a new chart to the XLSX document.
```

Parameters -

```
type : Type of chart to create. Must be one of the following values.  
"area"  
"area_stacked"  
"area_stacked_percent"  
"bar"  
"bar_stacked"  
"bar_stacked_percent"  
"column"  
"column_stacked"  
"column_stacked_percent"  
"doughnut"  
"line"  
"line_stacked"  
"line_stacked_percent"  
"pie"  
"scatter"  
"scatter_straight"  
"scatter_stright_markers"  
"scatter_smooth"  
"scatter_smooth_markers"  
"radar"  
"radar_with_markers"  
"radar_filled"  
title : The title for this chart.  
xname : The title for the x-axis.  
yname : The title for the y-axis.  
row : Row to place the cell (0 indexed).  
col : Column to place the cell (0 indexed).  
stylenum : Number of the built in Excel style to use. Must be between  
"1" and "48". The default style is 2. The value is one of  
the 48 built-in styles available on the "Design" tab in  
Excel 2007.  
sheet : Handle to sheet to insert the chart on. Use blank, "0", or  
"-1" to use the default sheet.  
xoff : X axis offset to place the chart, in pixels.  
yoff : Y axis offset to place the chart, in pixels.  
xscale : Scale the chart along the x axis. e.g. "1", "0.5" "2". Value  
cannot be negative.  
yscale : Scale the chart along the x axis. e.g. "1", "0.5" "2". Value  
cannot be negative.
```

Returns a handle to a new chart object on success and "-1" on error.
XL_ERROR() can be called to return the last error.

Note: The chart functions do not use the auto counter found in the sheets and instead will default to "0", "0" or "A1" when used for insertion.

```
handle = XL_CHART2(type [, title [, xname [, yname [, cell [, stylenum [, sheet  
[, xoff [, yoff [, xscale [, yscale]]]]]]]]])
```

Add a new chart to the XLSX document.

Parameters -

type : Type of chart to create. Must be one of the following values.
"area"
"area_stacked"
"area_stacked_percent"
"bar"
"bar_stacked"
"bar_stacked_percent"
"column"
"column_stacked"
"column_stacked_percent"
"doughnut"
"line"
"line_stacked"
"line_stacked_percent"
"pie"
"scatter"
"scatter_straight"
"scatter_stright_markers"
"scatter_smooth"
"scatter_smooth_markers"
"radar"
"radar_with_markers"
"radar_filled"

title : The title for this chart.
xname : The title for the x-axis.
yname : The title for the y-axis.
cell : The Excel style cell to insert the chart. e.g. "A1" "D6" "F6".
stylenum : Number of the built in Excel style to use. Must be between "1" and "48". The default style is 2. The value is one of the 48 built-in styles available on the "Design" tab in Excel 2007.
sheet : Handle to sheet to insert the chart on. Use blank, "0", or "-1" to use the default sheet.
xoff : X axis offset to place the chart, in pixels.
yoff : Y axis offset to place the chart, in pixels.
xscale : Scale the chart along the x axis. e.g. "1", "0.5" "2". Value cannot be negative.
yscale : Scale the chart along the x axis. e.g. "1", "0.5" "2". Value cannot be negative.

Returns a handle to a new chart object on success and "-1" on error.
XL_ERROR() can be called to return the last error.

Note: The chart functions do not use the auto counter found in the sheets and instead will default to "0", "0" or "A1" when used for insertion.

```
handle = XL_CHARTSHEET(type [, title [, xname [, yname [, stylenum]]])
```

Add a new chartsheet to the XLSX document. A chartsheet is a full chart that occupies it's own sheet and cannot contain any cells.

Parameters -

type : Type of chart to create. Must be one of the following values.
"area"
"area_stacked"
"area_stacked_percent"
"bar"
"bar_stacked"
"bar_stacked_percent"
"column"
"column_stacked"
"column_stacked_percent"
"doughnut"
"line"
"line_stacked"
"line_stacked_percent"
"pie"
"scatter"
"scatter_straight"
"scatter_stright_markers"
"scatter_smooth"
"scatter_smooth_markers"
"radar"
"radar_with_markers"
"radar_filled"

title : The title for this chart.
xname : The title for the x-axis.
yname : The title for the y-axis.
stylenum : Number of the built in Excel style to use. Must be between "1" and "48". The default style is 2. The value is one of

the 48 built-in styles available on the "Design" tab in Excel 2007.

Returns a handle to a new chartsheet object on success and "-1" on error. XL_ERROR() can be called to return the last error.

```
e = XL_SERIES(chartnum, sheet, namerow, namecol, cfirstrow, cfirstcol, clastrow,
             clastcol, vfirstrow, vfirstcol, vlastrow, vlastcol)
Add a series to a chart or chartsheet.
```

Parameters -

- chartnum : Handle to a chart or chartsheet to add series.
- sheet : Handle to sheet to get values from. Use blank, "0", or "-1" to use the default sheet.
- namerow : Series name row (0 indexed).
- namecol : Series name column (0 indexed).
- cfirstrow : Categories first row (0 indexed).
- cfirstcol : Categories first column (0 indexed).
- clastrow : Categories last row (0 indexed).
- clastcol : Categories last column (0 indexed).
- vfirstrow : Values first row (0 indexed).
- vfirstcol : Values first column (0 indexed).
- vlastrow : Values last row (0 indexed).
- vlastcol : Values last column (0 indexed).

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

```
e = XL_SERIES2(chartnum, sheet, namecell, cfirst, clast, vfirst, vlast)
Add a series to a chart or chartsheet.
```

Parameters -

- chartnum : Handle to a chart or chartsheet to add series.
- sheet : Handle to sheet to get values from. Use blank, "0", or "-1" to use the default sheet.
- namecell : Series name Excel style cell. e.g. "A1" "D6" "F6".
- cfirst : Categories first Excel style cell. e.g. "A1" "D6" "F6".
- clast : Categories last Excel style cell. e.g. "A1" "D6" "F6".
- vfirst : Values first Excel style cell. e.g. "A1" "D6" "F6".
- vlast : Values last Excel style cell. e.g. "A1" "D6" "F6".

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

```
e = XL_PROTECTSHEET(sheet, password)
Add a password to restrict editing of a sheet.
```

Parameters -

- sheet : Handle to sheet to protect. Use blank, "0", or "-1" to use the default sheet.
- password : Password to use to protect this sheet.

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

```
e = XL_PROTECTCHARTSHEET(cs, password)
Add a password to restrict editing of a chartsheet.
```

Parameters -

- cs : Handle to chartsheet protect.
- password : Password to use to protect this sheet.

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

```
e = XL_ERROR()
Return the last error generated by the XLSX set of functions.
```

Returns the last error string generated by the XLSX engine.

```
e = XL_SETPOS(row [, col [, sheet]])
Set the auto counter position for a sheet.
```

Parameters -

- row : Row to move auto counter to (0 indexed).
- col : Column to move auto counter to (0 indexed).
- sheet : Handle of sheet to set. Use blank, "0", or "-1" to use the default sheet.

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

```
e = XL_SETPOS2(cell [, sheet])
Set the auto counter position for a sheet.
```

Parameters -

cell : Excel style cell to set the auto counter to. e.g. "A1" "D6".
sheet : Handle of sheet to set. Use blank, "0", or "-1" to use the
default sheet.

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

```
e = XL_NEXTRROW([sheet])
```

Move the auto counter down a row for a sheet.

Parameters -
sheet : Handle of sheet to set. Use blank, "0", or "-1" to use the
default sheet.

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

```
e = XL_NEXTCOL([sheet])
```

Move the auto counter one column for a sheet.

Parameters -
sheet : Handle of sheet to set. Use blank, "0", or "-1" to use the
default sheet.

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

```
handle = XL_STYLE([font [, fill [, fmt [, btop [, bbot [, bleft  
[, bright]]]]]])
```

Add a new style to the XLSX document.

Parameters -
font : Handle to font object to use.
fill : Handle to fill object to use.
fmt : Handle to format object to use.
btop : Handle to border object to use for top border.
bbot : Handle to border object to use for bottom border.
bleft : Handle to border object to use for left border.
bright : Handle to border object to use for right border.

Returns a handle to a new style object on success and "-1" on error.
XL_ERROR() can be called to return the last error.

```
e = XL_IMAGE(img [, row [, col [, sheet [, xoff [, yoff [, scalex [, scaley  
[, flag]]]]]]]])
```

Add a new image to the XLSX document.

Parameters -
img : Path to image file to use.
row : Row to insert the image on (0 indexed).
col : Column to insert the image on (0 indexed).
sheet : Handle of sheet to insert image. Use blank, "0", or "-1" to use
the default sheet.
xoff : X-axis offset for the image, in pixels.
yoff : Y-axis offset for the image, in pixels.
scalex : Scale the image along the x-axis. e.g. "1", "0.5" "2". Value
cannot be negative.
scaley : Scale the image along the y-axis. e.g. "1", "0.5" "2". Value
cannot be negative.
flag : Option of how to position image.
"0" - Default positioning.
"1" - Move and size image with the cells.
"2" - Move but don't size image with the cells.
"3" - Don't move or size the image with the cells.
"4" - Same as "1" but wait to apply hidden cells.

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

Note: The image functions only support PNG, JPEG, and BMP files.

```
e = XL_IMAGE2(img [, cell [, sheet [, xoff [, yoff [, scalex [, scaley  
[, flag]]]]]]]);
```

Add a new image to the XLSX document.

Parameters -
img : Path to image file to use.
cell : Excel style cell to insert the image. e.g. "A1" "D6" "F6".
sheet : Handle of sheet to insert image. Use blank, "0", or "-1" to use
the default sheet.
xoff : X-axis offset for the image, in pixels.
yoff : Y-axis offset for the image, in pixels.
scalex : Scale the image along the x-axis. e.g. "1", "0.5" "2". Value
cannot be negative.
scaley : Scale the image along the y-axis. e.g. "1", "0.5" "2". Value
cannot be negative.
flag : Option of how to position image.
"0" - Default positioning.

"1" - Move and size image with the cells.
"2" - Move but don't size image with the cells.
"3" - Don't move or size the image with the cells.
"4" - Same as "1" but wait to apply hidden cells.

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

Note: The image functions only support PNG, JPEG, and BMP files.

e = XL_LASTCMD()
Get debug information about the last XLSX call.

Returns the last evaluated command parse string.

e = XL_MARGINS([left, [right, [top, [bottom, [sheet]]]])
Set the worksheet print margins.

Parameters -

left : Left margin in inches, e.g. "0.5", "1", "0.75". A blank or negative value will use the default of "0.7".
right : Right margin in inches, e.g. "0.5", "1", "0.75". A blank or negative value will use the default of "0.7".
top : Top margin in inches, e.g. "0.5", "1", "0.75". A blank or negative value will use the default of "0.75".
bottom : Bottom margin in inches, e.g. "0.5", "1", "0.75". A blank or negative value will use the default of "0.75".
sheet : Handle of sheet to set the margins. Use blank, "0", or "-1" to use the default sheet.

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

e = XL_LANDSCAPE([sheet])
Set the worksheet to print in landscape mode.

Parameters -

sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use the default sheet.

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

e = XL_PORTRAIT([sheet])
Set the worksheet to print in portrait mode.

Parameters -

sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use the default sheet.

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

e = XL_GRIDLINES(option [, sheet])
Set if the worksheet should display gridlines when printed.

Parameters -

option : Which Gridlines to print. Cannot be blank. Must be one of the following values.
"hide_all"
"show_all"
"show_screen"
"show_print"
sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use the default sheet.

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

e = XL_FITPAGES([height, [width, [sheet]])
Fit the printed area to a specific number of pages both vertically and horizontally.

Parameters -

height : Number of pages vertically. A value of "0" or blank will set the height as necessary.
width : Number of pages horizontally. A value of "0" or blank will set the height as necessary.
sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use the default sheet.

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

e = XL_PAPERTYPE(type [, sheet])
Set the paper format for the printed output of a worksheet.

Parameters -

type : The paper format to use with a printed worksheet. Must be one of the following values.

"default"
"letter"
"tabloid"
"ledger"
"legal"
"statement"
"executive"
"a3"
"a4"
"a5"
"b4"
"b5"
"folio"
"quarto"
"10x14"
"11x17"
"note"
"envelope"
"envelope_9"
"envelope_10"
"envelope_11"
"envelope_12"
"envelope_14"
"c"
"d"
"e"
"envelope_d1"
"envelope_c3"
"envelope_c4"
"envelope_c5"
"envelope_c6"
"envelope_c65"
"envelope_b4"
"envelope_b5"
"envelope_b6"
"monarch"
"fanfold"
"german_std_fanfold"
"german_legal_fanfold"

sheet : Handle of sheet to change type. Use blank, "0", or "-1" to use the default sheet.

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

e = XL_CENTERH([sheet])

Center the worksheet data horizontally between the margins on the printed page.

Parameters -

sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use the default sheet.

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

e = XL_CENTERV([sheet])

Center the worksheet data vertically between the margins on the printed page.

Parameters -

sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use the default sheet.

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

e = XL_PRINTACROSS([sheet])

Change the default print direction to across then down.

Parameters -

sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use the default sheet.

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

e = XL_SETHEADER(string [, margin, [limage, [cimage, [rimage, [sheet]]]])

Set the printed page header.

e = XL_SETFOOTER(string [, margin, [limage, [cimage, [rimage, [sheet]]]])

Set the printed page footer.

Parameters -

string : The header/footer definition string. See below for format options. Cannot be blank.

margin : The margin in inches to use for the header/footer. A blank, "0", or negative value will use the default margin of "0.3".

limage : Full path to an image to use in place of the left image placeholder.

cimage : Full path to an image to use in place of the center image placeholder.

rimage : Full path to an image to use in place of the right image placeholder.

sheet : Handle of sheet to set header/footer. Use blank, "0", or "-1" to use the default sheet.

Format Options -

Control	Category	Description
&L	Justification	Left
&C		Center
&R		Right
&P	Information	Page number
&N		Total number of pages
&D		Date
&T		Time
&F		File name
&A		Worksheet name
&Z		Workbook path
&fontsize	Font	Font size
&"font,style"		Font name and style
&U		Single underline
&E		Double underline
&S		Strikethrough
&X		Superscript
&Y		Subscript
&[Picture]	Images	Image placeholder
&G		Same as &[Picture]
&&	Miscellaneous	Literal ampersand &

Text in headers and footers can be justified to the left, center and right by prefixing the text with the control characters &L, &C and &R. For example, "&LHello, World!", "&CHello, World!", "&RHello, World!"

For simple text, if the justification is not specified the text will be center aligned. However, you must prefix the text with &C if you use any other formatting.

You are limited to 3 images in a header/footer.

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

Note: The image types supported are PNG, JPEG, and BMP files. There is a hard limit of 255 characters in a header/footer string, including control characters. Strings longer than this will not be written to the document.

e = XL_SETBACKGROUND(image [, sheet])
Set the background image for a worksheet.

Parameters -
image : Full path to an image to use as the sheet background.
sheet : Handle of sheet to set background image. Use blank, "0", or "-1" to use the default sheet.

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

Note: The image types supported are PNG, JPEG, and BMP files.

e = XL_HIDEZEROS([sheet])
Hide zero values in worksheet cells.

Parameters -
sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use the default sheet.

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

e = XL_SHOWROWCOL([sheet])
Show row and column headers on the printed page.

Parameters -
sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use the default sheet.

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

Rebuild All Indexes on a file. item '8' on the dialog. Note: this is in the "extended" dialog which shows when a filename is not specified from the command line. Indexes can be selected individually, or all (with F7). Press SAVE, and rebuild begins

Ability to SPLIT data into array

Usage:
sz=SPLIT(array, string, delimiter)
array is the array that the data will be placed into
string is the data to split
delimiter is the sequence of characters to split on

NOTE: The array being used must have the size defined for its elements and cannot be an alias.

Added the ability to show record locks from *clerk. Can also be used to terminate sessions directly. New option !L added to *clerk. Using !L will activate the new locked records list. Enter on a selected entry will give additional options to the user, including the ability to Kill or Terminate a locked process without having to go to the command line. Note: This option is only available on Unix/Linux/BSD

Added UID mapping to filePro, ddir/dprodir option F5.

This allows for UIDs (User IDs) to be aliased to specific usernames. In the event that a login account is removed from your system, this can be used to maintain the link between the removed login's UID and those stored in filePro, effectively allowing system variables such as @CB and @UB to be maintained.

Windows Only:

This also has the added benefit of allowing @CB and @UB to function on Windows by linking a "pseudo" UID to a given username. These UIDs are automatically generated but can also be manually added. When a user opens filePro and their username does not exist in the UID map file, a UID will be generated for that user. filePro will find the next available UID in the list, starting from 2000, and assign it to that username.

On all platforms, UIDs stored in this program must be unique and in the range 0-65535. Usernames can be duplicated on Unix and Linux platforms, but must be unique on Windows.

Usernames are case-sensitive on Unix and Linux platforms and are case-insensitive on Windows platforms.

Environmental Variables:

PFUIDMAP = /path
Alternate filePro UID map file. (Use full path)
Note: Must be set in the environment.

PFUSEUIDMAP = ON
Allows filePro to do UID mapping. Also expands the maximum username length returned by @CB, @UB, and @ID to 32.
Default: ON

String Functions

All "is" functions return "1" for true and "0" for false.

x=isalpha(fld [, pos])
Is the character at the position given a letter?

x=isdigit(fld [, pos])
Is the character at the position given a number?

x=isalnum(fld [, pos])
Is the character at the position given a letter or number?

x=isspace(fld [, pos])
Is the character at the position given a whitespace character?
' ', '\t', '\n', '\r', '\v', '\f'

x=islower(fld [, pos])
Is the character at the position given lowercase?

x=isupper(fld [, pos])
Is the character at the position given uppercase?

x=isxdigit(fld [, pos])
Is the character at the position given a hexadecimal character?
'0'-'9', 'A'-'F'

x=iscntrl(fld [, pos])
Is the character at the position given a control character?
ASCII codes 0x00 (nul) - 0x1f (US), and 0x7f (del)

x=isprint(fld [, pos])
Is the character at the position given a printable character?
ASCII codes greater than 0x1f (US) not including 0x7f (del)

x=ispunct(fld [, pos])

Is the character at the position given a punctuation character?

`x=isgraph(fld [, pos])`

Is the character at the position given a character with a graphical representation? The characters with graphical representation are all those characters than can be printed (as determined by `isprint`) except for space.

`x=tolower(fld [, pos])`

Return the character at the position given as a lowercase character.

`x=toupper(fld [, pos])`

Return the character at the position given as an uppercase character.

`str=strtolower(fld)`

Return the entire string converted to lowercase.

`str=strtoupper(fld)`

Return the entire string converted to uppercase.

Added new array size function to get the size of an array. Can be used with GLOBAL, EXTERN, LOCAL, and SYSTEM arrays.

`x=ARRAYSIZE(array)`

Where array is the name of the array.
Where x is the returned size of the passed array.

Added new DECLARED function to check if an array or longvar is defined, meaning it is either declared LOCAL or GLOBAL or is declared EXTERN but has a matching GLOBAL definition.

`x=DECLARED(var)`

Where var is either a longvar or an array.
Where x is the return value.
Returns 0 if the variable is not fully defined.
Returns 1 if the variable is fully defined.

Increased ACTION length in debugger from 60 characters to full 128.
Should now be the same as *cabe.

Added new flag -DM to [dr]clerk to disable the Index Mode prompt from @ENTSEL. Only works when not in update mode.

Added flag -RH to report to disable the automatic record number reporting in the middle of the screen. This enables placing text on the center of the screen without it being overwritten when the display updates.

`x=@GUI.PAUSE()`

Pauses automatic screen updating while in GI/Web.

`x=@GUI.RESUME()`

Resumes automatic screen updating while in GI/Web.

REPLACE() enhancement - allow null characters
Enhanced REPLACE() to accept null characters

FORM WITHPROC

FORM WITHPROC "formname"
FORMM WITHPROC "formname"

Added additional command switch to FORM and FORMM commands to allow the associated processing table to run while in input processing.

Note: You cannot call the WITHPROC variant from within another form UNLESS the calling form is a processing only form.

Addqual Program

Addqual allows you to easily add qualifiers to your files either interactively or through the command line.

This runs interactively:
addqual [filename]

This runs automatically:
addqual filename -q <qualname>
as does this:
addqual filename -q <qualname> -x <qual-to-copy-from>

The automatic commands will display graphics on errors. You can keep graphics off with "-s" and errors will be printed on the command line if they occur.
example:
addqual filename -q <qualname> -s

List of switches:

-q qualifier to create
-x qualifier to copy indexes from
-s silent, no graphics
-h --help syntax help

XFER - encrypted transfers server-peer

CABE F6 list files from F8 L-Load

```
=====  
End End End End End End End End End  
=====
```

The filePro Plus software and the documentation provided with it are protected under United States Copyright Laws and is provided subject to the terms and conditions of the filePro License Agreement.

PLEASE NOTE the support and fax phone numbers listed in this readme file. Open new support incidents on our website.

WWW http://www.fpotech.com
Support support@fpotech.com
Sales sales@fpotech.com
Management filepro@fpotech.com

To submit bug reports

1. Login to your account portal on our website <http://www.fpotech.com/fpotech/login.php> and then go to the Support Incident Menu and submit an incident request.
2. EMail them to support@fpotech.com including the text "Bug Report" with the version # and your filePro License # in the subject line
3. FAX them to (813) 354-2722 clearly marking them as bug reports and be sure to reference your filePro License #
4. Call the customer support number (800) 847-4740

A special thank you to Jim Asman for his contribution to the functionality of our printer tables. Jim was a good friend to filePro and is dearly missed.

Contact Information

Surface Mail

fP Technologies, Inc.
432 W. Gypsy Lane Road
Bowling Green, OH 43402

Phones

Support (800) 847-4740
Sales (800) 847-4740
Fax (813) 354-2722

Email

Support support@fpotech.com
Sales sales@fpotech.com
Management filepro@fpotech.com

It's important that you clearly describe a suspected bug and include the filePro version number. If the programmer has trouble figuring out what you meant, you might as well not have reported the bug. Be very specific. For example, if you are reporting a bug concerning a Browse, identify if it is a lookup browse or browse created by using the [F6] key. A screen shot is very helpful and sometimes better than more than 1000 words.

Describe exactly how to duplicate the bug. Although it's sometimes difficult to create a working sample to demonstrate the problem, make every effort to trim down your code and provide a working sample application with test data. You may even discover that what you thought to be a bug is due to a coding error or the bug may only occur with lots of data or large processing tables.

Take good notes as to any error messages and under what circumstances the error message is presented. It never hurts to provide more information rather than not enough. This is particularly true when the programmer asks for additional information. Rather than responding with a single sentence, be verbose since this may shed some light on the bug or what you may be doing wrong in your code.

Read what you wrote. Closely read your bug report before submitting to make sure it's clear and complete. If you have listed steps for duplicating the bug in a sample, exercise the sample with the listed steps to make sure you haven't missed a step.

filePro and filePro Plus are registered
trademarks of fP Technologies, Inc.

=====
Bug fixes are below the New Items.
=====

Version 6.1.02.RR New USP Only Items

=====

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.

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.

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 :".

Added new option to ENCODE() and DECODE(), "URL", to handle URL percent encoding. Failure to decode will return an empty string.

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

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.

Added READMAP(file) function. 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:"

Added a new function 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.

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

+ Number	-- Sequence	----- Description	-----
1	%2 %3	Initialize printer	
2	<page>	New Page	
3		Set Font	

+-----+
if: ' x will contain '<page> '
then: x = PRINTCODE("1")
if: ' x will contain '<page> '
then: x = PRINTCODE("1","0")
if: ' x will contain 'New Page'
then: x = PRINTCODE("2","1")

Added x=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
```

Added 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 retrieve 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.

Added 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.

Added 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.

Added 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.

Added s = GETQUAL(fname) function. GETQUAL() will return 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 :"
```

Added n = GETQUAL(array, fname) function. GETQUAL() will return 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
```

Added new XLSX functions: XL_FREEZEPANE, XL_FREEZEPANE2, XL_SPLITPANE

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.

Added new environmental variable PFXLASCII, default OFF. If enabled, any non-printable ASCII characters will be automatically stripped from data when inserted into an XLSX document.

Enhanced DIM to allow IMPORT and EXPORT commands to 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 commands to support arrays. 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
```

Added `x = 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 "")
```

Added XML import and export code.

`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

Added LOOP commands.

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
```

=====
Version 6.1.01.RR New USP Only Items
=====

Added JSON import and export code.

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

```

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

name = 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. 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.

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")

Added QRCODE FPML print code.

```
<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 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.

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.

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"

Added initial support for multi-dimensional arrays.

DIM array[n1,n2,...,n8] (1,e)
Multi-Dimensional array of fields with length "1" & 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"]

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 average 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.

=====

END OF NEW USP ITEMS

=====

6.1.XX.08 NEW ITEMS

=====

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.

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.

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.

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

=====

6.1.XX.07 NEW ITEMS

=====

Added a F7 last record option to clerk.

Added new system controlled fields for creation time (@CT), update time (@UT), and batch time (@BT) per record. Note: The time is stored in 2 second intervals.

=====

6.1.XX.06 NEW ITEMS

=====

Added a new option to show a stacktrace on a runtime error if PFERRTRACE is set. Default OFF.

Dxmaint will now always show qualifier if PFQUAL is set.

=====

6.1.XX.04 NEW ITEMS

=====

Added PFOLDCHAIN to allow CHAIN to return to the top of processing when a record is saved and the chain was performed inside of an event.

Added basic reconnect functionality into ODBC mirroring upon communications link failure.

Added the ability to directly assign to a longvar when declaring it.
e.g.
declare myvar = "Hello!"

Updated Fuzzy search screen in clerk to be larger and show correct button prompts.

```
n = STACKTRACE(array)
Fill an array with a processing trace, listing the current and past
processing tables and their line numbers to the current line being executed.
This will show lines "jumped" from gosubs and follow calls and functions.
```

Returns the number of elements that could fit into the array.

Added new debugger option "T" to show the current stacktrace while debugging.

=====

6.1.00.03 NEW ITEMS

=====

Updated all programs to no longer require unixODBC by default. unixODBC will now only be required when an ODBC related function is used. If unixODBC is not found when an ODBC function is required, a filePro error will be returned.

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

e.g.
declare myvariable(32,*)="Hello, World!"

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

Added PFPDFAUTOBREAK=ON (default OFF) to allow PDFs to automatically break pages based off of selected paper type.

Added menu letter to menu script editor.

=====
6.1.00.00 NEW ITEMS
=====

You can now use: @wlf<letter>*
ex. @wlfT*
This will apply to any dummy/associated field that begins with 'T'
Overrides any other @wlf*

Added logging to ddefine.
ddefine can now optionally track changes made to filePro file layouts. This includes the name of the file, who changed it, and what fields were changed. Requires a logging configuration file to be added under the ./fp/logs directory named 'ddefine.cfg'. Format of the config file is the same as the servlog.cfg file that comes shipped with filePro.
Example ddefine.cfg:
ROLLING,DEBUG,ddefine.log,60000

xx=FORMERROR
syntax: xx=FORMERROR()
returns: errno from last FORM or FORMM command.
e.g. 2=file not found, 13=permission error

Validate menu script before prompting for removal

Added new option 'C' to F8 Extended Functions for dmoedef to show a list of all print codes on an output format. Selecting an item from the list will jump the editor to it.

TRIM command to remove spaces
aa=ltrim(fld)
left trim
aa=rtrim(fld)
right trim
aa=trim(fld)
trim both left and right

PFIXGT can now be set in dxmaint F8 options.
This is backwards compatible, so if PFIXGT is still set in config, then it is honored by clerk *if true*. If false, the index header is checked for the flag.

Windows fPtransfer now will accept wildcards.

A compress-filePro file routine
fppack

Function:
Remove deleted records from a filePro file, and then (optionally) rebuild all automatic indexes.

Syntax:
fppack [filename | -] [-H heading] [-E] [-R] [-X] [-EX] [-C] [-M name | -MD | -MQ mesg | -MA] [-BG] [-BS]

-H "heading" custom title to display in box.
-E don't actually pack the records, just give statistics.
-R rebuild the automatic indexes even if no records were deleted.
-EX skip statistics
-C skip continue and finished prompts

-X skip rebuilding the auto indexes.
-M name qualifier file name to use.
-MD ask for qualifier with default prompt.
-MQ "mesg" ask for qualifier with "mesg" as the prompt.
-MA use all qualified files & main file.
UNIX/XENIX only:
-BG work in the background.
-BS suppress "completed in background" message.

Added various enhancements to PDF engine.
See on-line or ~/fp/docs PDF documentation.

Added optional error message suppression and basic password auditing to filePro.

PFERRSUPPRESS=ON, default OFF
FFPWAUDIT=ON, default OFF

Password auditing also requires a ./fp/logs/pwaudit.cfg file. Same

structure as servlog.cfg.
Any error that would be sent to mail will still be mailed on
unix/linux based systems.
Errors reported in the background will still be suppressed.
Including the program name.
Invalid password and license errors will still be reported. Password errors
omit the filename.
dcabe and rcabe are exempt from the error suppression.

These functions lock or unlock bytes of the file specified by handle.

```
x=lock(handle,how[,nbyte])
  handle - an open handle to a file
  how    - U|0 : unlock bytes
          L|1 : lock bytes
          N|2 : lock bytes non-blocking
  nbyte  - How many bytes in the file to lock, if omitted, lock
          the billionth byte in the file (file does not have to be
          that large)
```

```
x=unlock(handle[,nbyte])
  handle - an open handle to a file
  nbyte  - How many bytes in the file to unlock, if omitted,
          unlock the billionth byte in the file (file does not
          have to be that large)
```

(returns "1" on success and returns negated system error on error)

ddefine will now create new screens the same as dscreen does instead of just
mono.

NEW command OPENDIR2 to handle long-named files and paths.

```
Syntax:
  N = OPENDIR2(mask, path, fmt_sz, ext_sz, nam_sz)
  All arguments are optional.
  Format Length
  Extension Length
  Fullname Length
```

*cabe lookup wizard will now honor PFQUAL and show qualified indexes

Added new FPML commands to control the appearance of underlines. (See PDF Docs)

New RINSTR, and INSTR now allows negative positions for working backwards.

New Gladmin that will count GUI (GI or Web) sessions, ease of system
and user configuration files and additional security.

Added PDF syntax as an option for printer maintenance (pmaint): Windows only

Lookup Wizard in cabe now allows long vars as key.

Added alias and arrays to F6-D-L display in *cabe.

Updated color with new routines and corrected the shell escape codes.

Automated processing table backups.

```
CABEBACKUP ON|OFF (on by default)
CABEBACKUPMINS n (minutes between backups)
CABEBACKUPCT n (backup files per process)
```

Menu maintenance (makemenu) now asks if you wish to remove
an unused menu script if the menu item is not used.

*report now allows one to use .outs from a pathed directory library

SCREEN command can switch fields in a POPUP UPDATE -, provided no screen name
is passed to the SCREEN command.

MEMO EDIT now accept maxsize to limit the number of
characters that can be entered into a memo field.
memo NNN edit (row,col,lines,width,startLine,startcol,maxSize)
(Text mode only)

Added option 7 to dxmaint to clear qualifier

New -SE *report flag to allow report to edit/save a selection set.

Added @EXIT label to *clerk processing. This is executed whenever
a record is exited or broken out of. Events that trigger this are
'X' while not in update mode, 'BRKY' while not in update mode, and
'exit' in processing. It is the opposite of @entsel, and is the last
thing executed when leaving a record. Assignment of real fields is
not allowed, this is similar to @once in that the processing that is
executed is NOT sitting on a record, but rather record '0'.

Partial lookup flag added to *cabe lookup wizard.

-O on an exact lookup now does partial key matching. This kills a
lookup once the beginning of the key value no longer matches the lookup
key value.

BUSYBOX

```
BUSYBOX "my message"
BUSYBOX("10","10")
BUSYBOX("10","10") "my message"
```

Added PFFFFULLPATH as an enhancement to PDFPOSTPRINT
and added a PFNEWPOSTPRINT alias to name to PDFPOSTPRINT
Added PFFFFULLPATH to augment the filename passed to the post print
handler, default ON, this causes the filename passed to the postprint
script to contain the full path to the file, not just the file name.
Set to OFF to revert to old behaviour. PFPOSTPRINTnnn will now work
with normal file destinations. Same rules as the old global PFPOSTPRINT
but also supports PDF files.

Clerk will now allow a full path to a form when using
the FORM and FORMM command in processing.

User defined functions

```
Forward declare functions to be used:
(function|func) [file.]name([dim|var] var1, [dim|var] var2, ...)
```

e.g.

```
function fplib.showlock(var pid)
function fplib.log(file, line, what)
function somefunc(dim myarray)
```

Call a function:

```
[x]=[file.]name(var1, var2, ...)
```

Return a value from a function:

```
return(value)
```

Can pass fields: real, dummy, longvar

Can pass arrays: Alias and system arrays are copied to a non-aliased
array. Non-aliased arrays are passed by reference.

Function names must be at least 3 characters in length.
Functions cannot modify values outside of its scope.
Functions do not call automatic processing.
Functions cannot modify real fields.
Functions cannot be called unless it they are declared.
Functions can pass values by reference (changes made to the value will
carry back out of the function, only to arrays).
Functions can optionally return a value.

Parameter names must be at least 3 characters in length.
Parameters will be passed to the function using the name they were defined
with in the declaration statement.

Environment variables:

```
PFFUNCDBG=(ON|OFF), default OFF.
If ON the debugger will be allowed to continue into the function
call. If OFF the debugger will skip over user defined functions.
NOTE: Debug statements inside of functions will still be able to
be activated. If debug is set inside of a function, it will
continue even after the function is left.
```

Example:

Processing table for fibonacci:

```
If:                ' Declare for future use
Then: function fibonacci(nval)
If:                ' Get the parameter
Then: declare extern nval
If: nval le "1"    ' Return the result
Then: return(nval)
If:                ' Return the result
Then: return(fibonacci(nval-"1")+fibonacci(nval-"2"))
```

Usage:

```
If:                ' Declare for future use
Then: function fibonacci(nval)
If:                ' Call the function
Then: n=fibonacci("9")
If:                ' Display the result
Then: msgbox ""{n ' Prints "34"
```

EXTERN and GLOBAL arrays

```
DIM GLOBAL name(size)
DIM EXTERN name
```

Only non-aliased arrays can be declared GLOBAL/EXTERN.
Functions similar to GLOBAL/EXTERN longvars.

New compare condition for Associated Fields

Added new selection set relational operators:

```
AEQ - Associated field, all equal
ANE - Associated field, all not equal
ACO - Associated field, all contain
```

These require ALL components of an associated field to match the
comparison being done, rather than just one of its component fields.

New functions for creating XLSX documents from filePro.

```
e = XL_OPEN(file [, name])
Start building an XLSX output file.
```

Parameters -

file : Path to the file to create. If no full path is given the generated file will be placed in the PFTMP or equivalent directory.
name : The name for the default sheet that will be created. Defaults to Sheet1.

If the filename does not end in ".xlsx" it will be added on creation.

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

Note: Only one XLSX file can be created at a time.

e = XL_SAVE([password])

Save the current XLSX file.

Parameters -

password : If specified, encrypt the XLSX output file using Agile encryption (AES128).

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

Note: Encrypted XLSX files cannot be opened with most third party programs such as LibreOffice and OpenOffice. They are fully supported by Excel however. The documents are saved in an encrypted CFB file.

handle = XL_ADDSHEET([name])

Add a new sheet to the XLSX document.

Parameters -

name : The name for the sheet to be created. Defaults to auto naming the sheet based on the Sheet1, Sheet2, ..., SheetN template.

Returns a handle to a new sheet object on success and "-1" on error. XL_ERROR() can be called to return the last error.

e = XL_ADDCELL([data [, style [, sheet [, row [, col]]]])

Add a new cell to the XLSX document.

Parameters -

data : Data to be inserted into the document. A cell starting with '=' will be treated as a formula.
style : Handle to style to be used for this cell. Use blank to use the default style.
sheet : Handle to sheet to insert the cell on. Use blank, "0", or "-1" to use the default sheet.
row : Row to place the cell (0 indexed).
col : Column to place the cell (0 indexed).

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

Note: Using an empty or negative row/column value will cause the cell to be added using the auto counter in the sheet, incrementing the column value after the cell is added. Specifying a location will reposition the auto counter. Formulas can be used as part of the data as well by prefixing the string with '='.

e = XL_ADDCELL2([data [, style [, sheet [, cell]]]])

Add a new cell to the XLSX document.

Parameters -

data : Data to be inserted into the document. A cell starting with '=' will be treated as a formula.
style : Handle to style to be used for this cell. Use blank to use the default style.
sheet : Handle to sheet to insert the cell on. Use blank, "0", or "-1" to use the default sheet.
cell : The Excel style cell to insert the cell. e.g. "A1" "D6" "F6".

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

Note: Using an empty cell number will cause the cell to be added using the auto counter in the sheet, incrementing the column value after the cell is added. Specifying a location will reposition the auto counter. Formulas can be used as part of the data as well by prefixing the string with '='.

handle = XL_FORMAT(format)

Create a new format to use with the XLSX document.

Parameters -

format : Excel format string to use to format the a style. e.g.
"\$ #,###,nnn.nn"
"% ##n.n"

"m/d/yyyy"

Returns a handle to a new format object on success and "-1" on error.
XL_ERROR() can be called to return the last error.

e = XL_COLWIDTH(width, firstcol, lastcol [, sheet])
Change the default column width for a sheet between a range.

Parameters -
width : Width of the column(s). e.g. "24" "12.5", "11"
firstcol : Zero based column index or column letter to set from.
lastcol : Zero based column index or column letter to set to.
sheet : Handle to sheet to change the cell widths.

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

handle = XL_FONT(font, [size [, attr [, color]])
Create a new font to use with the XLSX document.

Parameters -
font : Name of the font to use.
size : Point size of the font. e.g. "11" "8.42" "12", default "11.0"
attr : List of attributes to apply to this font, separated by commas.
e.g. "bold,italic"
Values:
"bold"
"italic"
"underline"
"strike"
"unlocked"
"hidden"
"wrap"
"shrink"
"fill"
"left"
"center"
"right"
"justify"
"top"
"bottom"
"vjustify"
"vcenter"
color : The RGB Hex value to set the font color.
e.g. "000000" "ADD8E6"

Returns a handle to a new font object on success and "-1" on error.
XL_ERROR() can be called to return the last error.

handle = XL_BORDER(borderstyle [, color])
Create a new border to use with the XLSX document.

Parameters -
borderstyle : The style to use with this border. Must be one of the following values:
"thin"
"medium"
"dashed"
"dotted"
"thick"
"hair"
"medium_dashed"
"dash_dot"
"medium_dash_dot"
"dash_dot_dot"
"medium_dash_dot_dot"
"slant_dash_dot"
color : The RGB Hex value to set the border color.
e.g. "000000" "ADD8E6"

Returns a handle to a new border object on success and "-1" on error.
XL_ERROR() can be called to return the last error.

handle = XL_FILL(bg [, fg [, fill]])
Create a new fill to use with the XLSX document.

Parameters -
bg : The RGB Hex value to set the background fill color.
e.g. "000000" "ADD8E6"
fg : The RGB Hex value to set the foreground fill color.
e.g. "000000" "ADD8E6"
fill : The fill pattern to use, defaults to "solid" fill. Value must be one of the following.
"solid"
"medium_gray"
"dark_gray"
"light_gray"
"dark_horizontal"
"dark_vertical"

```
"dark_down"  
"dark_up"  
"dark_grid"  
"dark_trellis"  
"light_horizontal"  
"light_vertical"  
"light_down"  
"light_up"  
"light_grid"  
"light_trellis"  
"gray_125"  
"gray_0625"
```

Returns a handle to a new fill object on success and "-1" on error.
XL_ERROR() can be called to return the last error.

```
e = XL_ADD_DT(date, time [, style [, sheet [, row [, col]]]])  
Combine two fields into a single spreadsheet datetime field and insert it as  
a new cell in the XLSX document.
```

Parameters -
date : filePro date field.
time : filePro time field.
style : Handle to style to be used for this cell. Use blank to use the
default style.
sheet : Handle to sheet to insert the cell on. Use blank, "0", or "-1"
to use the default sheet.
row : Row to place the cell (0 indexed).
col : Column to place the cell (0 indexed).

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

```
e = XL_ADD_DT2(date, time [, style [, sheet [, cell]])  
Combine two fields into a single spreadsheet datetime field and insert it as  
a new cell in the XLSX document.
```

Parameters -
date : filePro date field.
time : filePro time field.
style : Handle to style to be used for this cell. Use blank to use the
default style.
sheet : Handle to sheet to insert the cell on. Use blank, "0", or "-1"
to use the default sheet.
cell : The Excel style cell to insert the cell. e.g. "A1" "D6" "F6".

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

```
handle = XL_CHART(type [, title [, xname [, yname [, row [, col [, stylenum  
[, sheet [, xoff [, yoff [, xscale [, yscale]]]]]]]]]])  
Add a new chart to the XLSX document.
```

Parameters -
type : Type of chart to create. Must be one of the following values.
"area"
"area_stacked"
"area_stacked_percent"
"bar"
"bar_stacked"
"bar_stacked_percent"
"column"
"column_stacked"
"column_stacked_percent"
"doughnut"
"line"
"line_stacked"
"line_stacked_percent"
"pie"
"scatter"
"scatter_straight"
"scatter_stright_markers"
"scatter_smooth"
"scatter_smooth_markers"
"radar"
"radar_with_markers"
"radar_filled"
title : The title for this chart.
xname : The title for the x-axis.
yname : The title for the y-axis.
row : Row to place the cell (0 indexed).
col : Column to place the cell (0 indexed).
stylenum : Number of the built in Excel style to use. Must be between
"1" and "48". The default style is 2. The value is one of
the 48 built-in styles available on the "Design" tab in
Excel 2007.
sheet : Handle to sheet to insert the chart on. Use blank, "0", or
"-1" to use the default sheet.
xoff : X axis offset to place the chart, in pixels.
yoff : Y axis offset to place the chart, in pixels.

xscale : Scale the chart along the x axis. e.g. "1", "0.5" "2". Value cannot be negative.
yscale : Scale the chart along the x axis. e.g. "1", "0.5" "2". Value cannot be negative.

Returns a handle to a new chart object on success and "-1" on error.
XL_ERROR() can be called to return the last error.

Note: The chart functions do not use the auto counter found in the sheets and instead will default to "0", "0" or "A1" when used for insertion.

```
handle = XL_CHART2(type [, title [, xname [, yname [, cell [, stylenum [, sheet  
[, xoff [, yoff [, xscale [, yscale]]]]]]]]])  
Add a new chart to the XLSX document.
```

Parameters -

type : Type of chart to create. Must be one of the following values.
"area"
"area_stacked"
"area_stacked_percent"
"bar"
"bar_stacked"
"bar_stacked_percent"
"column"
"column_stacked"
"column_stacked_percent"
"doughnut"
"line"
"line_stacked"
"line_stacked_percent"
"pie"
"scatter"
"scatter_straight"
"scatter_stright_markers"
"scatter_smooth"
"scatter_smooth_markers"
"radar"
"radar_with_markers"
"radar_filled"
title : The title for this chart.
xname : The title for the x-axis.
yname : The title for the y-axis.
cell : The Excel style cell to insert the cell. e.g. "A1" "D6" "F6".
stylenum : Number of the built in Excel style to use. Must be between "1" and "48". The default style is 2. The value is one of the 48 built-in styles available on the "Design" tab in Excel 2007.
sheet : Handle to sheet to insert the chart on. Use blank, "0", or "-1" to use the default sheet.
xoff : X axis offset to place the chart, in pixels.
yoff : Y axis offset to place the chart, in pixels.
xscale : Scale the chart along the x axis. e.g. "1", "0.5" "2". Value cannot be negative.
yscale : Scale the chart along the x axis. e.g. "1", "0.5" "2". Value cannot be negative.

Returns a handle to a new chart object on success and "-1" on error.
XL_ERROR() can be called to return the last error.

Note: The chart functions do not use the auto counter found in the sheets and instead will default to "0", "0" or "A1" when used for insertion.

```
handle = XL_CHARTSHEET(type [, title [, xname [, yname [, stylenum]]])  
Add a new chartsheet to the XLSX document. A chartsheet is a full chart that occupies it's own sheet and cannot contain any cells.
```

Parameters -

type : Type of chart to create. Must be one of the following values.
"area"
"area_stacked"
"area_stacked_percent"
"bar"
"bar_stacked"
"bar_stacked_percent"
"column"
"column_stacked"
"column_stacked_percent"
"doughnut"
"line"
"line_stacked"
"line_stacked_percent"
"pie"
"scatter"
"scatter_straight"
"scatter_stright_markers"
"scatter_smooth"
"scatter_smooth_markers"
"radar"
"radar_with_markers"
"radar_filled"
title : The title for this chart.

xname : The title for the x-axis.
yname : The title for the y-axis.
stylenum : Number of the built in Excel style to use. Must be between "1" and "48". The default style is 2. The value is one of the 48 built-in styles available on the "Design" tab in Excel 2007.

Returns a handle to a new chartsheet object on success and "-1" on error. XL_ERROR() can be called to return the last error.

e = XL_SERIES(chartnum, sheet, namerow, namecol, cfirstrow, cfirstcol, clastrow, clastcol, vfirstrow, vfirstcol, vlastrow, vlastcol)
Add a series to a chart or chartsheet.

Parameters -

chartnum : Handle to a chart or chartsheet to add series.
sheet : Handle to sheet to get values from. Use blank, "0", or "-1" to use the default sheet.
namerow : Series name row (0 indexed).
namecol : Series name column (0 indexed).
cfirstrow : Categories first row (0 indexed).
cfirstcol : Categories first column (0 indexed).
clastrow : Categories last row (0 indexed).
clastcol : Categories last column (0 indexed).
vfirstrow : Values first row (0 indexed).
vfirstcol : Values first column (0 indexed).
vlastrow : Values last row (0 indexed).
vlastcol : Values last column (0 indexed).

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

e = XL_SERIES2(chartnum, sheet, namecell, cfirst, clast, vfirst, vlast)
Add a series to a chart or chartsheet.

Parameters -

chartnum : Handle to a chart or chartsheet to add series.
sheet : Handle to sheet to get values from. Use blank, "0", or "-1" to use the default sheet.
namecell : Series name Excel style cell. e.g. "A1" "D6" "F6".
cfirst : Categories first Excel style cell. e.g. "A1" "D6" "F6".
clast : Categories last Excel style cell. e.g. "A1" "D6" "F6".
vfirst : Values first Excel style cell. e.g. "A1" "D6" "F6".
vlast : Values last Excel style cell. e.g. "A1" "D6" "F6".

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

e = XL_PROTECTSHEET(sheet, password)
Add a password to restrict editing of a sheet.

Parameters -

sheet : Handle to sheet to protect. Use blank, "0", or "-1" to use the default sheet.
password : Password to use to protect this sheet.

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

e = XL_PROTECTCHARTSHEET(cs, password)
Add a password to restrict editing of a chartsheet.

Parameters -

cs : Handle to chartsheet protect.
password : Password to use to protect this sheet.

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

e = XL_ERROR()
Return the last error generated by the XLSX set of functions.

Returns the last error string generated by the XLSX engine.

e = XL_SETPOS(row [, col [, sheet]])
Set the auto counter position for a sheet.

Parameters -

row : Row to move auto counter to (0 indexed).
col : Column to move auto counter to (0 indexed).
sheet : Handle of sheet to set. Use blank, "0", or "-1" to use the default sheet.

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

e = XL_SETPOS2(cell [, sheet])
Set the auto counter position for a sheet.

Parameters -
cell : Excel style cell to set the auto counter to. e.g. "A1" "D6".
sheet : Handle of sheet to set. Use blank, "0", or "-1" to use the default sheet.

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

e = XL_NEXTRROW([sheet])
Move the auto counter down a row for a sheet.

Parameters -
sheet : Handle of sheet to set. Use blank, "0", or "-1" to use the default sheet.

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

e = XL_NEXTCOL([sheet])
Move the auto counter one column for a sheet.

Parameters -
sheet : Handle of sheet to set. Use blank, "0", or "-1" to use the default sheet.

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

handle = XL_STYLE([font [, fill [, fmt [, btop [, bbot [, bleft
[, bright]]]]]])
Add a new style to the XLSX document.

Parameters -
font : Handle to font object to use.
fill : Handle to fill object to use.
fmt : Handle to format object to use.
btop : Handle to border object to use for top border.
bbot : Handle to border object to use for bottom border.
bleft : Handle to border object to use for left border.
bright : Handle to border object to use for right border.

Returns a handle to a new style object on success and "-1" on error. XL_ERROR() can be called to return the last error.

e = XL_IMAGE(img [, row [, col [, sheet [, xoff [, yoff [, scalex [, scaley
[, flag]]]]]]]])
Add a new image to the XLSX document.

Parameters -
img : Path to image file to use.
row : Row to insert the image on (0 indexed).
col : Column to insert the image on (0 indexed).
sheet : Handle of sheet to insert image. Use blank, "0", or "-1" to use the default sheet.
xoff : X-axis offset for the image, in pixels.
yoff : Y-axis offset for the image, in pixels.
scalex : Scale the image along the x-axis. e.g. "1", "0.5" "2". Value cannot be negative.
scaley : Scale the image along the y-axis. e.g. "1", "0.5" "2". Value cannot be negative.
flag : Option of how to position image.
"0" - Default positioning.
"1" - Move and size image with the cells.
"2" - Move but don't size image with the cells.
"3" - Don't move or size the image with the cells.
"4" - Same as "1" but wait to apply hidden cells.

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

Note: The image functions only support PNG, JPEG, and BMP files.

e = XL_IMAGE2(img [, cell [, sheet [, xoff [, yoff [, scalex [, scaley
[, flag]]]]]]]])
Add a new image to the XLSX document.

Parameters -
img : Path to image file to use.
cell : Excel style cell to insert the image. e.g. "A1" "D6" "F6".
sheet : Handle of sheet to insert image. Use blank, "0", or "-1" to use the default sheet.
xoff : X-axis offset for the image, in pixels.
yoff : Y-axis offset for the image, in pixels.
scalex : Scale the image along the x-axis. e.g. "1", "0.5" "2". Value cannot be negative.

scaley : Scale the image along the y-axis. e.g. "1", "0.5" "2". Value cannot be negative.
flag : Option of how to position image.
"0" - Default positioning.
"1" - Move and size image with the cells.
"2" - Move but don't size image with the cells.
"3" - Don't move or size the image with the cells.
"4" - Same as "1" but wait to apply hidden cells.

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

Note: The image functions only support PNG, JPEG, and BMP files.

e = XL_LASTCMD()
Get debug information about the last XLSX call.

Returns the last evaluated command parse string.

e = XL_MARGINS([left, [right, [top, [bottom, [sheet]]]])
Set the worksheet print margins.

Parameters -
left : Left margin in inches, e.g. "0.5", "1", "0.75". A blank or negative value will use the default of "0.7".
right : Right margin in inches, e.g. "0.5", "1", "0.75". A blank or negative value will use the default of "0.7".
top : Top margin in inches, e.g. "0.5", "1", "0.75". A blank or negative value will use the default of "0.75".
bottom : Bottom margin in inches, e.g. "0.5", "1", "0.75". A blank or negative value will use the default of "0.75".
sheet : Handle of sheet to set the margins. Use blank, "0", or "-1" to use the default sheet.

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

e = XL_LANDSCAPE([sheet])
Set the worksheet to print in landscape mode.

Parameters -
sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use the default sheet.

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

e = XL_PORTRAIT([sheet])
Set the worksheet to print in portrait mode.

Parameters -
sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use the default sheet.

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

e = XL_GRIDLINES(option [, sheet])
Set if the worksheet should display gridlines when printed.

Parameters -
option : Which Gridlines to print. Cannot be blank. Must be one of the following values.
"hide_all"
"show_all"
"show_screen"
"show_print"
sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use the default sheet.

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

e = XL_FITPAGES([height, [width, [sheet]])
Fit the printed area to a specific number of pages both vertically and horizontally.

Parameters -
height : Number of pages vertically. A value of "0" or blank will set the height as necessary.
width : Number of pages horizontally. A value of "0" or blank will set the height as necessary.
sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use the default sheet.

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

e = XL_PAPERTYPE(type [, sheet])
Set the paper format for the printed output of a worksheet.

Parameters -

type : The paper format to use with a printed worksheet. Must be one of the following values.

"default"
"letter"
"tabloid"
"ledger"
"legal"
"statement"
"executive"
"a3"
"a4"
"a5"
"b4"
"b5"
"folio"
"quarto"
"10x14"
"11x17"
"note"
"envelope"
"envelope_9"
"envelope_10"
"envelope_11"
"envelope_12"
"envelope_14"
"c"
"d"
"e"
"envelope_d1"
"envelope_c3"
"envelope_c4"
"envelope_c5"
"envelope_c6"
"envelope_c65"
"envelope_b4"
"envelope_b5"
"envelope_b6"
"monarch"
"fanfold"
"german_std_fanfold"
"german_legal_fanfold"

sheet : Handle of sheet to change type. Use blank, "0", or "-1" to use the default sheet.

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

e = XL_CENTERH([sheet])

Center the worksheet data horizontally between the margins on the printed page.

Parameters -

sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use the default sheet.

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

e = XL_CENTERV([sheet])

Center the worksheet data vertically between the margins on the printed page.

Parameters -

sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use the default sheet.

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

e = XL_PRINTACROSS([sheet])

Change the default print direction to across then down.

Parameters -

sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use the default sheet.

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

e = XL_SETHEADER(string [, margin, [limage, [cimage, [rimage, [sheet]]]])

Set the printed page header.

e = XL_SETFOOTER(string [, margin, [limage, [cimage, [rimage, [sheet]]]])

Set the printed page footer.

Parameters -

string : The header/footer definition string. See below for format options. Cannot be blank.
margin : The margin in inches to use for the header/footer. A blank, "0", or negative value will use the default margin of "0.3".
limage : Full path to an image to use in place of the left image placeholder.
cimage : Full path to an image to use in place of the center image placeholder.
rimage : Full path to an image to use in place of the right image placeholder.
sheet : Handle of sheet to set header/footer. Use blank, "0", or "-1" to use the default sheet.

Format Options -

Control	Category	Description
&L	Justification	Left
&C		Center
&R		Right
&P	Information	Page number
&N		Total number of pages
&D		Date
&T		Time
&F		File name
&A		Worksheet name
&Z		Workbook path
&fontsize	Font	Font size
&"font,style"		Font name and style
&U		Single underline
&E		Double underline
&S		Strikethrough
&X		Superscript
&Y		Subscript
&[Picture]	Images	Image placeholder
&G		Same as &[Picture]
&&	Miscellaneous	Literal ampersand &

Text in headers and footers can be justified to the left, center and right by prefixing the text with the control characters &L, &C and &R. For example, "&LHello, World!", "&CHello, World!", "&RHello, World!"

For simple text, if the justification is not specified the text will be center aligned. However, you must prefix the text with &C if you use any other formatting.

You are limited to 3 images in a header/footer.

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

Note: The image types supported are PNG, JPEG, and BMP files. There is a hard limit of 255 characters in a header/footer string, including control characters. Strings longer than this will not be written to the document.

e = XL_SETBACKGROUND(image [, sheet])
Set the background image for a worksheet.

Parameters -

image : Full path to an image to use as the sheet background.
sheet : Handle of sheet to set background image. Use blank, "0", or "-1" to use the default sheet.

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

Note: The image types supported are PNG, JPEG, and BMP files.

e = XL_HIDEZEROS([sheet])
Hide zero values in worksheet cells.

Parameters -

sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use the default sheet.

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

e = XL_SHOWROWCOL([sheet])
Show row and column headers on the printed page.

Parameters -

sheet : Handle of sheet to change mode. Use blank, "0", or "-1" to use

the default sheet.

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

Rebuild All Indexes on a file. item '8' on the dialog. Note: this is in the "extended" dialog which shows when a filename is not specified from the command line. Indexes can be selected individually, or all (with F7). Press SAVE, and rebuild begins

Ability to SPLIT data into array

Usage:

```
sz=SPLIT(array, string, delimiter)
array is the array that the data will be placed into
string is the data to split
delimiter is the sequence of characters to split on
```

NOTE: The array being used must have the size defined for its elements and cannot be an alias.

Added the ability to show record locks from *clerk. Can also be used to terminate sessions directly. New option !L added to *clerk. Using !L will activate the new locked records list. Enter on a selected entry will give additional options to the user, including the ability to Kill or Terminate a locked process without having to go to the command line. Note: This option is only available on Unix/Linux/BSD

Added UID mapping to filePro, ddir/dprodir option F5.

This allows for UIDs (User IDs) to be aliased to specific usernames. In the event that a login account is removed from your system, this can be used to maintain the link between the removed login's UID and those stored in filePro, effectively allowing system variables such as @CB and @UB to be maintained.

Windows Only:

This also has the added benefit of allowing @CB and @UB to function on Windows by linking a "pseudo" UID to a given username. These UIDs are automatically generated but can also be manually added. When a user opens filePro and their username does not exist in the UID map file, a UID will be generated for that user. filePro will find the next available UID in the list, starting from 2000, and assign it to that username.

On all platforms, UIDs stored in this program must be unique and in the range 0-65535. Usernames can be duplicated on Unix and Linux platforms, but must be unique on Windows.

Usernames are case-sensitive on Unix and Linux platforms and are case-insensitive on Windows platforms.

Environmental Variables:

```
PFUIDMAP = /path
Alternate filePro UID map file. (Use full path)
Note: Must be set in the environment.
```

```
PFUSEUIDMAP = ON
```

Allows filePro to do UID mapping. Also expands the maximum username length returned by @CB, @UB, and @ID to 32.
Default: ON

String Functions

All "is" functions return "1" for true and "0" for false.

```
x=isalpha(fld [, pos])
Is the character at the position given a letter?
```

```
x=isdigit(fld [, pos])
Is the character at the position given a number?
```

```
x=isalnum(fld [, pos])
Is the character at the position given a letter or number?
```

```
x=isspace(fld [, pos])
Is the character at the position given a whitespace character?
' ', '\t', '\n', '\r', '\v', '\f'
```

```
x=islower(fld [, pos])
Is the character at the position given lowercase?
```

```
x=isupper(fld [, pos])
Is the character at the position given uppercase?
```

```
x=isxdigit(fld [, pos])
Is the character at the position given a hexadecimal character?
'0'-'9', 'A'-'F'
```

```
x=iscntrl(fld [, pos])
Is the character at the position given a control character?
ASCII codes 0x00 (nul) - 0x1f (US), and 0x7f (del)
```

```
x=isprint(fld [, pos])
```

Is the character at the position given a printable character?
ASCII codes greater than 0x1f (US) not including 0x7f (del)

x=ispunct(fld [, pos])
Is the character at the position given a punctuation character?

x=isgraph(fld [, pos])
Is the character at the position given a character with a graphical representation? The characters with graphical representation are all those characters than can be printed (as determined by isprint) except for space.

x=tolower(fld [, pos])
Return the character at the position given as a lowercase character.

x=toupper(fld [, pos])
Return the character at the position given as an uppercase character.

str=strtolower(fld)
Return the entire string converted to lowercase.

str=strtoupper(fld)
Return the entire string converted to uppercase.

Added new array size function to get the size of an array. Can be used with GLOBAL, EXTERN, LOCAL, and SYSTEM arrays.

x=ARRAYSIZE(array)
Where array is the name of the array.
Where x is the returned size of the passed array.

Added new DECLARED function to check if an array or longvar is defined, meaning it is either declared LOCAL or GLOBAL or is declared EXTERN but has a matcing GLOBAL definition.

x=DECLARED(var)
Where var is either a longvar or an array.
Where x is the return value.
Returns 0 if the variable is not fully defined.
Returns 1 if the variable is fully defined.

Increased ACTION length in debugger from 60 characters to full 128.
Should now be the same as *cabe.

Added new flag -DM to [dr]clerk to disable the Index Mode prompt from @ENTSEL. Only works when not in update mode.

Added flag -RH to report to disable the automatic record number reporting in the middle of the screen. This enables placing text on the center of the screen without it being overwritten when the display updates.

x=@GUI.PAUSE()
Pauses automatic screen updating while in GI/Web.

x=@GUI.RESUME()
Resumes automatic screen updating while in GI/Web.

REPLACE() enhancement - allow null characters
Enhanced REPLACE() to accept null characters

FORM WITHPROC
FORM WITHPROC "formname"
FORMM WITHPROC "formname"

Added additional command switch to FORM and FORMM commands to allow the associated processing table to run while in input processing.

Note: You cannot call the WITHPROC variant from within another form UNLESS the calling form is a processing only form.

Addqual Program

Addqual allows you to easily add qualifiers to your files either interactively or through the command line.

This runs interactively:
addqual [filename]

This runs automatically:
addqual filename -q <qualname>
as does this:
addqual filename -q <qualname> -x <qual-to-copy-from>

The automatic commands will display graphics on errors. You can keep graphics off with "-s" and errors will be printed on the command line if they occur.

example:
addqual filename -q <qualname> -s

List of switches:
-q qualifier to create
-x qualifier to copy indexes from

-s silent, no graphics
-h --help syntax help

XFER - encrypted transfers server-peer

CABE F6 list files from F8 L-Load

=====
6.1.XX.08 Bug Fixes
=====

Fixed the F7 last record option when selecting a record in dclerk/rcclerk while running under fileProWeb or filePro GIClient.

Corrected an issue where an ODBC table wouldn't report as existing.

Corrected an issue with Find and Replace in dcabe/rcabe where control codes were incorrectly being interpreted as character and color codes.

Corrected an issue in dreport/rreport where exiting with an EXIT action in @DONE would not use the supplied exit value if zero records were selected.

Updated F6-D-L listing in dcabe/rcabe to restore typedown behaviour.

Fixed an issue in rcabe, dcabe, and dclerk where a syntax error could cause a crash upon saving or continuing.

Fixed a show map (F6 - View Fields) regression in dcabe/rcabe where button labels were missing valid options.

Fixed an issue with the bottom screen display on Windows in dscreen.

Reworked the SPLIT() command to work with all standard array types.

Fixed SPLIT() return value. There was an issue where the number of elements returned could have been more than the returned type allowed.

SPLIT() now clears the array before updating values and will now honor the edit type(s) of the array, rather than always treating the data as a string.

Fixed a crash when using READOUTPUT() in processing.

Corrected a memory leak in LISTBOX() and SELECTBOX().

Corrected an issue with the "!L" escape in dclerk/rcclerk. Previously, the calculation to determine the locked record number was incorrect.

Added an additional check when saving a record in dclerk/rcclerk so that a zero length must-fill field will be treated as filled.

Corrected an issue with fields resetting when using user defined browses in clerk when a screen contains scrolling fields.

=====
Version 6.1.XX.07 bug fixes
=====

Fixed UID import feature in ddir where it wasn't finding any files.

User can now save a blank UID map in ddir.

Corrected a lock issue with UID maps.

Added additional error messages when importing files for UID mapping.

Fixed an issue with VARCHAR fields not working when using some ODBC drivers.

Corrected an issue with script cleanup causing a crash in dmakemenu.

Fixed a crash when adding an index to an existing empty file in ddefine.

Fixed a crash in fpack when rebuilding an index containing system controlled fields.

Corrected an issue in fpsql where viewing a file's layout would not retain the previous selection.

Fixed an issue in all runtime programs where aliased real fields in an array would not explicitly write on end.

Corrected a potential crash when adding a duplicate key to an index.

Fixed a crash caused by inserting a new unique key after a very long chain of duplicate keys to an index.

Fixed a break key issue in cab F6 label lookup in F9 search. filePro was requiring twice as many break key presses than was actually required.

Fixed a crash when creating a selection set and pressing F6 while in a relationship field.

=====
Version 6.1.XX.06 bug fixes
=====

Corrected an issue on Linux/BSD where fuzzy search could cause a crash.

Fixed array handling in user defined functions that could cause a crash.

Fixed associated field comparisons in clerk and report. Was previously only comparing the first field in a set.

Corrected some command line arguments being ignored in ddefine, autosshuf, and doresync.

Corrected edit types not being tokenized correctly in rcabe.

Added duplicate variable check when saving in ddir/dprodir.

Updated fuzzy search to better handle long fields.

Fix a crash when moving through a line that contains a malformed CALL statement in cabc.

Fixed a crash in find and replace in cabc.

Fixed a crash when copying lines that don't exist in the file in cabc.

Corrected various files not being copied correctly in fpcopy.

Fixed indexes on qualified files in fpcopy.

Added sanity check to locked records check in clerk.

SPLIT() - Removed restriction on delimiter size. Size of array elements still need to be defined for destination.

Fixed syntax error line reporting in cabc when jumping to a different place in a prc file. Cabc now shows the line number in the editor correctly.

Corrected an issue loading tokenized global arrays in rclerk and rreport.

=====
Version 6.1.XX.05 bug fixes
=====

Corrected an issue when using the Rebuild Indexes option in dxmaint where options were not toggling correctly.

Fixed a regression where scrollable fields in a popup weren't displaying correctly.

Fixed message boxes to better handle filePro escape codes.

Fixed -pv flag and print to screen to no longer corrupt the output.

Fixed alternate automatic processing loading in cabc, preventing variables from resolving correctly during syntax check.

Fixed a too many open files bug in fpcopy when working on a file with many qualifiers and indexes.

Corrected fppack to correctly handle encrypted files.

Changed index rebuild message location on the screen to no longer be hidden behind the progress updates.

Corrected an issue preventing GI/fileProWeb from loading [dr]report and [dr]clerk on Windows.

Fixed a crash with the PDF import code.

Corrected a crash when opening more than one JSON file at a time.

=====
Version 6.1.XX.04 bug fixes
=====

Fixed an issue where libodbc would not correctly be found when initializing features that use ODBC.

Corrected an issue with RINSTR() where the starting position wasn't honored correctly.

Option 'C' to clear selection set in [dr]clerk will no longer cause an infinite loop when going back into index selection.

Updated PNG support for PDF outputs. Previously, some PNG files would appear corrupted when imported.

Corrected a potential crash when moving/reordering blob fields inside of dmoedef.

Added PFOLDCHAIN to allow CHAIN to return to the top of processing when a record is saved and the chain was performed inside of an event.

Updated listbox and selectbox code to no longer go outside of screen bounds.

Fixed date handling in XLSX generation when not using the datetime functions.

Fixed an issue where blobs/memos could become corrupted if assigning to the

field more than once without writing the record.

=====
Version 6.1.XX.03 bug fixes
=====

Updated tokenization engine to increase parsing speed.

Corrected Memory fault in FPSQL

Corrected licinfo to read license fallback file.

Corrected memory leaks in [dr]clerk and [dr]report.

Corrected issue where a select or list box would not clear correctly from the screen.

Fixed positioning and moving objects (memo) on a form.

Corrected button text in F6 cabe.

Fixed an early error exit condition in ddir to report an error rather than exiting.

Corrected "stair step" issue in cabe when using the -C flag on Linux/Unix.

Corrected a crash in clerk when using F5 to duplicate fields between records.

Updated F5 duplicate key in clerk to work with scrolling fields.

Added PFREUSEADDR=ON (default ON) to enable a port to be rebound more quickly when using sockets.

Added code to prevent a dummy field from being used as a foreign key when performing a fuzzy search.

Corrected and reverted wildcard behaviour during selection in clerk.

Corrected type checking for associated fields in selection sets.

Added buttons to clerk fuzzy search for scrolling the file map.

Increased the number of fields shown in fuzzy search in clerk.

Fixed some button shifting for F6 key in cabe.

Corrected ALL operator in short selection to properly update the selection popup.

=====
Version 6.1.XX.02 bug fixes
=====

Task #1948 Autosave not honoring config flags
Corrected an issue where Autosave was not correctly reading config variables. Added initial change backup.

Task #1950 Scrolling fields in popups break placement
Corrected an issue when drawing a popup that contains a scrolling field.

Task #1951 Enhanced runtime format for WHEN flags
Enhanced runtime format to support extended WHEN flags.

Added support for @WUKx* @WHPx* and @WBLx*. New WHEN values will be ignored in older versions of filePro.

=====
Version 6.1.XX.01 bug fixes
=====

Task #1945 ALL fields search in selection broken
Corrected ALL field search code for selections.

Task #1947 Short selection prompting twice
Corrected an issue where short selection was displaying the old selection screen.

Task #1949 Enable REVERT command
Correctly enabled the REVERT command for release.

=====
End End End End End End End End End
=====

The filePro Plus software and the documentation provided with it are protected under United States Copyright Laws and is provided subject to the terms and conditions of the filePro License Agreement.

PLEASE NOTE the support and fax phone numbers listed in this readme file. Open new support incidents on our website.

WWW http://www.fpotech.com
Support support@fpotech.com
Sales sales@fpotech.com
Management filepro@fpotech.com

To submit bug reports

1. Login to your account portal on our website <http://www.fpotech.com/fpotech/login.php> and then go to the Support Incident Menu and submit an incident request.
2. Email them to support@fpotech.com including the text "Bug Report" with the version # and your filePro License # in the subject line
3. FAX them to (813) 354-2722 clearly marking them as bug reports and be sure to reference your filePro License #
4. Call the customer support number (800) 847-4740

A special thank you to Jim Asman for his contribution to the functionality of our printer tables. Jim was a good friend to filePro and is dearly missed.

Contact Information

Surface Mail

fP Technologies, Inc.
432 W. Gypsy Lane Road
Bowling Green, OH 43402

Phones

Support (800) 847-4740
Sales (800) 847-4740
Fax (813) 354-2722

Email

Support support@fpotech.com
Sales sales@fpotech.com
Management filepro@fpotech.com

It's important that you clearly describe a suspected bug and include the filePro version number. If the programmer has trouble figuring out what you meant, you might as well not have reported the bug. Be very specific. For example, if you are reporting a bug concerning a Browse, identify if it is a lookup browse or browse created by using the [F6] key. A screen shot is very helpful and sometimes better than more than 1000 words.

Describe exactly how to duplicate the bug. Although it's sometimes difficult to create a working sample to demonstrate the problem, make every effort to trim down your code and provide a working sample application with test data. You may even discover that what you thought to be a bug is due to a coding error or the bug may only occur with lots of data or large processing tables.

Take good notes as to any error messages and under what circumstances the error message is presented. It never hurts to provide more information rather than not enough. This is particularly true when the programmer asks for additional information. Rather than responding with a single sentence, be verbose since this may shed some light on the bug or what you may be doing wrong in your code.

Read what you wrote. Closely read your bug report before submitting to make sure it's clear and complete. If you have listed steps for duplicating the bug in a sample, exercise the sample with the listed steps to make sure you haven't missed a step.

filePro and filePro Plus are registered
trademarks of fP Technologies, Inc.

=====
Bug fixes are below the New Items.
=====

Version 6.0.VV.18 New Items

=====

Updated all programs to no longer require unixODBC by default. unixODBC will now only be required when an ODBC related function is used. If unixODBC is not found when an ODBC function is required, a filePro error will be returned.

Added PFPDFAUTOBREAK=ON (default OFF) to allow PDFs to automatically break pages based off of selected paper type.

Added menu letter to menu script editor.

=====

Version 6.0.VV.RR New USP Only Items

=====

Task #1782 <LINK text="click here" uri="http://fptech.com">

Optional: underline="off"

Optional: color="#0000FF"

There is a problem with the concept of "destination" for in-document links e.g. dest="page4". This would require pre-initializing all required pages before processing the FPML. Therefore that functionality is not included. Only HTTP links are known to work.

Task #1832 You can now use: @wlf<letter*>

ex. @wlfT*

This will apply to any dummy/associated field that begins with 'T'

Overrides any other @wlf*

Task #1833 Added logging to ddefine.

ddefine can now optionally track changes made to filePro file layouts. This includes the name of the file, who changed it, and what fields were changed. Requires a logging configuration file to be added under the ./fp/logs directory named 'ddefine.cfg'. Format of the config file is the same as the servlog.cfg file that comes shipped with filePro.

Example ddefine.cfg:

ROLLING,DEBUG,ddefine.log,60000

Task #1851 xx=FORMERERROR

syntax: xx=FORMERERROR()

returns: errno from last FORM or FORMM command.

e.g. 2=file not found, 13=permission error

Task #1852 Validate menu script before prompting for removal

Task #1853 CABEBACKUP setting to only save changed PRC

Task #1859 Added new option 'C' to F8 Extended Functions for dmoedef to show a list of all print codes on an output format. Selecting an item from the list will jump the editor to it.

Trim #1875 TRIM command to remove spaces

aa=ltrim(fld)

left trim

aa=rtrim(fld)

right trim

aa=trim(fld)

trim both left and right

Task #1879 PFIXGT can now be set in dxmaint F8 options.

This is backwards compatible, so if PFIXGT is still set in config, then it is honored by clerk *if true*. If false, the index header is checked for the flag.

Task #1887 Added additional checks and field verification for ODBC mirroring. This now attempts to handle invalid data in fields.

Task #1888 Added prelim support for MSSQL for ODBC mirroring.

PFDODBCMSSQL=on (default off).

Setting the variable to on changes the internal handling of the SQL connection.

Task #1668 Windows fPTtransfer now will accept wildcards.

Task #1883 A compress-filePro routine

FPPACK

Function:

Remove deleted records from a filePro file, and then (optionally) rebuild all automatic indexes.

Syntax:

FPPACK [filename | -] [-H heading] [-E] [-R] [-X] [-EX] [-C] [-M name | -MD | -MQ mesg | -MA] [-BG] [-BS]

-H "heading" custom title to display in box.

-E don't actually pack the records, just give statistics.

-R rebuild the automatic indexes even if no records were deleted.

-EX skip statistics

-C skip continue and finished prompts

-X skip rebuilding the auto indexes.

-M name qualifier file name to use.
-MD ask for qualifier with default prompt.
-MQ "mesg" ask for qualifier with "mesg" as the prompt.
-MA use all qualified files & main file.
UNIX/XENIX only:
-BG work in the background.
-BS suppress "completed in background" message.

Task #1902 Added various enhancements to PDF engine.
See on-line or ~/fp/docs PDF documentation.

Task #1906 Added optional error message suppression and basic password
auditing to filePro.

PFERRSUPPRESS=ON, default OFF
FFPWAUDIT=ON, default OFF

Password auditing also requires a ./fp/logs/pwaudit.cfg file. Same
structure as servlog.cfg.
Any error that would be sent to mail will still be mailed on
unix/linux based systems.
Errors reported in the background will still be suppressed.
Including the program name.
Invalid password and license errors will still be reported. Password errors omit the filename.
dcabe and rcabe are exempt from the error suppression.

Task #565 READLINE using "-1" for length to force read to EOL

Task #888 show=pkeep retain position if brw format changes
In a browse lookup with "show=pkeep", the browse was
enhanced to allow the position to be retained, even if the
browse format has changed.

Task #1000 -fp *report flag would not report an error if the prc to use
did not exist.

Task #1227 A new function to lock or unlock nbyte bytes of the file
specified by handle.

Task #1303 ddefine will now create new screens the same as dscreen does
instead of just mono.

Task #1306 NEW arguments for OPENDIR
format length
extension length
fullname length

Task #1340 *cabe lookup wizard will now honor PFQUAL and show qualified
indexes

Task #1359 Added new FPML commands to control the appearance of underlines.

Task #1534 new RINSTR, and INSTR now allows negative positions for working
backwards.

Task #1421 New GIadmin that will count GUI (GI or Web) sessions, ease of system
and user configuration files and additional security.

Task #1504 Added PDF syntax as an option for printer maintenance (pmaint)
Windows only

Task #1564 Remote Licensing for GIserver and fileProWeb. Hardware tie-in no
longer required with internet access

Task #1574 Lookup Wizard in cabe now allows long vars as key

Task #1583 Added alias and arrays to F6-D-L display in *cabe

Task #1584 Added SHOWPROGRESS

Task #1592 updated color with new routines and corrected the shell escape
codes.

Task #1613 New variables
CABEBACKUP ON|OFF (on by default)
CABEBACKUPMINS n (minutes between backups)
CABEBACKUPCT n (backup files per process)

Task #1637 Menu maintenance (makemenu) now asks if you wish to remove
an unused menu script if the menu item is not used.

Task #1676 *report now allows one to use .outs from a pathed directory library

Task #1678 SCREEN command can switch fields in a POPUP UPDATE -, provided no screen name is passed to the SCREEN command.

Task #1447 MEMO EDIT now accept maxsize to limit the number of
characters that can be intered into a memo field.
memo NNN edit (row,col,lines,width,startLine,startcol,maxSize)
(Text mode only)

Task #1609 (All)
Added option 7 to dxmaint to clear qualifier

Task #1610 (All)

New -SE *report flag to allow report to edit/save a selection set.

Task #1601 (All)

Added @EXIT label to *clerk processing. This is executed whenever a record is exited or broken out of. Events that trigger this are 'X' while not in update mode, 'BRKY' while not in update mode, and 'exit' in processing. It is the opposite of @entsel, and is the last thing executed when leaving a record. Assignment of real fields is not allowed, this is similar to @once in that the processing that is executed is NOT sitting on a record, but rather record '0'.

Task #1606 (All)

Partial lookup flag added to *cabe lookup wizard.
-O on an exact lookup now does partial key matching. This kills a lookup once the beginning of the key value no longer matches the lookup key value.

Task #1546 (All)

```
BUSYBOX
BUSYBOX "my message"
BUSYBOX("10","10")
BUSYBOX("10","10") "my message"
```

Task #1723 Added PFPFFULLPATH as an enhancement to PDFPOSTPRINT

and added an PFNEWPOSTPRINT alias to name to PDFPOSTPRINT
Added PFPFFULLPATH to augment the filename passed to the post print handler, default ON, this causes the filename passed to the postprint script to contain the full path to the file, not just the file name. Set to OFF to revert to old behaviour. PFPOSTPRINTnnn will now work with normal file destinations. Same rules as the old global PFPOSTPRINT but also supports PDF files.

Task #1639 Added new conditions for searching used for

associated fields only.

Task #1662 PDF printing should now honor the page length set in dmoedef

Task #1691 clerk will now allow a full path to a form when using

the FORM and FORMM command in processing.

Task #1757 User defined functions - Forward declare functions to be used:

```
(function|func) [file.]name([dim|var] var1, [dim|var] var2, ...)
```

e.g.

```
function fplib.showlock(var pid)
function fplib.log(file, line, what)
function somefunc(dim myarray)
```

Call a function:

```
[x=][file.]name(var1, var2, ...)
```

Return a value from a function:

```
return(value)
```

Can pass fields: real, dummy, longvar

Can pass arrays: Alias and system arrays are copied to a non-aliased array. Non-aliased arrays are passed by reference.

Function names must be at least 3 characters in length.

Functions cannot modify values outside of its scope.

Functions do not call automatic processing.

Functions cannot modify real fields.

Functions cannot be called unless it they are declared.

Functions can pass values by reference (changes made to the value will carry back out of the function, only to arrays).

Functions can optionally return a value.

Parameter names must be at least 3 characters in length.

Parameters will be passed to the function using the name they were defined with in the declaration statement.

Environment variables:

```
PFFUNCDBG=(ON|OFF), default OFF.
```

If ON the debugger will be allowed to continue into the function call. If OFF the debugger will skip over user defined functions.
NOTE: Debug statements inside of functions will still be able to be activated. If debug is set inside of a function, it will continue even after the function is left.

Example:

Processing table for fibonacci:

```
If:          ' Declare for future use
Then: function fibonacci(nval)
If:          ' Get the parameter
Then: declare extern nval
If: nval le "1" ' Return the result
Then: return(nval)
If:          ' Return the result
Then: return(fibonacci(nval-"1")+fibonacci(nval-"2"))
```

Usage:

```
If:          ' Declare for future use
Then: function fibonacci(nval)
```

```
If:          ' Call the function
Then: n=fibonacci("9")
If:          ' Display the result
Then: msgbox ""{n      ' Prints "34"
```

Task #1756 EXTERN and GLOBAL arrays
DIM GLOBAL name(size)
DIM EXTERN name

Only non-aliased arrays can be declared GLOBAL/EXTERN.
Functions similar to GLOBAL/EXTERN longvars.

Task #1639 New compare condition for Associated Fields
Added new selection set relational operators:
AEQ - Associated field, all equal
ANE - Associated field, all not equal
ACO - Associated field, all contain
These require ALL components of an associated field to match the
comparison being done, rather than just one of its component fields.

Task #1667 Extended and Short selection will now check edits when leaving the
field, provided that the relationship code is not CO. If the code is CO,
the field will be treated as NOEDIT instead of its defined type.

Task #1721 Print spool initialization move beyond a large chunk of processing
which should remedy most if not all problems with the spool timeout.

Task #1733 NEW XLSX Mark-up language for creating XLSX files.
(see ~\fp\docs\xlsx_docs.pdf or on our website)
https://www.fpotech.com/fpotech/pdf/xlsx_doc.pdf

Task #1749 Rebuild All Indexes on a file. item '8' on the dialog. Note: this
is in the "extended" dialog which shows when a filename is not specified
from the command line. Indexes can be selected individually,
or all (with F7). Press SAVE, and rebuild begins

Task #1751 Ability to EXPLODE or SPLIT import data
Usage:
sz=SPLIT(array, string, delimiter)
array is the array that the data will be placed into
string is the data to split
delimiter is the sequence of characters to split on

NOTE: The array being used must have the size defined for its elements and
cannot be an alias.

Task #1754 Added the ability to show record locks from *clerk. Can also be
used to terminate sessions directly. New option !L added to *clerk. Using
!L will activate the new locked records list. Enter on a selected entry
will give additional options to the user, including the ability to Kill
or Terminate a locked process without having to go to the command line.
Note: This option is only available on Unix/Linux/BSD

Task #1761 Added UID mapping to filePro, ddir/dprodir option F5.
This allows for UIDs (User IDs) to be aliased to specific
usernames. In the event that a login account is removed from
your system, this can be used to maintain the link between the
removed login's UID and those stored in filePro, effectively
allowing system variables such as @CB and @UB to be maintained.

Windows Only:

This also has the added benefit of allowing @CB and @UB to
function on Windows by linking a "pseudo" UID to a given
username. These UIDs are automatically generated but can
also be manually added. When a user opens filePro and their
username does not exist in the UID map file, a UID will
be generated for that user. filePro will find the next
available UID in the list, starting from 2000, and assign
it to that username.

On all platforms, UIDs stored in this program must be unique
and in the range 0-65535. Usernames can be duplicated on Unix
and Linux platforms, but must be unique on Windows.

Usernames are case-sensitive on Unix and Linux platforms and
are case-insensitive on Windows platforms.

Environmental Variables:

PFUIDMAP = /path
Alternate filePro UID map file. (Use full path)
Note: Must be set in the environment.

PFUSEUIDMAP = ON
Allows filePro to do UID mapping. Also expands the maximum
username length returned by @CB, @UB, and @ID to 32.
Default: ON

Task #1762 All functions that take a position default to the first character
in a field if not specified.

All "is" functions return "1" for true and "0" for false.

```

x=isalpha(fld [, pos])
    Is the character at the position given a letter?

x=isdigit(fld [, pos])
    Is the character at the position given a number?

x=isalnum(fld [, pos])
    Is the character at the position given a letter or number?

x=isspace(fld [, pos])
    Is the character at the position given a whitespace character?
    ' ', '\t', '\n', '\r', '\v', '\f'

x=islower(fld [, pos])
    Is the character at the position given lowercase?

x=isupper(fld [, pos])
    Is the character at the position given uppercase?

x=isxdigit(fld [, pos])
    Is the character at the position given a hexadecimal character?
    '0'-'9', 'A'-'F'

x=iscntrl(fld [, pos])
    Is the character at the position given a control character?
    ASCII codes 0x00 (nul) - 0x1f (US), and 0x7f (del)

x=isprint(fld [, pos])
    Is the character at the position given a printable character?
    ASCII codes greater than 0x1f (US) not including 0x7f (del)

x=ispunct(fld [, pos])
    Is the character at the position given a punctuation character?

x=isgraph(fld [, pos])
    Is the character at the position given a character with a
    graphical representation? The characters with graphical
    representation are all those characters than can be printed
    (as determined by isprint) except for space.

x=tolower(fld [, pos])
    Return the character at the position given as a lowercase
    character.

x=toupper(fld [, pos])
    Return the character at the position given as an uppercase
    character.

str=strtolower(fld)
    Return the entire string converted to lowercase.

str=strtoupper(fld)
    Return the entire string converted to uppercase.

```

Task #1767 Corrected a bug caused by #1691. FORM/FORMM should now corectly produce output when not fully pathing to an output format.

Task #1773 Added new array size function to get the size of an array. Can be used with GLOBAL, EXTERN, LOCAL, and SYSTEM arrays.

```

x=ARRAYSIZE(array)
    Where array is the name of the array.
    Where x is the returned size of the passed array.

```

Tsk #1774 Added new DECLARED function to check if an array or longvar is defined, meaning it is either declared LOCAL or GLOBAL or is declared EXTERN but has a matcing GLOBAL definition.

```

x=DECLARED(var)
    Where var is either a longvar or an array.
    Where x is the return value.
    Returns 0 if the variable is not fully defined.
    Returns 1 if the variable is fully defined.

```

Task #1784 Increased ACTION length in debugger from 60 characters to full 128. Should now be the same as *cabe.

Task #1818 Corrected an issue where a duplicate field warning could display while defining a lookup. Warning is now suppressed until saving the processing table.

Task #1829 Added new flag -DM to [dr]clerk to disable the Index Mode prompt from @ENTSEL. Only works when not in update mode.

Task #1835 Added flag -RH to report to disable the automatic record number reporting in the middle of the screen. This enables placing text on the center of the screen without it being overwritten when the display updates.

```

Task #1836
x=@GUI.PAUSE()
    Pauses automatic screen updating while in GI/Web.

```

x=@GUI.RESUME()
Resumes automatic screen updating while in GI/Web.

Task #1936 REPLACE() enhancement - allow null characters
Enhanced REPLACE() to accept null characters

Task #1938 New flags -EX and -C for fppack
New flags -EX and -C for fppack.
-EX : skip statistics
-C : skip continue and finished prompts

Task #1880 FORM WITHPROC
FORM WITHPROC "formname"
FORMM WITHPROC "formname"

Added additional command switch to FORM and FORMM commands to allow the associated processing table to run while in input processing.

Note: You cannot call the WITHPROC variant from within another form UNLESS the calling form is a processing only form.

Task #1882 Addqual Program
Addqual allows you to easily add qualifiers to your files either interactively or through the command line.

This runs interactively:
addqual [filename]

This runs automatically:
addqual filename -q <qualname>
as does this:
addqual filename -q <qualname> -x <qual-to-copy-from>

The automatic commands will display graphics on errors. You can keep graphics off with "-s" and errors will be printed on the command line if they occur.
example:
addqual filename -q <qualname> -s

List of switches:
-q qualifier to create
-x qualifier to copy indexes from
-s silent, no graphics
-h --help syntax help

Task #1777 XFER - encrypted transfers server-peer

Task #1732 CABE F6 list files from F8 L-Load

=====
END OF NEW USP ITEMS
=====
6.0.XX.23 Bug Fixes
=====

Corrected an issue where an ODBC table wouldn't report as existing.

Corrected an issue with Find and Replace in dcabe/rcabe where control codes were incorrectly being interpreted as character and color codes.

Corrected an issue in dreport/rreport where exiting with an EXIT action in @DONE would not use the supplied exit value if zero records were selected.

Updated F6-D-L listing in dcabe/rcabe to restore typedown behaviour.

Fixed an issue in rcabe, dcabe, and dclerk where a syntax error could cause a crash upon saving or continuing.

Fixed a show map (F6 - View Fields) regression in dcabe/rcabe where button labels were missing valid options.

Fixed an issue with the bottom screen display on Windows in dscreen.

Reworked the SPLIT() command to work with all standard array types.

Fixed SPLIT() return value. There was an issue where the number of elements returned could have been more than the returned type allowed.

SPLIT() now clears the array before updating values and will now honor the edit type(s) of the array, rather than always treating the data as a string.

Fixed a crash when using READOUTPUT() in processing.

Corrected a memory leak in LISTBOX() and SELECTBOX().

Corrected an issue with the "!L" escape in dclerk/rcclerk. Previously, the calculation to determine the locked record number was incorrect.

Added an additional check when saving a record in dclerk/rcclerk so that a zero length must-fill field will be treated as filled.

Corrected an issue with fields resetting when using user defined browses in clerk when a screen contains scrolling fields.

=====
Version 6.0.XX.22 bug fixes
=====

Fixed UID import feature in ddir where it wasn't finding any files.

User can now save a blank UID map in ddir.

Corrected a lock issue with UID maps.

Added additional error messages when importing files for UID mapping.

Fixed an issue with VARCHAR fields not working when using some ODBC drivers.

Corrected an issue with script cleanup causing a crash in dmakemenu.

Fixed a crash when adding an index to an existing empty file in ddefine.

Fixed a crash in fppack when rebuilding an index containing system controlled fields.

Corrected an issue in fpsql where viewing a file's layout would not retain the previous selection.

Fixed an issue in all runtime programs where aliased real fields in an array would not explicitly write on end.

Corrected a potential crash when adding a duplicate key to an index.

Fixed a crash caused by inserting a new unique key after a very long chain of duplicate keys to an index.

Fixed a break key issue in caber F6 label lookup in F9 search. filePro was requiring twice as many break key presses than was actually required.

=====
Version 6.0.XX.21 bug fixes
=====

Corrected an issue on Linux/BSD where fuzzy search could cause a crash.

Fixed array handling in user defined functions that could cause a crash.

Updated fuzzy search to better handle long fields.

Fixed a crash in find and replace in caber.

Fixed a crash when copying lines that don't exist in the file in caber.

Corrected various files not being copied correctly in fpcopy.

Fixed indexes on qualified files in fpcopy.

Added sanity check to locked records check in clerk.

SPLIT() - Removed restriction on delimiter size. Size of array elements still need to be defined for destination.

Fixed syntax error line reporting in caber when jumping to a different place in a prc file. Caber now shows the line number in the editor correctly.

Corrected an issue loading tokenized global arrays in rclerk and rreport.

=====
Version 6.0.XX.20 bug fixes
=====

Corrected an issue when using the Rebuild Indexes option in dxmaint where options were not toggling correctly.

Fixed a regression where scrollable fields in a popup weren't displaying correctly.

Fixed message boxes to better handle filePro escape codes.

Fixed -pv flag and print to screen to no longer corrupt the output.

Fixed alternate automatic processing loading in caber, preventing variables from resolving correctly during syntax check.

Fixed a too many open files bug in fpcopy when working on a file with many qualifiers and indexes.

Corrected fppack to correctly handle encrypted files.

Changed index rebuild message location on the screen to no longer be hidden behind the progress updates.

=====
Version 6.0.XX.19 bug fixes
=====

Fixed an issue where libodbc would not correctly be found when initializing features that use ODBC.

Corrected an issue with RINSTR() where the starting position wasn't honored

correctly.

Option 'C' to clear selection set in [dr]clerk will no longer cause an infinite loop when going back into index selection.

Updated PNG support for PDF outputs. Previously, some PNG files would appear corrupted when imported.

Corrected a potential crash when moving/reordering blob fields inside of dmoedef.

Added PFOLDCHAIN to allow CHAIN to return to the top of processing when a record is saved and the chain was performed inside of an event.

Updated listbox and selectbox code to no longer go outside of screen bounds.

Fixed date handling in XLSX generation when not using the datetime functions.

Fixed an issue where blobs/memos could become corrupted if assigning to the field more than once without writing the record.

=====

Version 6.0.XX.18 bug fixes

=====

Corrected Memory fault in FPSQL

Corrected licinfo to read license fallback file.

Corrected memory leaks in [dr]clerk and [dr]report.

Corrected issue where a select or list box would not clear correctly from the screen.

Fixed positioning and moving objects (memo) on a form.

Corrected button text in F6 cabé.

Fixed an early error exit condition in ddir to report an error rather than exiting.

Corrected "stair step" issue in cabé when using the -C flag on Linux/Unix.

Corrected a crash in clerk when using F5 to duplicate fields between records.

Updated F5 duplicate key in clerk to work with scrolling fields.

Added PFREUSEADDR=ON (default ON) to enable a port to be rebound more quickly when using sockets.

Added code to prevent a dummy field from being used as a foreign key when performing a fuzzy search.

Corrected and reverted wildcard behaviour during selection in clerk.

Corrected type checking for associated fields in selection sets.

Added buttons to clerk fuzzy search for scrolling the file map.

Increased the number of fields shown in fuzzy search in clerk.

Fixed some button shifting for F6 key in cabé.

=====

Version 6.0.XX.17 bug fixes

=====

Task #1948 Autosave not honoring config flags

Corrected an issue where Autosave was not correctly reading config variables. Added initial change backup.

Task #1950 Scrolling fields in popups break placement

Corrected an issue when drawing a popup that contains a scrolling field.

Task #1951 Enhanced runtime format for WHEN flags

Enhanced runtime format to support extended WHEN flags.

Added support for @WUKx* @WHPx* and @WBLx*. New WHEN values will be ignored in older versions of filePro.

=====

Version 6.0.XX.16 bug fixes

=====

Task #1928 XFER too many open files

Corrected prc_backups handling in xfer.

Task #1939 fppack BG/BS flag fix

Corrected the run in background feature for fppack.

=====

Version 6.0.XX.15 bug fixes

=====

Task #1918 PDF print MAP not working

Task #1916 SCO cabé not loading file List properly

Task #1917 Font Size issue with Windows 11 terminal.

Task #1914 Spellcheck crash when using personal list.

Task #1913 Corrected an issue where merge labels (IMPORT/EXPORT) were not correctly indicating that the merge was closed.

Task #1891 Enhanced [dr]clerk to honor a passed index flag for browse when @ONCE or @MENU is used in processing.
PFNEWIXS=on (default off)

Task #1871 Updated dxmaint to keep the user in the index list when working on automatic indexes, saving the position in the list between operations.

Task #1844 Corrected an issue where non-text data could show up as NULL in an ODBC mirror. Requires mirror to be resynchronized to update missing field information.

Task #1845 Corrected an issue in an ODBC mirror where non-concise data types were being used for synchronizing data. This fix prevents TIME and DATE fields from resolving to the non-concise data type of DATETIME.

Task #1849 Checksum error not being reported in outputs

Task #1857 dprodir crashed on non-filePro file when deleting key/data

Task #1865 (rcabe) Corrected an issue where processing tables containing user defined functions would fail to tokenize properly.

Task #1866 Limit outfiles.xml to only hidden outputs

Task #1870 Fixed a potential crash when providing an invalid field number to a lookup expression in the debugger.

Task #1873 SPLIT function not working with leading spaces

Task #1892 Corrected -D flag handling in [dr]clerk to suppress the correct bottom of screen messages.

Task #1901 dmoedef @b4, @c4, @t4, @u4 are now allowed in sort selection.

Task #1864 Corrected an issue on Windows where an alternate auto table set in cabre could not start with the word "auto"

Task #1869 Moved the auto cursor path options in dscreen to the normal cursor path screen under a new F8 option.

Task #1893 Corrected an issue with user defined function argument name parsing. Names that begin with the same sequence of characters will now correctly be allowed, i.e. "col" and "color".

Task #1908 Corrected an issue where the field position wasn't being kept between the header and data sections when editing browse formats.

Task #1911 Corrected a missing initializer value related to index searching that prevented the index search from being performed.

Task #1872 (Linux) Show Locked Records crash
From IUA prompt !L to show locked records.
Tested 25 locked records. They were correctly sorted by record number. Terminated all records one-by-one without incident.

=====
Version 6.0.XX.14 bug fixes
=====

Task #1839 Was not previously documented for 6.0.XX.13
Corrected an issue from a previous bugfix to prevent oversized scrollable fields from being dropped from the end of the screen if the field definition was too long to fit on the screen.

Task #1841
Fixed a crash in dmakemenu when a menu option name started with "BLOB" or "MEMO".

Task #1842
Corrected a bug due to task #1579. Reverted the task to allow correct expression parsing in the processing engine. Restores the use of cm, cw, cx, cy, and cz in expressions.

=====
Version 6.0.XX.13 bug fixes
=====

Task #1763
Fixed an issue where switching between menus in dmakemenu on a bad password could cause a crash.

Task #1764

Fixed an issue where switching between menus in dmakemenu on a bad password could cause a crash.

Task #1765

Changed [dr]cabe to not exit if an edit used in automatic processing and used in input processing was not found.

Task #1768

Corrected an issue with the lookup wizard in [dr]cabe when defining a fuzzy lookup. The browse line would not be generated correctly when leaving the wizard.

Task #1769

Fixed an issue where using '[' instead of '(' in parameter lists could cause an invalid syntax error.
e.g. x=listbox[array] vs x=listbox(array)

Task #1785

Increased ACTION length in debugger from 60 characters to full 128. Should now be the same as *cabe.

Task #1786

Changes caused by a duplicate task #1691 broke the pathed version of the FORM command added by #1722. Reverted changes.

Task #1779

CLEARB now correctly ignored in drop processing

Task #1781

Fixed an issue with SIGPIPE and user commands. SIGPIPE is now handled like in 5.8 unless PFCATCHSIGPIPE is set to ON in the environment.

Task #1784

Increased ACTION length in debugger from 60 characters to full 128. Should now be the same as *cabe.

Task #1790

Corrected an issue where 'killing' a file using dprodir wouldn't remove the directory and its children on Windows.

Task #1791

Corrected an issue with fpcopy that would prevent the prc_backups folder from being properly handled.

Task #1793

fpcopy will now correctly handle all extents on a file and will now mirror them correctly.

Task #1794

Fixed an issue where @GUI.xxx commands could run multiple times (once per break level) in reports.

Task #1795

Updated mirroring to correctly handle multiple extents on a file.

Task #1796

Corrected an issue where upon turning on mirroring for a file and re-executing the dialog option could cause ddefine to crash.

Task #1797

Fixed an issue in GIservr where connecting with an unsupported client (version 7.12.1.6 or earlier) could cause the server to deadlock and stop accepting connections (SCO).

Task #1798

Corrected an issue where all files, when first mirrored, would be created as empty blob files (blob binary data and structures in the file). All files are now correctly created of the right type.

Task #1799

Corrected an issue in ddefine where a broken mirror would not be displayed as such (missing splash screen and 'B' flag in options).

Task #1800

Fixed an issue where when switching a mirror from a broken state 'B' flag in options in ddefine to a 'Y' would not re-create and update missing files.

Task #1801

Corrected an issue in ddefine when trying to create qualifiers. The list would not display unless also creating an index, it now correctly displays without having other options flagged.

Task #1802

Fixed a file handle leak in GIservr that could result in a Too Many Open files error, hanging or crashing the server.

Task #1803

Mirroring disabled warning will now correctly keep its own state between files. Previously, switching between two "broken" mirrors would cause the splash screen to show again.

Task #1804

Corrected an issue in dprodir where filePro could crash if there were a large number

of files in the directory to be deleted.

Task #1805

Corrected index scanning in *clerk when both a selection set and index are specified on the command line. The scan will now also accept system variables in the selection set, such as @PM, to be used in conjunction with the index from the command line.

Note: Using both flags implies PFIIXS.

Task #1806

Fixed an issue in case when the screen was redrawn by scrolling the page. If the previous field allowed for a lookup to be created and the destination field did not, the prompt would still be displayed.

Task #1807

Corrected the "Remove Script" feature when redefining a menu. Previously, if the option had a flag such as '@', it would be included in the path of the file to be removed.

Note: Raw output is still not displayed.

Task #1808 (Web only)

Corrected the wait flag on a menu option in web to display a message rather than a blank screen.

Task #1809

fpcopy will now copy all files under a filePro file's directory structure.

Task #1816

Corrected and added a syntax error to commands that parse screen names as literals, i.e. SCREEN, SWITCHTO, etc. The command would not throw an error on screen names over 1 character if it was not quoted and would parse them incorrectly.

Task #1817

Corrected an issue with field switching in GUI under report. Fields will now correctly switch when clicked on.

Task #1818

Corrected an issue where a duplicate field warning could display while defining a lookup. Warning is now suppressed until saving the processing table.

Task #1820

Corrected an issue with ddefine where having PFQUAL set could cause issues if the qualifier does not yet exist in the file while defining indexes.

Task #1824

Corrected a crash in rclerk when using a POPUP due to a stack underflow. Issue introduced in 6.0.01.06.

Task #1828

Corrected an issue where screens created with ddefine sometimes would have incorrect colors added to the bottom of the screen.

Also updated the screens generated by ddefine from 3.0 to 4.0+.

Added environmental variable:

PFDEFECOLOR=ON|OFF (default ON)

When on, ddefine will create color screens, when off, monochrome.

Task #1830

Corrected the free space checks on all operating systems for ddefine and dexpand. Drive free space is now correctly accounted for. Some space is still reserved in the check for the operating system.

Task #1831

Corrected an issue where drives set in PFDSK in the format of PFDSK=C;D;E;F on Windows would not correctly be parsed, leading to some programs failing to find other drives.

Task #1835 Added flag -RH to report to disable the automatic record number reporting in the middle of the screen. This enables placing text on the center of the screen without it being overwritten when the display updates.

Task #1836

x=@GUI.PAUSE()

Pauses automatic screen updating while in GI/Web.

x=@GUI.RESUME()

Resumes automatic screen updating while in GI/Web.

Task #1838

Corrected an issue when entering an invalid printer type in pmain. Break key will now correctly be honored when answering 'N' or breaking out of the question asking if you want to pick a valid print code table from a list.

Task #1838

Corrected an issue when entering an invalid printer type in pmain. Break key will now correctly be honored when answering 'N' or breaking out of the question asking if you want to pick a valid print code table from a list.

Task #1840

Corrected an issue where reports and forms generated in [dr]report were initialized twice, causing two init print codes to be inserted into the output.

=====
Version 6.0.XX.12 bug fixes
=====

Task #1008 (All) **NEW**

New xcabe program to allow Run-time quick start compiles. There is no user interface except to press ENTER if an error is encountered.

Task # 1699 (All)

Fixed MEMO text to ignore spaces after an ENTER

Task #1712 (Nix)

Updated "finish" script to honor PFLICFILE.

Task #1723 (All)

Added an alias to PDFPOSTPRINT (PFNEWPOSTPRINT), default OFF. Added FPPPFULLPATH to augment the filename passed to the post print handler, default ON, this causes the filename passed to the postprint script to contain the full path to the file, not just the file name. Set to OFF to revert to old behaviour. PFPOSTPRINTnnn will now work with normal file destinations. Same rules as the old global PFPOSTPRINT but also supports PDF files.

Task #1728 (GI)

Reworded "Server Serial Number" prompt when adding a new server entry to GAdmin to "filePro Serial Number" to avoid confusion.

Task #1750 (All)

XFER was not properly copying 'backup' prc files.

Task #1758 (All)

NEW command OPENDIR2 to handle long-named files and paths.
e.x.
N = OPENDIR2(mask, path, fmt_sz, ext_sz, nam_sz)
All arguments are optional.

Task #1776 (All)

If you had an environment or config variable named XXSOMEVAR it would still qualify as PFSOMEVAR.

Task #1764

Fixed an issue where CO in a selection set would not properly check if the value was contained in a field.

Task #1766 (Windows) XFER: sending files on Windows is very slow

Speed improvements for XFER on Windows

Task #1770 Readline EOL not removing CR

=====
Version 6.0.XX.11 bug fixes
=====

Task #1738 (All)

Reversed task 1737 and added new @DV for the distribution release number.

Task #1740 (All)

PFCHECKLOCK was causing an improper error on read-only MEMO Show

Task #1741 (All)

MEMO CLEAR was not working with a lookup memo when reference by the lookup handle

Task #1742 (All)

MEMO functions were causing sporadic memory leaks.

Task #1743 (All)

Modifying a lookup that has an Auto-Index selection set would cause a DKNF error in certain cases.

Task #1748 (Windows)

Uppercase FP in menu command causes comspec error

Task #1611 (All)

Change in behavior for CContains in Long and Short Selection.
Any CContains no longer worries about length or edit in its search.
IMPORTANT: Any indexes previous built with selection criteria in prior version MUST BE REBUILT!!!

Task #1647 (All)

*clerk index selection was missing the correct record in some cases when the index had a selection set.

Task #1739 (Windows)

GAdmin shortcut was not always created during install.

Task #1752 (All)

Fixed a bug in ODBC Mirroring.

Task #1753 (All)

Added PFCALLDBG, default ON, that can be used to have the debugger

ignore calls when turned OFF.
Note: DEBUG ON will still enable the debugger in a call.

Task #1744 Cabe autosave not checking if prc_backups is a directory.
Fixed an issue where cabe wouldn't check to make sure that prc_backups was a directory. It now correctly checks and removes the file if it's invalid.

Task #1747 (Windows) pmain crashes on F6 in printer selection list on Windows.
Corrected a bug where pressing F6 twice in pmain to select a printer would cause a crash.

=====
Version 6.0.XX.10 bug fixes
=====

Task #1721 (All)
Improved Spooler timeout handling in order to avoid spooler shutdown during large printing operations.

Task #1722 (All)
Fixed the ability to use path and filename on FORM

Task #1729 (Web)
Fixed the sequence in which @VR was being cleared.

Task #1730 (All)
Fixed a bug in pkeep by reverting Task #888 which will be readdressed in a future update

Task #1731 (All)
Enhanced transfer speed of fPTransfer

Task #1734 (All)
Fixed a bug where assigning to a real field from a mid statement could blank the field if the field passed to mid was the same.

Task #1735 (All)
A truncated field on a screen was not allowed in cursor pathing in some situations.

Task #1736 (All)
Fixed and enhanced the @VR system variable

Task #1737 (All)
Added PFZEROLENWARN=OFF (default ON) to disable cannot assign to zero length field message when saving a processing table.

Task #1778 (All)
ddefine was not creating qualified indexes

=====
Version 6.0.XX.09 bug fixes
=====

Task #1706 (All)
Setting PFQUAL in *cabe broke the lookup wizard

Task #1710 (All)
runmenu could crash loading some older menus

Task #1711 (NIX)
Some OS errors were bypassing the signal handler

Task #1713 (All)
*cabe was ignoring the new -Cxx flags

Task #1715 (SCO)
'BREAK' in dscreen was crashing

Task #1718 (All)
POPUP was not display the full screen

Task #1719 (All)
POPUP UPDATE was crashing when r,c was used

Task #1720 (All)
Added Error Message if browse lookup closed in DROP processing

Task #1726 (All)
Some commands were seGving on long or invalid filenames

Task #1727 (NIX)
Removed the need for libfpodbc.so by statically linking it the executables.

Task #1727 (NIX) Remove libfpodbc.so requirement
Removed the need for libfpodbc.so on all *nix platforms.

=====
Version 6.0.XX.08 bug fixes

=====

Task #908 (All)

A repeat fix of a previous bug in prc=
checking of a browse lookup

Task #1058 (Nix)

Adjusted finish script for a bug in handling
PFLICFILE value

Task #1693 (All)

String parsing on chmod, mkdir, etc. caused syntax
errors/warnings. This was caused do a preprocessor running the
same syntax routines as the actual syntax checker/tokenizer.

Task #1694 (All)

Corrected a case where some bitwise operators (~^ ~| ~& ~> ~<)
would cause a buffer overrun or return a signed value incorrectly.

Task #1695 (All)

PUSHKEY is now correctly ignored when running processing in
debugging mode.

Task #1696 (All)

FLUSHKEY does not clear keystrokes added by PUSHKEY.

Task #1375 (All)

Updated errmsg file with latest error codes

Task #1675 (All)

F5 would not properly fill a truncated field

Task #1705 (All)

WORDWRAP broke when correcting another MEMO issue. It
now properly breaks.

Task #1704 (All)

Cursor path editing would not properly ignore protected fields.

=====

Version 6.0.XX.06 bug fixes

=====

Task #1673 (OpenSuSE)

GIserver install was failing to create service

Task #1674 (All)

Added AUTO to clarify description in dmoedef

Task #1681 (All)

Syntax check pointed to wrong line on a
duplicate long variable declarartion

Task #1682 (All)

ftp message show/hide flags were reversed

Task #1667 (All)

Modified Extended Selection so that edit does not allows
apply. i.e. Using C0tains for a part of a phone number.

Task #1679 (All)

Modified record selection to accept only numbers.
Allowing other input selected incorrect records.

Task #1672 (All)

The changes were to IXSEL (new optional 3rd parameter)
and IXSORT (new optional 4th parameter). The parameter must be
a path representing an alternate PFDIR to use temporarily for
the function.
IXSORT and IXSEL were failing on qualified indexes

Task #1117 (All)

Improved license related shared memory error reporting.

Task #1687 (All)

Fixed a bug where having a field defined at the end of a row
would not resolve properly if the following line started with
text instead of a space.

Task #1631 (All)

Memo text edit window was not scrolling in certain circumstances

=====

Version 6.0.00.05 bug fixes

=====

Task #1624 (Windows)

Added Desktop Shortcut for GIadmin 6.0

Task #1656 (All)

Archive did not handle lookup to lookup

Task #1657 (All)

Creating a new scree required two Saves to save and exit

Task #1658 (All)
Cursor positionin was wrong after using search in the config editor

Task #1659 (All)
Search & Replace in cabe caused a line truncation when answering NOT

Task #1661 (All)
Index in *clerk is going to the incorrect record when built off of multiple fields. Related to a bad fix from #1503

Task #1663 (All)
Improper key buffer management when using certain combinations of PUSHKY

Task #1664 (All)
Memo memory fault when using certain sequences of Page Up and Page Down in memo editor

=====
Version 6.0.00.04 bug fixes
=====

Task #1652 (All)
When editing a defined lookup in processing, the highlight was on the wrong index.

Task #1653 (All)
Define processing was not reporting a correct line number on a syntax error.

Task #1654 (All)
Define processing was crashing on Search & Replace on a new table that had not yet been saved.

Task #1655 (All)
JSFILE was passing syntax when a function had not been sepcified

Task #1656 (All)
Enhanced ARCHIVE command to work with 2 lookups
ARCHIVE lul TO lu2

=====
Version 6.0.00.03 bug fixes
=====

Task #1651 (All)
Corrected a change introduced into ddefine that caused the map file to be truncated on the last section of the header. This caused ddefine and other developer programs to treat the file as if it had a creation password.

=====
Version 6.0.00.02 bug fixes
=====

Task #1626 (All)
Using a merge field from import fails on condition line would fail.

Task #1628 (All)
Bug occurred in an index when a record was added or deleted from an automatic index using a selection set through a lookup.

Task #1629 (All)
In certain situations the screen text would scroll a screens content and message to wrap in F6 Browse function.

Task #1633 (All)
Added an error message to cabe if code assigns a value to an invalid field.

Task #1636 (GI)
A previous fix for CRON functions broke MSGBOX in GI.

Task #1625 (All)
SORTARRAY now puts any blank fields at the end of the sort.

Task #1634 (All)
Corrected text on setting selection set passwords

Task #1635 (SCO)
Help display was improperly wrapping.

Task #1638 (All)
Field length for description was extended from 80 to 255

Task #1640 (All)
Under certain conditions clearing the BUSYBOX display

would not properly redisplay the screen contents.

- Task #1641 (All)
Enhanced fPTransfer to properly xfer the auto index selection criteria
- Task #1642 (All)
ddefine create screen 0 caused a *clerk invalid screen error
- Task #1643 (All)
Removed debug display of field numbers from IUA index selection
- Task #1644 (All)
dmoedef was not displaying all print codes
- Task #1632 (All)
32bit Installs had the wrong 'rename' program.
- Task #1645 (All)
fPCopy was not properly copying the Auto Index Selection Sets
- Task #1646 (Linux)
Certain colors and highlights were not being shown properly.
- Task #1648 (Linux)
Under some situations a *clerk screen would shift improperly
- Task #1649 (All)
ddefine did not properly make an index with a selection set
- Task #1650 (All)
fPCopy would crash or give an error message when copying a mirrored file.

=====
Version 6.0.00.01 bug fixes
=====

- Task #1607 (All)
Removed double F7 label
- Task #1608 (All)
dosetforms - replace toggle with select all / select none
- Task #1386 (All)
Display when index is built on selection set.
- Task #1619 (All)
Fixed changing Broken mirror status to Yes mirrored status in define files.
- Task #1620 (NIX)
Correct error to redirected stdin

=====
Version 6.0.00.00 bug fixes
=====

- Task #1128 (GI)
F6 was not working from pmain to select Windows printers.
- Task #1165 (All)
Better error handling when indexes are in a known need to rebuild state.
- Task #1239 (All)
MySQL/ODBC was returning a -1 error on longblobs. The problem was caused by the precision being reported as -1.
- Task #1255 (GI)
GI now traps no password in user.cfg file
- Task #1310 (GI)
GIserver now handles HD Serial Number with or without dashes
- Task #1407 (All)
xx = @odbcexception.clear crashes filePro
- Task #1571 (GI)
Occasional broken lines were drawn
- Task #1569 (GI)
GIserver was missing some shutdown messages

=====
End End End End End End End End End
=====

The filePro Plus software and the documentation provided with it are protected under United States Copyright Laws and is provided subject to the terms and conditions of the filePro License Agreement.

PLEASE NOTE the support and fax phone numbers listed in this readme file. Open new support incidents on our website.

WWW http://www.fpotech.com
Support support@fpotech.com
Sales sales@fpotech.com
Management filepro@fpotech.com

To submit bug reports

1. Login to your account portal on our website <http://www.fpotech.com/fpotech/login.php> and then go to the Support Incident Menu and submit an incident request.
2. Email them to support@fpotech.com including the text "Bug Report" with the version # and your filePro License # in the subject line
3. FAX them to (813) 354-2722 clearly marking them as bug reports and be sure to reference your filePro License #
4. Call the customer support number (800) 847-4740

A special thank you to Jim Asman for his contribution to the functionality of our printer tables. Jim was a good friend to filePro and is dearly missed.

Contact Information

Surface Mail

fP Technologies, Inc.
432 W. Gypsy Lane Road
Bowling Green, OH 43402

Phones

Support (800) 847-4740
Sales (800) 847-4740
Fax (813) 354-2722

Email

Support support@fpotech.com
Sales sales@fpotech.com
Management filepro@fpotech.com

It's important that you clearly describe a suspected bug and include the filePro version number. If the programmer has trouble figuring out what you meant, you might as well not have reported the bug. Be very specific. For example, if you are reporting a bug concerning a Browse, identify if it is a lookup browse or browse created by using the [F6] key. A screen shot is very helpful and sometimes better than more than 1000 words.

Describe exactly how to duplicate the bug. Although it's sometimes difficult to create a working sample to demonstrate the problem, make every effort to trim down your code and provide a working sample application with test data. You may even discover that what you thought to be a bug is due to a coding error or the bug may only occur with lots of data or large processing tables.

Take good notes as to any error messages and under what circumstances the error message is presented. It never hurts to provide more information rather than not enough. This is particularly true when the programmer asks for additional information. Rather than responding with a single sentence, be verbose since this may shed some light on the bug or what you may be doing wrong in your code.

Read what you wrote. Closely read your bug report before submitting to make sure it's clear and complete. If you have listed steps for duplicating the bug in a sample, exercise the sample with the listed steps to make sure you haven't missed a step.

filePro and filePro Plus are registered
trademarks of fP Technologies, Inc.

=====
Bug fixes are below the New Items.
=====

New in 5.8.03.XX Update Subscription Program only

New #50 (All)
Printers can be flagged to eliminate them from the -PQ selection listing. See PFPQEXCLUDE

Two new commands in *cabe processing, mode(path) and group(path).
Mode() will return the octal permission mask on a file.
Group() returns a string containing "owner:group" on the file

debug now will accept long variables as break points.

New interface that now allows managing up to 99 printers.

New for Define processing:
Search and Replace
Show all @labels
Go to @labels

ddefine will now use the version 4.5+ dxmaint interface when making indexes for new files

Password protection of .sel and .brw formats

C - Change cursor path you can now press F5 to view the screen in define screens

Line & Box drawing in PDF

replace() command will return a search and replace string of data from either a field or variable

Browsing printers in Options for output formats, filePro now shows only valid printers

New Env Setting PFSEMTIMEOUT. Watchdog code added to the session count code in filePro to prevent semaphore lockups. The value defaults to 3 seconds before it will unlock a broken semaphore. A value of 0 disables the new timeout.

New in 5.8.02.XX Update Subscription Program only

New SELECTBOX() function.

New Form Filtering for IUA(clerk) Form command.

Ability to define screens for *clerk with "truncated/scrolling" fields.

Added timeout option to user, and related TIMEOUT() test.

PDF Enhancement to for MARGINT|MARGINB|MARGINL|MARGINR="margin"

New in 5.8.01.XX Update Subscription Program only

New #1312
Automatic indexes can now be built using a selection set. As records are created or modified, they will be placed in the index only if they pass the selection criteria.

New
Dual Write or Mirroring. Extended options in define files (ddefine) now has an option to mirror a file. Use of PFDIR2 and PFDATA2 settings determine the path of the mirror directory.

New #1324
PFPQ=ON Acts as if the "-PQ" flag was passed to *clerk/*report.

END OF NEW USP ITEMS

Version 5.8.XX.36 bug fixes

Task #1922 Add option to ignore too many edits error
Added new variable to ignore "Too many edits" error message.
PFIGNTMEDS, default OFF

Task #1926 fpcopy -RP flag not working
Corrected an issue where -RP (1|2|3) on a fpcopy command was ignored.

Task #1927 Longvar with no type/size crash in rclerk
Corrected an issue where an untyped local long named variable could cause a crash when inside of a tok file called multiple subsequent times.

Task #1929 PDF incorrectly adding header to grand total page
PDF printing will no longer incorrectly add a header to a grand total

page.

Task #1930 Updated license file format
Corrected some security issues with licensing.

Requires a new download of a license file starting with version 5.8.03.36

Make sure to download a new license file when upgrading.

Updated fplmserver and fplmservice will still authenticate previous versions of filePro. New versions can no longer use the old fplmserver/fplmservice.

Task #1931 Demand/4.1 Auto Index name truncation fix
Corrected a buffer overflow on demand/4.1 auto indexes on large file names.

Task #1932 Prevent crash when using a lookup that hasn't been opened
Added changes to prevent a crash when attempting to write to a lookup that hasn't been opened yet.

Task #1933 MULTI export corrupted
export MULTI will no longer produce an incorrect export file when fields are used in order.

Task #1934 Deleted key not found fix in report
Updated filePro to prevent writing to record 0. This should resolve most DKNF issues.

Task #1935 pmaint insert mode not working
Corrected support in pmaint to allow for the use of insert mode.

Task #1937 File name not showing in ddefine
Corrected display issue in ddefine that prevented the current file name from being displayed.

Task #1940 Correct SCREEN command when using a single character screen
Corrected a change that broke the handling of single character screen names.

Task #1941 FPSQL string handling/overlap issues
Corrected a memory bug in FPSQL that prevented queries from being processed correctly.

=====
Version 5.8.XX.35 bug fixes
=====

Task #1915 [dr]clerk -xi and -xs can cause a crash when used together
Corrected an issue preventing a selection set and an index from being set on [dr]clerk at the same time.

Task #1918 PDF print MAP not working

Task #1917 Font Size issue with Windows 11 terminal.

Task #1914 Spellcheck crash when using personal list.

Task #1913 Corrected an issue where merge labels (IMPORT/EXPORT) were not correctly indicating that the merge was closed.

Task #1891 Enhanced [dr]clerk to honor a passed index flag for browse when @ONCE or @MENU is used in processing.
PFNEWIXS=on (default off)

Enhanced [dr]clerk to honor a passed index flag for browse when @ONCE or @MENU is used in processing.
PFNEWIXS=on (default off)

Task #1907
New PRC inherits previous tables password

Task #1904
EOF marker bit not displaying correctly on Windows

Task #1860
PFAUTOGOTOWARN not honored during runtime

Task #1863
fPCopy failed on copy of files with certain permissions

Task #1903
Corrected an issue where fields would only show as one character in *clerk. Issue only with non-USP licenses in 5.8.

Task #1886
Reverted and updated SCREEN command syntax checking to allow code using undocumented syntax to still function.

Task #1889
Corrected an off by one error in the SELECTBOX position handling.

Task #1890
Corrected runmenu to correctly display the menu item
description color.

Task #1894
Corrected a crash when using an expression as a browse
lookup key field and breaking out.

Task #1895
Removed leftover debug code for GI button handling.

Task #1898
Corrected an issue where the map file could be parsed
incorrectly or cause a crash in ddefine if it ended in
an additional blank line.

=====
Version 5.8.XX.34 bug fixes
=====

Task #1764
Fixed an issue where CO in a selection set would not properly check if the
value was contained in a field.

Task #1765
Changed [dr]cabe to not exit if an edit used in automatic processing and
used in input processing was not found.

Task #1779
CLEARB now correctly ignored in drop processing

Task #1793
fpcopy will now correctly handle all extents on a file and will now mirror
them correctly.

Task #1788
Corrected an issue where a null entry in the outfiles.xml file could prevent
access to a filePro file or cause a crash.

Task #1795
Updated mirroring to correctly handle multiple extents on a file.

Task #1796
Corrected an issue where upon turning on mirroring for a file and reexecuting the
dialog option could cause ddefine to crash.

Task #1798
Corrected an issue where all files, when first mirrored, would be created as
empty blob files (blob binary data and structures in the file). All files are now
correctly created of the right type.

Task #1799
Corrected an issue in ddefine where a broken mirror would not be displayed
as such (missing splash screen and 'B' flag in options).

Task #1800
Fixed an issue where when switching a mirror from a broken state 'B' flag in
options in ddefine to a 'Y' would not re-create and update missing files.

Task #1801
Corrected an issue in ddefine when trying to create qualifiers. The list
would not display unless also creating an index, it now correctly displays without
having other options flagged.

Task #1803
Mirroring disabled warning will now correctly keep its own state between
files. Previously, switching between two "broken" mirrors would cause the splash
screen to show again.

Task #1804
Corrected an issue in dprodir where filePro could crash if there were a large
number of files in the directory to be deleted.

Task #1805
Corrected index scanning in *clerk when both a selection set and index are
specified on the command line. The scan will now also accept system variables in the
selection set, such as @PM, to be used in conjunction with the index from the command
line. Note: Using both flags implies PFIIXS.

Task #1806
Fixed an issue in cabe when the screen was redrawn by scrolling the page. If
the previous field allowed for a lookup to be created and the destination field
did not, the prompt would still be displayed.

Task #1809
fpcopy will now copy all files under a filePro file's directory structure.

Task #1813
Fixed an issue where loading a call in dclerk could cause an invalid syntax
error to be thrown. This was caused by leftover binary data when switching
contexts.

- Task #1814
Fixed a crash in dmoedef when editing multiple reports in a row. The issue could be triggered by changing the printer associated with a report under F8-O for options and was caused by mishandling of the break key stack.
- Task #1816
Corrected and added a syntax error to commands that parse screen names as literals, i.e. SCREEN, SWITCHTO, etc. The command would not throw an error on screen names over 1 character if it was not quoted and would parse them incorrectly.
- Task #1817
Corrected an issue with field switching in GUI under report. Fields will now correctly switch when clicked on.
- Task #1820
Corrected an issue with ddefine where having PFQUAL set could cause issues if the qualifier does not yet exist in the file while defining indexes.
- Task #1830
Corrected the free space checks on all operating systems for ddefine and dexpand. Drive free space is now correctly accounted for. Some space is still reserved in the check for the operating system.
- Task #1831
Corrected an issue where drives set in PFDSK in the format of PFDSK=C;D;E;F on Windows would not correctly be parsed, leading to some programs failing to find other drives.
- Task #1838
Corrected an issue when entering an invalid printer type in pmain. Break key will now correctly be honored when answering 'N' or breaking out of the question asking if you want to pick a valid print code table from a list.
- Task #1839
Corrected an issue from a previous bugfix to prevent oversized scrollable fields from being dropped from the end of the screen if the field definition was too long to fit on the screen.

=====
Version 5.8.XX.33 bug fixes
=====

- Task #1699 (All)
Fixed MEMO text to ignore spaces after an ENTER
- Task #1776 (All)
If you had an environment or config variable named XXSOMEVAR it would still qualify as PFSOMEVAR.
- Task #1778 (All)
ddefine was not creating qualified indexes

=====
Version 5.8.XX.32 bug fixes
=====

- Task #1738 (All)
Reversed task 1737 and added new @DV for the distribution release number.
- Task #1740 (All)
PFCHECKLOCK was causing an improper error on read-only MEMO Show
- Task #1741 (All)
MEMO CLEAR was not working with a lookup memo when reference by the lookup handle
- Task #1742 (All)
MEMO functions were causing sporadic memory leaks.
- Task #1743 (All)
Modifying a lookup that has an Auto-Index selection set would cause a DKNF error in certain cases.
- Task #1748 (Windows)
Uppercase FP in menu command causes comspec error

=====
Version 5.8.XX.31 bug fixes
=====

- Task #1721 (All)
Improved Spooler timeout handling in order to avoid spooler shutdown during large printing operations.
- Task #1734 (All)
Fixed a bug where assigning to a real field from a mid statement could blank the field if the field passed to mid was the same.
- Task #1735 (All)
A truncated field on a screen was not allowed in cursor pathing in some situations.

Task #1736 (All)
Fixed and enhanced the @VR system variable

Task #1737 (All)
Added PFZEROLENWARN=OFF (default ON) to disable cannot assign to zero length field message when saving a processing table.

=====
Version 5.8.XX.30 bug fixes
=====

Task #1718 (All)
POPUP was not display the full screen

Task #1720 (All)
Added Error Message if browse lookup closed in DROP processing

=====
Version 5.8.XX.30 bug fixes
=====

Task #1696 (All)
FLUSHKEY does not clear keystrokes added by PUSHKEY.

Task #1375 (All)
Updated errmsg file with latest error codes

Task #1675 (All)
F5 would not properly fill a truncated field

=====
Version 5.8.XX.29 bug fixes
=====

Task #1672 (All)
The changes were to IXSEL (new optional 3rd parameter) and IXSORT (new optional 4th parameter). The parameter must be a path representing an alternate PFDIR to use temporarily for the function. IXSORT and IXSEL were failing on qualified indexes

Task #1117 (All)
Improved license related shared memory error reporting.

Task #1687 (All)
Fixed a bug where having a field defined at the end of a row would not resolve properly if the following line started with text instead of a space.

Task #1631 (All)
Memo text edit window was not scrolling in certain circumstances

=====
Version 5.8.XX.28 bug fixes
=====

Task #1663 (All)
Improper key buffer management when using certain combinations of PUSHKY

Task #1664 (All)
Memo memory fault when using certain sequences of Page Up and Page Down in memo editor

=====
Version 5.8.XX.27 bug fixes
=====

Task #1652 (All)
When editing a defined lookup in processing, the highlight was on the wrong index.

Task #1653 (All)
Define processing was not reporting a correct line number on a syntax error.

Task #1654 (All)
Define processing was crashing on Search & Replace on a new table that had not yet been saved.

Task #1655 (All)
JSFILE was passing syntax when a function had not been sepcified

=====
Version 5.8.XX.26 bug fixes
=====

Task #1651 (All)
Corrected a change introduced into ddefine that caused the map file to be truncated on the last section of the header. This

caused ddefine and other developer programs to treat the file as if it had a creation password.

=====
5.8.XX.25 Bug Fixes
=====

- Task #1626 (All)
Using a merge field from import fails on condition line would fail.
- Task #1628 (All)
Bug occurred in an index when a record was added or deleted from an automatic index using a selection set through a lookup.
IMPORTANT: Any indexes previous built with selection criteria in prior version MUST BE REBUILT!!!
- Task #1633 (All)
Added an error message to cabc if code assigns a value to an invalid field.
- Task #1636 (GI)
A previous fix for CRON functions broke MSGBOX in GI.
- Task #1634 (All)
Corrected text on setting selection set passwords
- Task #1638 (All)
Field length for description was extended from 80 to 255
- Task #1641 (All)
Enhanced fPTransfer to properly xfer the auto index selection criteria
- Task #1643 (All)
Removed debug display of field numbers from IUA index selection
- Task #1632 (All)
32bit Installs had the wrong 'rename' program.
- Task #1645 (All)
fPCopy was not properly copying the Auto Index Selection Sets
- Task #1650 (All)
fPCopy would crash or give an error message when copying a mirrored file.

=====
5.8.XX.24 Bug Fixes
=====

- Task #1580 (GI)
DEV programs will not run under GI/WEB clients
- Task #1581 (All)
Fixed configuration loading to support old ownership values.
- Task #1614 (All)
Invalid field assignment error pointed to the wrong prc.
- Task #1615 (All)
Removed Natural Order option and bug on adding TAB's
- Task #1617 (Linux)
dossetforms was causing a SegV on certain files
- Task #1618 (All)
Increased the maximum number of open files to 512

=====
5.8.XX.23 Bug Fixes
=====

- Task # 82 (All)
Locking a prc table for editing was not properly working and releasing when encountered.
- Task # 1163 (All)
In some instances a variable was not properly setting when importing in to a memo field.
- Task # 1313 (All)
A new system variable, @VR, was implemented to holding the major version of the full version of filePro.
Task # 1604 (All)
Under some conditions, a file table was filling up due to a mirrored file not being closed properly.

=====
5.8.XX.22 Bug Fixes
=====

- Task # 1597 (All)
Demand index may have displayed strange behavior because

of an improper 64 bit flag.

Task # 1598 (All)
BLOB import was not working in some cases.

Task # 1599 (GI)
Occasional bleed through on certain help file displays.

=====
5.8.XX.21 Bug Fixes
=====

Task # 1586 (All)
Fixed syntax error for PRINTER NAME (exp) command when (exp) was invalid, it now correctly sets to default printer (on prompt) on invalid printer.

Task # 1442 (Windows)
Rewrote the pipe routine and put some windows specific code inside of a new library.

Task # 1582 (All)
Adjusted F5 Lookup Wizard so that browse windows and show windows were the same size.

Task # 1588 (All)
Fixed display on Index Selection on multiple field indexes.

Task # 1590 (All)
Fixed an error on re-browse if selected record is deleted.

=====
5.8.XX.20 Bug Fixes
=====

Task #1556 (All)
cabe when called from dmeodef was not properly exiting if the process was locked by another session.

Task #1575 (All)
MEMO editor would allow a blank line unless INSERT was ON

Task #1576 (All)
Dmoedef was not properly allowing and saving the positioning coordinates and on occasion could corrupt the .out header

Task #1577 (All)
dscreen was not properly toggling F9 Graphics mode

Task #1578 (All)
cabe would lock up on multiple pages of dummy variables

=====
5.8.XX.19 Bug Fixes
=====

Task #526 (GI)
Strip \r from menu title feed to GI

Task #1128 (GI)
F6 in pmain destination field on Windows was not working

Task #1163 (All)
Executing a import memo to a variable was inadvertently storing the memo in the blob file without any record pointer

Task #1552 (Windows)
Customer reported that PDF was not properly printing lines on a form and it was determined that PFPDFFONTSIZE and PFFONT_COURIER was not set properly

Task #1548 (SCO)
Permissions on a system call to run MUTT was not properly set

Task #1549 (All)
dxmaint was honoring PFMBTO on an index delete confirming prompt

Task #1550 (All)
spelledit would crash on list selection listing

Task #1551 (All)
Fixed a bug in spelledit.exe that happened when there were no user spelling dictionary files
Task #1553 (All)
PFMBTO was not being ignored by the MEMO editor F8 dialog box

Task #1554 (All)
WRITE handle in processing was inadvertently writing the main record to disk

Task #1556 (All)

Process lock warning in caber would only allow you to press
Y to continue

Task #1558 (All)
LOOKUP WIZARD in caber would crash on a mirrored file if the
mirror path was invalid

Task #1560 (All)
Incomplete blank printers were being saved in pmaint

Task #1561 (All)
Removing a blank printer in pmaint would prompt for a
confirmation

Task #1562 (All)
F3 and F4 keys were not working in all field columns

Task #1563 (All)
Sometimes pmaint would improperly sort printer in the config
file causing clerk and report not to find out of order printers

Task #1570 (GI)
Yes or No prompts below printer #20 were not working
properly in GI

=====
5.8.XX.18 Bug Fixes
=====

Task #1543 (All)
Adding a comment to an 4.1 style index was not properly saving
a comment and sort criteria or switching the style to 4.5

Task #1542 (All)
Editing the config file could cause clerk or report to not
properly read the contents on startup.

Task #1544 (All)
F8 Options to save but not rebuild was causing an issue with
old style 4.1 indexes

Task #1547 (All)
Printer maintenance could go out of sync with Insert if more than
one page of printers.

=====
5.8.XX.17 Bug Fixes
=====

Task #691 (All)
fileProODBC bug - dprodir fails with fileProODBC

Task #1525 (All)
PFMBTO was being ignored

Task #1349 (Linux)
lockinfo Linux displays effective UID - needs real UID
Two fixes: you must have the latest 5.8.03.17 and the latest
lock.info. You must have PFRROOTFIX set.

Task #1531 (All)
X-Exit not displaying if Index X is hidden

Task #1404 (All)
Using declared variable will not allow call function

Task #1416 (All)
Under some circumstances a Free record lookup would
crashes

Task #1417 (All)
Under some circumstances a report run in GI would hang

Task #1423 (All)
Previous bug fix was not done in rclerk. Now fixed.

Task #1530 (NIX)
Added a new variable PFSECUREDEBUG to disable !b in dclerk.
Default is OFF, setting it to ON will prevent the escape from
working, similar to PFSCC. Prevents root shell access.

Task #1517 (All)
Alignment check question disable for PDF files.

Task #1526 (All)
Problem caused by a fix last March. Work-around was to use
PFOLDWRITE=ON, but that caused a conflict:
dreport eee -f export_csv -a -ro
Now, PFOLDWRITE is not necessary since pfcB will not be marked
dirty IF offending code is reached from *report and -ro is used.

Task #1532 (All)
Arrows would glitch when moving left to right in some

selection windows

Task #1528 (All)
dmoedef would not properly overwrite existing print codes
when copied and pasted.

Task #1533 (All)
Configuration editor was not working properly after a
record lock error message.

Task #1538 (All)
Configuration editor was not properly locking records.

Task #1540 (All)
Added PFINDEXX variable to allow non-default display of
X-Exit in Index Selection

Task #1541 (All)
FORMM was not leaving the pipe open for more data when
output was to a PDF

=====
5.8.XX.16 Bug Fixes
=====

Task #1522 (All)
freechain segmentation fault in some instances

Task #1501 (NIX)
PFUMASK was not honored by PDF files

=====
5.8.XX.15 Bug Fixes
=====

Task #1519 (All)
Long fields that wrap or are scrolled would cover other fields
in the array for cursor path verification

Task #1511 refix (All)
This previous fix broke variables from holding certain values in
reports

Task #1349 (Linux)
lock.info now shows the real UID instead of the effective UID

Task #1513 (All)
Close fails on a dash lookup and crashes in Windows

Task #1518 (Linux)
fpconfig was improperly drawing the active box around the
selection

=====
5.8.XX.14 Bug Fixes
=====

Task #1503
F7 was not properly going to the last record in the index in clerk.

Task #1505
Fixed a record deletion bug in *clerk (Windows only) where a WRITE
would be reverted/deleted on BRKY.

Task #1507
Fixed a bug with dmoedef, a null character made its way into the shortened
printer list, appended to each name, it caused a strange graphical glitch
that
made it almost useless.

Task #1508
Changed selectbox to be "1" based instead of "0" on screen positioning.
Added PFOLDSELECTBOX to enable "0" based positioning, default OFF.

Task #1509
Fixed dummy fields disappearing off of the screen after a WRITE and END
combo if a record has not been changed. Added PFOLDWRITE to revert to
old method in case of issues, default OFF.

Task #1511
Overflow/segfault in reports, was caused by a scoping issue between
global and local longvar fields.

Task #1512
Enabled spaces in selectbox. Added PFSELBOXSPACE to disable s
paces in selectbox(), default ON.

=====
5.8.XX.13 Bug Fixes
=====

Task #1475 (All)
dscreen locked message was not properly displayed

- Task #705 (All)
 - Using -XI and -XS was never intended or coded. As a result the requested index was ignored in favor of the best fit according to the selection set. Now, if both -XI and -XS are specified the selection scan will use the requested index.
- Task #1483 (All)
 - PDF now honors width and height in tag
- Task #1253 (All)
 - configuration editor now allows selection of config files other than the one currently in use by the one named in PFCONFIG
- Task #1399 (All)
 - In dscreen, F5 resolve fields did not properly show the truncated length of the truncated fields.
- Task #1487 (All)
 - In pmaint there were some issues with Paging and also setting a printer on pages greater than page one improperly removed the printer NAME & TYPE information.
- Task #1489 (All)
 - pmaint will now allow you to enter an invalid printer type after prompting to make sure that is what you want to do.
- Task #1490 (All)
 - Cursor pathing information was not printing and a dscreen hardcopy
- Task #1491 (Linux)
 - Free Space was not properly stated on 64 bit systems
- Task #1492 (All)
 - Added the ability to use flags from the command line when using FPCOPY
- Task #1493 (All)
 - Mirrored index were adding freen chain data to the end of the index. This does not invalidate the index but destorted the file size.
- Task # 1497 (All)
 - Could not remove dummy from cursor path if it did not exist on the screen, also could not arrow/move past it. F4 now removes the field and blanks properly
- Task #1498 (All)
 - LOGTEXT is now limited only by disk space up to 2,147,483,647 bytes.

=====
 5.8.XX.12 Bug Fixes
 =====

- Task #82 (All)
 - rcabe now properly locks a prc when in edit mode
- Task #1477 (All)
 - A dubious printer name in processing would cause a segV
- Task #1476 (All)
 - dcabe was failing on a CLOSE or WRITE command in processing
- Task #1478 (All)
 - dreport could result in a zero length PDF file
- Task #1358 (All)
 - Run-time programs should now give a message when the limits of 205 combined EDITS is reached instead of aborting
- Task #1427 (All)
 - dscreen and ddefine should now format properly when the default printer is PDF
- Task #1482 (NIX)
 - Minor display modifications to dscreen for when used on old Wyse 60 terminals
- Task #1485 (All)
 - BLOB/MEMO fields were crashing when used with tok processing that had not been recompiled.

=====
 5.8.XX.11
 =====

- Task #1472 (All)
 - Define Output - F6 for Printers was offset one line.
- Task #1473 (NIX)
 - pmaint screen did not work properly with wy60 emulation
- Task #1474 (All)

pmaint F9 (go to line #) only had 2 characters

=====
5.8.XX.10
=====

Task #1469 (All)
CREATE / WRITELINE was failing or throwing garbage to the screen

Task #1470 (All)
SHOW POPUP not working

Task #1471 (All)
Encryption Error (Encrypted file / Grace Period Mismatch) when trying to
access an encrypted file.

=====
5.8.XX.09
=====

Task #1308 (All)
Some Development programs were responding to PFMBTO

Task #1334 (All)
Invalid lookups to a mirrored file would give an
incorrect 'not licensed' error

Task #1363 (NIX)
The new fpconfig utility program was not included in
the NIX compiles

Task #1431 (Windows)
dclerk debug F9 Search for \{ displays graphics
instead or previous search criteria

Task #1439 (All)
Development programs were not working properly when
PDF was the default printer

Task #1442 (Windows 64 bit)
USER command was not executing correctly on 64 WIN

Task #1449 (All)
ddefine - when changing 16,memo to 100,allup in map of
mirrored file filePro would crash

Task #1450 (All)
cabe - was missing the O - Options to the F8 options

Task #1451 (Windows 64 bit)
fpdaemon_win.exe on 64 bit never displayed a fileProGI
menu

Task #1454 (All)
clerk - FORM in 5.8.02.08K3 crashed in clerk on
repeated executions

Task #1461 (All)
clerk/report - PDF with overlay was not working
properly

Task #1460 (All)
clerk - Selectbox was not respecting width and height
parameters

Task #1464 (All)
clerk/report - EXPORT WORD -A was not appending to an
existing file

Task #1043 (All)
When a BLOB field was removed in ddefine, the blob
file was not properly removed.

Task #1356 (All)
debug now will accept long variables as break points.
The scope of a longvar is different from a normal
dummy field. Technically, longvar is not at a true
global scope, and isn't available in the automatic
processing table. Declaring it 'g' only will work
across records, but not tables, declaring it GLOBAL
will fix that, but it has to be matched with an EXTERN
in the other prc table.

Task #1409 (All)
The index license ownership display has been
reformated so it does run into the version display.

Task #1423 (All)
popup("7","-1") was not centering properly

Task #1425 (All)
In certain case, there was a stray character at the
end of @ID when running under fileProGI

Task #1441 (All)
The ability to cancel when adding a new screen did not work.

Task #1444 (All)
The check free disk space routine was updated to handle hard drives over the old 32 bit limits.

Task #1118 (All)
Page Up and Page Down was not working in memo edit windows.

Task #1468 (All)
xfer was causing errors in restoring key files.

=====
5.8.XX.08
=====

(All) #1269
fPCopy was not properly handling encrypted files.

(All) #1308
F8 - Options in dmoedef and dscreen were honoring PFMBTO and now ignore it

(All) #1331
ddefine could crash on save of missing a blob file when turning on mirroring.

(NIX) #1363
utility fpconfig was not compiled in some previous releases for NIX platforms.

(All) #1370
Added the prc table name to error messages in most places to help with filepro error identification

(All) #1403 & 1418
F9 was not allowing a search of a prc table in *rcabe and configuration editor

(All) #1411
cosmetic adjustments to the F8 Options screens and popups in dmoedef

(All) #1412
Under certain circumstances the cursor would move outside of the memo edit area

=====
5.8.XX.07
=====

(All) #1174
After saving a screen in Define Screens, it was possible for a session to update the same screen at the same time.

(All) #1377
If you execute a lookup-dash to an existing record in automatic processing while in add records mode, and then cancel out of update, the record would be deleted.

(All) #1384
INPUTPW did not respect the edit type of the input field, and treated everything as "*".

(All) #1385
If you have an automatic index with a selection set, and update the record such that the true/false value of that selection set changes, but none of the key fields for that index change, the index was not updated.

(All) #1388
Pressing F5 (create script file) in Define Menus when there was no menu item keystroke defined yet would create a script named "menuname-.X". Now, you cannot create a script until the keystroke is defined.

(5.8.02 and later) #1392
If the "filecount" value in a file's "outfiles.xml" file was higher than the actual number of files listed, Define Output would crash upon selecting that file name.

(5.8.02 and later) #1393
fpcopy did not copy the "outputfiles.xml" file if it exists.

(Windows only) #1394
If an error occurred in xfer while receiving a file, the error display might show a black area of text, rather than the file name.

(All) #1395

When transferring a filePro file with a "map.new" file that does not match the "map", it was possible for xfer to not properly receive the filePro file.

=====
5.8.XX.06
=====

(All) #1061
If you have two lookups to a file with different aliases, with the second lookup being a browse lookup, if that browse lookup has processing, and within that processing you refer to a field in the first lookup alias, non-"g" dummy fields are cleared.

(All) #1262
If you have a protected lookup to the main file in a CALLED processing table, and CLOSE that lookup, without having ever modified the record, or using WRITE, the looked-up record will remain locked. (If you use GETNEXT/GETPREV on that lookup, then it would be the last record read by the lookup which remains locked.)

(All) #1319
If you are running filePro with standard input either redirected (such as from a file, or via a pipe), or with no standard input at all (such as from cron), and filePro needs input from the user (such as asking for a filename, due to a typo on the command line), filePro would wait in an infinite loop, using 100% of available CPU. filePro will now give a fatal error, and exit.

(All) #1355
If you have an IMPORT/EXPORT statement with the alias "alias", and somewhere in processing you had a reference to "alias()" in a comment, you could get a syntax error on that comment.

(All) #1377
If you execute a lookup-dash to an existing record in automatic processing while in add records mode, and then cancel out of update mode, the record would be deleted.

(5.8.01 &.02) #1385
If you have an automatic index with a selection set, and update the record such that the true/false value of that selection set changes, but none of the key fields for that index change, the index was not updated.

=====
5.8.XX.05
=====

(5.8.XX.04) #1374
Non-Lite versions of filePro failed to allow qualifiers in ddefine

(All) #1373
Non-Lite versions of filePro failed with "syntax error" on ENCRYPT() and DECRYPT().

=====
5.8.XX.04
=====

(All) #1054
Erroneous "feature not licensed" errors correct.

(All) #1313
filePro user menus can now use a pseudo-environment variable "@VR" to place the filePro version on the screen. You can use either Windows ("%VR%") or Unix/Linux ("\${VR}") syntax.

(fileProGI) #1317
filePro GI client displayed "garbage" for the "Enter" prompt.

(All) #1344
A spurious "requested feature not licensed" message would sometimes be included in filePro error messages.

(Quikstart) #1345
If you have a "locked(-)" test without a lookup-dash, rcabe did not give a syntax error.

(Windows 64-bit) #1347
Pressing F8/Options in printer maintenance would cash the 64-bit Windows version of filePro.

(All) #1350
dxmaint -m "" would not override a PFQUAL setting.

(Linux) #1352
filePro now turns off stty "iexten" mode on systems which support it.

(All) #1353
FPML now supports user-defined margins to the tag.

(All) #1357
New FPDFCOMPRESSMODE config variable to control PDF compression.

(Windows) #1360
The Windows version of dexpand didn't properly handle key files larger than 4G in size.

(Windows 64-bit) #1361
When concatenating a string with a memo field from a lookup file, 64-bit Windows filePro could crash.

=====
5.8.XX.03
=====

(All) #402
If you select a record via *clerk's "6 - fuzzy search" menu choice, and then press Break/Del/Ctrl-C, *clerk exits, rather than returning to the menu.

(fPSQL) #1053
fPSQL did not properly display user edits in the F5 field list.

(fileProGI) #1336
When running filePro on a Windows server, the "Enter" prompt displayed "garbage" characters.

(5.8.01.01 only) #1337
It was possible that the first lookup to a file would corrupt indexes built on field you modified.

(All) #1338
The CREATE() function did not respect the umask, always using a umask of 0177 instead. Now it will respect PFUMASK if set.

=====
5.8.XX.02
=====

(All) #402
If you select a record via *clerk's "6 - fuzzy search" menu choice, an then press Break/Del/Ctrl-C, *clerk exits, rather than returning to the *clerk menu.

(64-bit only) #1332
Spellcheck didn't work properly on 64-bit implementations.

(GIclient only) #1336
When running fileProGI against a Windows version of filePro, then "Enter" prompt displayed as garbage.

(5.8.01.01 only) #1337
It was possible that the first lookup to a file could corrupt indexes built on field you modified.

=====
5.8.XX.01
=====

(All) # 1332
Aspell ws replaced with hunspell for the 64 bit version of filePro

(Linux) #1330
The 64-bit Linux version of xfer didn't copy screens correctly.

(Linux) #1327
On 64-bit Linux. closing a lookup file could crash with a corrupt heap. ("*** glibc detected ***" error message.)

(Linux) #1316
Fuzzy browse lookups could crash on 64 bit Linux

(All) #1286
Setting PFIDLEN=32 will cause @ID, @CB, and @UB to have a length of 32 rather than 8. (The only legal values are currently "8" and "32". Any other value is undefined.)

(All) #1295
Alien and ODBC files can't be encrypted by filePro. ddefine will now prevent you from marking such files as encrypted.

(All) #1302
filePro now defaults to PFMAXALLOC=128 and PFMAXASIZE=128000, for up to 16MB of RAM for sorting.

(All) #1304
If you have a fatal error (such as "file not found" on import) in a CALLED table, filePro might crash upon exit.

(All) #1305
If you print a form from *clerk via "F"orn (not from FORM within

processing), and that form's output processing does a lookup to the same file as an open lookup in input/auto processing, and then CLOSEs that lookup, filePro can crash if the input/auto processing re-executes that lookup.

(All) #1314
The FPML documentation was incorrect. The "" tag is documented as taking an "ORIENTATION" attribute, but it should have said "ORIENT". filePro will now allow either spelling.

(Windows 10) #1318
On Windows 10, if you set filePro's TextNormal attribute to a value from 0x80-0xff, there could be "ghost" characters left on the screen.

(Windows, 64-bit) #1321
The 64-bit Windows version of rclerk/rreport could crash when using OPENDIR().

(All) #1322
If you have a very long filename that "looks like" a qualified filename (such as "key [conflict at 2014-12-09_23-28-27]"), filePro could crash.

=====
5.8.00.00
=====

(Linux) #1292
On some Linux systems, filePro would see a different MAC address than reported by "ifconfig" or "ip addr".

(All) #1294
If you edit a prc file outside of filePro, and there is no newline character at the end of the last line, filePro may crash on exit or "switch files" in *clerk.

(All) #1295
Alien files can't be encrypted. Ddefine now prevents you from marking such files as encrypted.

(OSR5) #1297
On SCO OSR5, if you have a fuzzy browse lookup, and pass it a zero-length key, filePro can crash.

(OSR5) #1299
Fuzzy search on SCO OSR5 returned incorrect results.

(Windows) #1301
When using PFXS=ON or the "-jy" flag, it was possible for Windows to throw an "uninitialized variable" error when scanning for records using an indexed field.

(All) #1304
If you have a fatal error (such as "file not found" on import) in a CALLed table, filePro might crash upon exit.

(All) #1305
If you print a form from *clerk via "F"orm (not from FORM within processing), and that form's output processing does a lookup to the same file as an open lookup in input/auto processing, and CLOSEs that lookup, filePro can crash if the input/auto processing re-executes that lookup.

(All) Version 5.8
New function to get error code for ENCRYPT()/DECRYPT() failure.
status = CRYPTERROR([format])
If "format" is omitted, or zero, then the value is returned as a numeric error number, or zero for "no error". If "format" is "1", then the value is returned as a string. Other values for "format" are undefined. (If ENCRYPT/DEcrypt fails, a null string -- "" -- is returned.)

(All) Version 5.8
New function to set the ODBC query timeout:
old = @ODBC.handle.TIMEOUT(timeout)
where "timeout" is the desired timeout in seconds. The function returns the old timeout value, if available, or "" if not.
The default timeout is 15 seconds. Not all ODBC sources allow the timeout to be set. Setting the timeout to zero disables any timeout functionality, and can cause filePro to simply wait forever.
This only affects future queries. Also, some ODBC sources share the timeout between all handles attached to the same ODBC_CONNECTION handle.

=====
End End End End End End End End
=====

5.7.(00,01,02,03,04).07 Release Notes

(Windows) #1226

The Windows version of filepro didn't show the user limit on the Alt-F10 screen.

(Windows) #1271

If you have a large number of large arrays (ie: 96 arrays of 1000 elements each) in processing, filePro may be slow to exit.

(Windows) #1277

Setting FFEOF changes the "Press <--> to continue" prompt. (All) #1278 LOCKED(-) would always test "false".

(SCO OSR5 only) #1279

Running an fPSQL query with a WHERE clause containing a date literal could crash.

(All) #1280

Using the processing debugger, if you use "E" to enter an expression using an invalid field (such as "imp(10)" on a 9-field import), filePro can crash after giving the error message.

(All) #1288

If you have a lookup to the current record in the main file, and MEMO EXPORT a field via the lookup, filePro will complain that you are trying to modify the current record via a lookup.

(All) #1289

When hardcopying a file in ddefine, if the output was not to "PDF:", the heading wasn't printed on the first page.

(*nix) #1290

On some *nix systems, an idle runmenu can wake up every 3 minutes and cause a spike in CPU usage for 1 second. While not usually noticeable, this can be a problem on a system with many users.

(All) #1291

If you execute a DELETE (no lookup name), and then a CALL, the record was not deleted.

(Linux only) #1292

On some Linux systems, filePro would see a different MAC address than reported by "ifconfig" or "ip addr".

5.7.(00,01,02,03,04).06 Release Notes

(All) #1272

When printing to a PDF document, if you have an FPML tag with attribute="value" (using quotes around "value") and "value" is longer than 1024 characters, filePro could crash.

Note that this could also happen if you tried to embed a PCL file which happened to include "just the right sequence" of characters.

(All) #1273

fPSQL would crash on files with MEMO/BLOB fields. Note that such fields are still not yet supported, but fPSQL will no longer crash on such files.

(SCO only) #1274

The HASH() and HMAC_HASH() functions would crash on SCO if passed an invalid hash name.

(SCO only) #1275

There appears to be a bug in the SCO C runtimes library, where the text-to-binary (strtod) function would return "not a number" for valid numbers, in some not-yet-determined scenario. This version of filePro works around it.

This would manifest itself within filePro in the power (^) operator sometimes returning "-NaN".

(*nix only) #1276

License check in 5.7.xx.05 versions of *nix filePro wouldn't work correctly for MAC-based licenses.

5.7.(00,01,02,03,04).05 Release Notes

(All) #1272

When printing to a PDF document, if you have an FPML tag with attribute="value" (using quotes around "value") and "value" is longer than 1042 characters, filePro could crash.

(Windows) #1250

If PFDSK wasn't set, dexpand could fail to check disk space.

(All) #1251

The "grace period" warning wasn't always shown properly when running in the grace period.

(Linux) #1254

Attempting to resolve fields in dscreen on a screen with @UB and/or @CB could crash on Linux.

(Linux) #1264

PFMONO=ON wasn't respected on Linux.

(*nix) #1265

Two new termcap entries -- FL and FM-- represent a second set of keys which can be used to DELC and INSC, respectively.

(Windows) #1267

When sending output to a PDF file, "H"ardcopy in *clerk might not open/print the PDF until you exited.

(All) #1268

If you execute PPRINT @SBRK on the very first record of output, *report might crash.

5.7.(00,01,02,03).04 Release Notes

(All) #1230

Removed starting splash screen and added customer and version to the clerk Index Selection screen.

(All) #1223

DOKEY did not switch screens if passed a digit.

(Windows) #1228

The default location for the PFCHECKLOCK=ON log file is moved from the root directory (which is not writable on current versions of Windows) to the user's HOME directory.

If %HOME% isn't set, filePro uses %HOMEPATH%. If that is not set either, then filePro will use the user's "my documents" directory.

(GI) #1229

When using @GUI.RECVFILE(), if the client gets an error (file not found, permission denied, etc.) or if the file is zero length, filePro would write garbage to the destination file.

(All) #1233

On filePro-MySQL, it was possible to crash filePro with certain combinations of lookup-dash followed by "if: not -" and "if: -".

(All) #1234

Using "if: locked(-)" could crash filePro.

(All) #1235

If you are sitting in an ODBC file, and have a lookup (not lookup dash) to the same file, filePro can crash when the file is closed. (Syntax check in *cabe on such processing could crash as well.)

(All) #1236

If you use one of the PRINTER commands to send the output to a PDF destination, followed by PRINTER RESET, and then come other PRINTER command to set the destination to a non-PDF destination, that output will be "lost", and no output generated.

(All) #1237

If you have a menu command with "@" for wait-on-return, and press Ctrl-C/Del/Break at the "Press enter to continue" prompt, runmenu would exit completely, even if this was a nested menu.

(All) #1241

If you have a memo field which you grow over time, such that the memo becomes fragmented, and then shrink the memo such that the last fragment is no longer needed, the blob file will be in an inconsistent state. If you then grow the memo again, the file will be corrupted such that either (1) filePro will crash when updating that memo, or (2) two memos will be "cross-linked" together.

(ODBC/MySQL) #1242

If you are sitting in an ODBC/MySQL file, and have a (non-dash) indexed lookup into the same file, GETNEXT will always find the record after the one you're sitting on, and not the lookup record.

(ODBC/MySQL) #1243

A browse lookup on an ODBC/MySQL file to itself would not respect the "must match" flag.

5.7.(00,01,02).03 Release Notes

(Quikstart) #1188

If you have a DECLARE GLOBAL variable in input processing, and CHAIN to another prc file, and then CHAIN back to the original input processing, the DECLARE GLOBAL variables would be cleared.

(All) #1204

If you have a processing table created outside of filePro, with a

"then" line longer than *cabe's 127-character limit, *cabe may crash on Windows 8 systems with a stack corruption error when you try to save the processing.

(All) #1205

There is now a "-FC" flag to ddir to delete selection sets.

(All) #1206

If you have a CALL from automatic processing, and that called table closes a lookup which is already open in input (or output) processing, then filePro may crash when that lookup is accessed in input (or output) processing.

(All) #1207

If you have a protected lookup-dash, followed by an unadorned "WRITE", the current record would be unlocked.

(All) #1208

If, in add records mode, you have a lookup to the current file, a DELETE, and a lookup-dash, you can get a "deleted key not found" error on the lookup-dash.

(All) #1209

If you executed an unprotected lookup-dash while in update mode, the resulting record was left unlocked, despite being treated as being in update mode. Such a lookup will now be implicitly locked.

(All) #1211

filePro didn't properly handle deleting an ODBC connection:

ODBC_CONNECTION handle DELETE

if there were open recordsets on that connection. filePro will now implicitly close (but not delete) any recordsets on that connection prior to closing the connection. (The recordset handles will remain valid, but the only operation permitted on such recordsets is to delete them.)

(Windows) #1213

When comparing a date field to a non-date field, it was possible to cause a Windows "debug assertion failed" error in "just the right combination of circumstances".

(All) #1214

If you are adding a new record, and have a lookup-dash to a record which is locked, and press BREAK/DEL while the "waiting for record to be unlocked" message is on the screen, filePro would delete the lookup record, rather than the yet-to-be-created record you were sitting on.

(All) #1215

Attempting to do date arithmetic on a date field containing "/OV" could cause filePro to hang/crash. filePro will now correctly return "/OV" in those cases.

(fileProG) #1216

Under G, ddefine didn't move the cursor into another field if you clicked the mouse.

(All) #1217

If the main file is encrypted, and you have a lookup to the main file, and you then close that lookup, further access to the main file may no longer decrypt/encrypt correctly.

(All) #1218

Task item #1181 caused @SF/@SH on output formats to always be blank. (Note that they still worked in output processing. It was only on the output format that failed.)

(*nix) #1220

Fix for #1127 (restore umask for SYSTEM) caused ddefine to not create new directories with the correct permissions.

(Windows) #1221

5.7.01 (and later) fpcopy didn't properly rename files under Windows.

(All) #1223

DOKEY would not properly handle a number to switch screens.

5.7.01.01 Release Notes

(All) #1175

When using dexpand to pre-expand an encrypted file, the new records could contain garbage.

5.7.00.02 Release Notes

-
-
- (All) #1013
The @WORDWRAP data would include the new line at the end of the last line, even when requesting that new lines be stripped.
- (All) #1058
licinfo did not respect FFLICFILE
- (Windows) #1131
Some Windows systems were unable to read the MAC address for licensing.
- (All) #1145
Grace period splash screen time can now be dismissed by pressing a key.
- (All) #1178
A selection set passed with "-s", which had a "-" in the name, would be truncated at the "-".
- (Windows) #1185
Some Windows systems were unable to read the Windows product key for licensing.
- (Quikstart) #1186
Datemath on 2-digit-year fields which overflowed the 2-digit-year range would result in erroneous values, rather than "/OV". (Quikstart only.)
- (All) #1190
Fixed cosmetic errors when displaying record numbers beyond 99,999,999.
- (All) #1191
Selection sets in files larger than 99,999,999 records which referenced @RN, did not properly select records.
- (All) #1193
INPUT POPUP had an undocumented limit of 70 characters for the input field. If the field was longer than 70, the remaining characters were filled with spaces. This limit has been removed, and the field will now scroll if too wide for the screen.
- (All) #1194
Fields @PM, @PW, @PX, @PY, and @PZ defaulted to the value " ", rather than "".
- (*nix) #1195
The USER command could leave defunct processes after CLOSE
- (All) #1196
@TS is now (10,.0) on files larger than 99,999,999 records.
- (All) #1197
More information will be recorded in the license log file for license failures.
- (All) #1199
If you have an F6-defined browse lookup which requires more than 73 characters to display the data, and then edit the format, the end of the data of the selected record would "bleed" into the format.

5.7.00.01 Release Notes

- (*nix) #1001
filePro didn't properly read free space on some systems with more than 4GB of free space.
- (*nix) #1127
When executing SYSTEM() commands in processing, filePro now restores the original umask value.
- (All) #1130
If you have a protected lookup to the main file, and end up on the same record you are updating (such as via a GETNEXT loop), and modify that record, and then move off the current record, the lookup (ie: the record you are sitting on) will be written and unlocked. This leaves you in update on an unlocked record, with the old data.
filePro will now prevent you from modifying the current record via a lookup while in update mode. (Just as it prevents you from deleting it.)
- (All) #1169
Some edits punctuation that occurred within a literal when that

literal was within the same punctuation, could cause filePro to crash. For example:

```
<"<">
("hello(there")
```

(All) #1170

When restructuring files larger than 4GB, ddefine could hang.

(All) #1171

If you run a report with sort/select processing, but no automatic or output processing, it is possible for *report to crash when it gets to the grand totals.

(All) #1173

When using fuzzy browse mode, if you search on an associated field, it is possible that the field name will be rejected upon returning to the "enter field" screen.

(All) #1179

If you have a protected lookup using an alias, and modify the lookup, and then execute another lookup using the same alias without issuing a WRITE, the previous lookup record will be unlocked before being written. This leaves a small window of opportunity to have another process read and lock the record before the first process writes the new data.

(All) #1180

A browse lookup-dash did not properly handle record locking, either leaving the previous record locked, nor not locking the new record.

(All) #1181

On files with more than 99,999,999 records, @RN was not properly set to 10 characters, causing erroneous record numbers to be reported for record 100,000,000 onwards.

Note the following issue still remains for now; If you enter *clerk on a file with less than 100 million records, and have @RN on the screen, and then start adding records so that the file then contains more than 100 million records, the value of @RN as displayed on the screen will remain at 8 characters until you switch screens. Other references to @RN will properly adjust to 10 characters.

(All) #1182

If you have a file with no automatic or input processing, and use "F-Formt" to print a form that has output processing, then use "7-Change File" to another file with no automatic or input processing, attempting to then update/add a record can crash *clerk.

(All) #1184

A selection set which includes 12 AND/OR items in the sentence can crash filePro.

5.7.01.00 Release Notes

(All)

Encrypted filePro key/data/blob.

(All)

"PFREADONLYWARNING=OFF" turns off all read-only warning messages.

(All) #1169

Some edits punctuation that occurred within a literal, when that literal was within that same punctuation, could cause filePro to crash. For example:

```
<"<">
("hello(there")
```

(All) #1170

When restructuring files larger than 4GB, ddefine could hang.

(All) #1171

If you run a report with sort/select processing, but no automatic or output processing, it is possible for *report to crash when it gets to the grand totals.

5.7.00.00 Release Notes

All platforms

All bug fixes from 5.6.11 and prior revisions are included in this product. Refer to the readme_5.6.11.txt file.

SCO CSR5

Note that on SCO's CSR5, "AF_INET6" is unknown, as our CSR5 development system does not support IPv6 as far as we can tell.

All platforms

If you pass an unknown family to SOCKET(), AF_INET ("IPv4") is used.

If you pass an unknown family to BIND2(), AF_UNSPEC ("any") is used.

All platforms

xx = @ODBCEXCEPTION.CLEAR

This will clear any text currently returned by @ODBCEXCEPTION]

Windows

Windows GIGclient -pq local printer support

All platforms

When using ENCRYPT() and not passing a nonce (ie: allow filePro to generate the nonce), it was possible to generate the same nonce on two calls in a row.

All platforms

The HMS edit in filePro 5.7 now supports times up to 1999999999:59:59, rather than the previous limit of 596523:14:07.

Windows

Under Windows, if a filePro program was run from the task scheduler, and an error occurred, filePro would wait forever for the non-existent user to press Enter to continue.

All platforms

If you have a MEMO EDIT command within @MUK processing, and have the "-d" flag on the command line, the memo editor prompts were left on-screen upon saving the memo.

All platforms

In some cases, if you get a syntax error on a "large" processing table in rcabe, the program will crash upon pressing Enter at the error message. Note, this is rcabe only.

All platforms

If you have an index built on a field with a user edit (ie: not a systemedit), and you add or delete edits prior to the edit definition, dclerk might not respect the edit in the "enter index search data" dialog after the initial scan.

Note that this is typically harmless, but it can interfere with the behavior of right-justified edits.

5.6.11 Release Notes

- (All) # 1162
In rclerk/rreport (only), the functions FIELDNAME(), FIELDEDIT(), FIELDLEN(), and FIELDVAL() would all fail with fatal "reference to a field that doesn't exist" error if given a non-existent field. In dclerk/dreport, such a condition was handled by returning a null string.?
- (All) #1091
If you have a protected lookup in @WBRK processing, and don't modify or WRITE the lookup, the record would remain locked.
- (All) #1093
A selection set comparing a memo field equal to "" doesn't work.
- (All) #1107
DOKEY from a CALLED table would not do anything.
- (All) #1113
If you use a file I/O function in output processing, and assign the result to a dummy field not defined in automatic processing, the I/O will occur multiple times.
- (fileProGI) #1121
Under some circumstances, GI could change the contents of a field on the screen, even if the user never entered that field.
- (Linux) #1122
dmakemenu could crash when pressing DEL/Ctrl-C after using the F5 shell script editor.
- (All) #1123
dmoedef could crash when loading an output format with a sort field higher than any field number in the file.
- (Linux) #1124
F9/search in *cabe wouldn't properly update the display when the user presses Backspace in the search field.
- (*nix) #1132
Improved signal handling, to properly handle things such as closing a window in FacetTerm while filePro is still running.
- (fileProGI) #1133
Under fileProGI, if you use the MENU command with an array, where some elements were never assigned a value, filePro could crash. (For example, DIM the array for 10 elements, and only assign a value to the first 5.)
- (Linux) #1135
ddefine could crash on some Linux systems on some circumstances.
- (fileProGI) #1136
Under fileProGI, if you have PFPT=ON set, and do not have any values set for PFTEMP, TEMP, and TMP, filePro can crash when executing a command with the "-pq" flag.
- (*nix) #1139
showlock didn't display correct information on some 64-bit *nix releases.
- (All) #1150
It was possible to get an erroneous "subscript out of bounds" error when referring to an array which was aliased to dummy fields or an associated field group.
- (All) #1151
On some platforms, it was possible for opendir() to return a list of filenames of the correct length, but which contained some blank entries instead of the names.
- (All) #1152
If you assign a value to a memo field via processing, and then refer to that same memo field before writing the record, the memo field can become corrupted, or cause filePro to crash.
- (All) #1153
When building an old-style index, dmxaint can fail with an "invalid argument" error should the index exceed 2GB in size, even on platforms that support 64-bit I/O.
- (All)
When restructuring a file larger than 2GB in size, ddefine could corrupt the data.
- (All) #1154
ddefine will now update the record counter during restructure only once a second, rather than every 100 records.
- (All) #1155
When calling ENCRYPT() without a nonce, and then calling GETNONCE() to read the value, it is possible for the same nonce to be generated more than once.

- (All) #1140
Under Windows, if a filePro program was run from the task scheduler, and an error occurred, filePro would wait forever for the non-existent user to press Enter to continue.
- (All) #1141
If you have a MEMO EDIT command within @WUK processing, and have the "-d" flag on the command line, the memo editor prompts were left on-screen upon saving the memo.
- (All) #1142
Quikstart didn't properly support dummy fields declared as (16,memo). See also #915.
- (All) #1143
In some cases, if you got a symtax error on a "large" processing table in rcabe, the program would crash upon pressing Enter at the error message. (Note: rcabe only.)
- (fileProGI) #1147
When running under GI, filePro on Windows will now include "local printer" as a destination when using "-pt".
- (All) #1154
When restructuring a file, ddefine/autosshuf will now update the record counter only once a second, rather than every 100 records.
- (All) #1159
Under "just the right combination of conditions", using *cabe's lookup wizard would change the lookup type to "Z-Fuzzy" within the wizard.
- (All) #1161
If you have an index built on a field with a user edit (ie: not a system edit), and you add or delete edits prior to the edit definition, dclerk might not respect the edit in the "enter index search data" dialog after the initial scan.
Note that this is typically harmless, but it can interfere with the behavior of right-justified edits.

5.6.10 Release Notes

(Nix)
Fixed a bug introduced where @sk did not hold special key when exiting a browse.

5.6.09 Release Notes

- (All)
The SYSTEM() function would give a syntax error on some platforms.
- (All) #1063
blobfix now takes command line parameters:
blobfix [filename | -] (-m qualifier)
- (Windows) #1074
fpcopy would give a "cannot create new file" error upon doing a "copy file formats only".
- (Quickstart only) #1104
Executing a LOOKUP in a processing table, CALLing another table which executes a LOOKUP to the same file, and then CLOSEs that lookup, can cause filePro to crash and/or give unusual errors upon re-executing the LOOKUP in the original processing table.
- (All) #1105
The configuration editor did not allow an equals sign to appear in the value of a setting.
- (All) #1106
Under certain circumstances, CLOSEDIR() could cause filePro to hang or crash.
- (All) #1108
A new config variable, PFPRTFIND=logfilename, was added to help debug problems in resolving printer name to destination.
- (All) #1109
In request output, if you execute a CALL in sort/select processing, and then execute a CLOSE in output processing with no intervening CALLs, filePro would crash.
- (All) #????
The SYSTEM() function would give a syntax error on some platforms.
- (All) #596
blobfix did not properly fix qualified blob files.
- (fPSQL) #701
fPSQL queries that selected based on associated fields behaved differently if an index was built on that associated field group.
- (All) #865
Selection sets did not properly compare memo fields no empty. For example, if field 6 is a memo field:
6 ne <nothing>
all recods would be selected, even those where field 6 is empty.
- (All) #1078
fpconfig did not allow values which contained an equals sign ('=').
- (SCO OSR5) #1031
fpcopy on SCO OSR5 did not copy files whose name was longer than 14 characters.
- (fPSQL) #1060
fPSQL on some systems would crash on queries using associated fields.
- (All) #1067
dxmaint did not override PFQUAL of passed -m "" on the command line.
- (All) #1094
A printer with a destination of "LOCAL" was not recognized as setting the destination to the local printer.
- (fPSQL) #1111
fPSQL could not use "filename.fieldname" format if the filename started with a digit.
Note that, with this change, invalid numeric literals such as "123b4" will now give an "unknown field" error rather than an "invalid numeric literal" error.
- (All) #1112
When converting from a non-time field to an HMS field, the conversion would not be correct if the input field were longer than 8 characters, or the hours value was larger than 99, regardless of the length of the destination field.
- (SCO OSR5)
Add a new variable, PFRENEWTIMEOUT=nnn, to control the license renew message timeout on SCO OSR5. The value "nnn" is the number of seconds between renewal messages, and can be any number from 2 to 600. The default is 60.
Note that this variable is unnecessary, and ignored, on other platforms. Connectivity issues on OSR5 (cabling, router, TCP/IP stack, network congestion) can cause unexpected delays in OSR5 filePro programs (some long enough to seem like program hangs). Reducing the value of the license renew timeout messages can reduce and even eliminate these

delays.

(*nix)
The following environment variables will be set within filePro if the license server is not being used:
PFPPID - the process id of the filePro program.
PFSID - the session id of the filePro program.
PFLICENSESUSED - the number of licenses that filePro shows to be currently in use.
PFPROCESSCOUNT - the number of license-counted programs that filePro shows to be currently running this may be higher than the number of licenses used since multiple sessions with the same session id will only use one license count.

(Windows) #1114
FP-SQL would crash when opening an ODBC alien file.

(All) #1115
fPcopy would crash on a very long filename.

(Windows) #1116
If the code:
handle = new odbc_connection(dsn) failed due to a license failure, "handle" would remain unchanged. The typical symptom would be seen as "odbc_conection returned a blank value".

5.6.08 Release Notes

(Windows) #1016 ddefine did not allow filenames with "." in Windows

(Linux) #1018 showlock did not display all locked records Linux

(All) #1026 Only one index was deleted when selecting multiple indexes in ddir/dprodir

(All) #1072 - enhancement GETENV("PFREADONLY") will now return "ON" if the "-ro" flag is used, even if not set in the environment.

(Quickstart) #1075 If you have a label "foo" and another "foo-bar" (ie: the same prefix, with a hyphen and a suffix added), references to "foo-bar" would refer to "foo" if "foo" occurs earlier in the processing. (All) #1068 If you had something following the "b=(expression)" part of a browse lookup which "looks" like part of a valid expression, filePro would take that as part of the "b=(expression)" part of the browse lookup. For example: lookup foo k=xx i=a -nx b=(brw head&brw data) *2" (If, for example, you were to edit the lookup line to remove a literal browse lookup, and missed the trailing "*"2".)

(All) #1021 In dcabe, if you load a processing table with a DECLARE GLOBAL and syntax check it, and then load and syntax check another processing table which has the same DECLARE GLOBAL name, but a different definition, you would get an "already defined" warning.

(All) #1040 EXIT in @ONCE processing didn't exit *report.

(Linux) #974 Website didn't allow license to be downloaded when using Domain Name as the license check value for Linux systems.

(All) #975 The following flags have been added to swappcu: -Q = Quiet mode. Suppresses "file already in destination order". -CN = Convert to native byte order. -CF = Convert to foreign byte order.

(All) #977 swappcu didn't swap the "blob" file.

(All) #982 PFMVTO only caused the first message box to time out. All subsequent message boxes still required Enter.

(Windows) #1016 On Windows, ddefine did not allow you to create or access files with a dot in the filename.

(Linux) #1071 On some systems, a fresh filePro install would fail with "fatal error check current user count: invalid argument".

(All) #1077 filePro 5.6 is now available on FreeBSD.

(All) #1088 Calling READLINE() with an uncast dummy field, which has never been assigned a value, and not including a maximumlength, could crash filePro.

(All) #1099 There was a memory leak in nested CALLS which has been fixed.

(All) #1100 Under some circumstances, filePro would not use the available free space in the blob file for new blobs/memos, causing the blob file to grow larger than needed. Note that the updated blobfix utility can be used to "fix" this, and shrink the blob file to the "correct" size.

(All) #1101 If you have a lookup to a file that has an automatic index built on an associated field group, but the lookup is not on that index, it was possible for filePro to crash.

(All) #1102 Executing a LOOKUP in a processing table, CALLing another table which executes a LOOKUP to the same file, and then CLOSEs that lookup, can cause filePro to crash and/or give unusual errors upon re-executing the LOOKUP in the original processing table.

(Linux) #1103 The Linux distribution included an old "lib/rename" utility, which wouldn't run on some current Linux distributions.

(*nix) #1014 User-count debug logging can now be controlled via the new config variable "FFCHKUSER=logfilename".

(All) xfer now handles the blob file.

(*nix) filePro could crash in some debugging modes of the systemenvironment variable MALLOC_CHECKS.

(*nix) #1070/#1096/#1097/#1098 Some places in filePro didn't properly require double-break.

(*nix) Debug versions hard-coded the user-count logging filename. This is now controlled with FFCHKUSER=logfilename.

(All) OPENDIR() could crash if given a directory with thousands of files, all in alphabetical order.

(All) If a user menu's title and version number filled the space allotted, filePro could crash.

(*nix) On some *nix systems, DEL (when not used as the Break key) would not be treated as a backspace.

5.6.07 Release Notes

(All) #964

ddir/dprodir would display "datax3" rather than "data"

(All) #967

In the memo editor, using the up and down arrows, if the cursor is in a column on the source line beyond the last column of the destination row, the cursor may be placed in the wrong location.

(*nix only) #968

When running without the license manager under Unix/Linux, *report and *cabe would not release the session upon exit.

(All) #969

TAB characters in help files were not properly handled.

(Sun) #1011

Files > 2gb cannot be accessed on SUN OS

(All) #975

Added the following flags to swapcpu:

-Q - Quiet mode. Suppresses "file already in destination order" messages.

-QN - Convert to native byte order.

-CF - Convert to foreign byte order.

(All) #976 swapcpu index already swapped crash segv

Running swapcpu on a file with an index already in destination byte order could crash swapcpu.

(All) #992

The following scenario corrupts the index, possibly giving a DKNF error in the future:

A new key is being inserted into the index. (Or, an existing key is being modified, and the new key value is being inserted.) The leaf node into which the key would be inserted is full, as are both of its sibling nodes. (Meaning a new leaf node gets created by splitting the current node.) The parent node needs to be updated to insert the new leaf node. The problem comes when the parent node, and the parent's right-side sibling are full. (And possibly the left-side sibling as well, in which case we need to continue up to the grandparent node, where the same situation occurs.)

(All) #1042

Blobfix doesn't fix the internal record number reference within the blob header. This does not affect anything at runtime, but does make examining the structure for debugging purposes more difficult.

(All) #1047

If you have more than one memo field in a record, update one of them and then execute a "memo ... delete" on the other one, and the other memo field never had any data in it, the blob file can be corrupted.

(All) #1059

In *report, if you CALL a process from within @VBRK or @WGT, and that processing table DECLAREs new variables, referencing those

variables can cause *report to crash/freeze.

(All) #1064

OPENDIR()'s return value was truncated to the last 3 digits, so if more than 999 files are returned, the return value was wrong. Note, however, that all files were returned.

(All) #966

If you have a lookup file with memo fields, and to a COPY to that lookup, the memo fields won't be copied correctly.

(All) #970

If you have an array aliased to fields which includes a memo field, executing "CLEAR arrayname" will crash filePro later.

(All) #1003

If you defined memo dummy fields or variables in a file which has to "real" memo or blob fields, filePro would crash upon access to those fields.

(All) #1012

If you have a filePro map with empty field names in the middle, any reference-by-name of the first field after the blank field name(s) will not work correctly.

(All) #1017

If you concatenate fields such that the combined length is greater than 32,767 you would get an out-of-memory error.

(All) #1027

In *report, if the first assignment to a dummy field that is not declared in automatic is in a CALLED table, then *report may crash at a reference to a dummy field done within @DONE processing.

(All) #1028

Under certain circumstances, using "@@" in an edit can crash filePro.

(All) #886

Attempting to use sockets without a socket license could crash filePro rather than give an error.

(All) #963

In the memo editor, if the cursor is on the last line of a word-wrapped line of text, and is on the first line within the memo editor window, and the rest of the window is filled with additional text, then placing the cursor at the end of the text on that line and backspacing such that the now-shorter text would re-wordwrap to take less lines, taking the cursor to the text which has been re-wordwrapped to the previous line (ie: the one above the top of the window), can cause filePro to crash.

(All) #1002

If the width of a listbox was at least 78, and dropshadows are turned off, the contents of the listbox are shifted right 2 by columns.

(All) #1048

DECLARE EXTERN in sort/select processing for a variable which is DECLARED GLOBAL within automatic processing could cause a "variable already declared" error after sort/select processing was finished.

5.6.06 Release Notes

(Nx) #968

When running without the license manager, *report and *cabe did not release their session license upon exit.

(Nx) #965

The SCO version of filePro could fail to validate a MAC-address-based license if enough filePro sessions were open. (Even though there were less sessions than the license would allow.)

(All) #966

If you have a lookup file with memo fields, and do a COPY to that lookup, the memos won't be copied correctly.

(All) #963

If the cursor is on the last line of a wordwrapped line of text, and is on the first line within the memo editor window, and the rest of the window is filled with additional text, then placing the cursor at the end of the text on that line and backspacing such that the now-shorter text would re-wrap to take less lines, taking the cursor to the text which has been re-wrapped to the previous line (ie: the one above the top of the window), can cause filePro to crash.

(All) #967

In the memo editor, using the up and down arrows, if the cursor is in a column on the source line beyond the last column of the destination row, the cursor may be placed in the wrong location.

5.6.05 Release Notes

(All) #949

The 2-digit-year system date fields resolved without an EOF marker in dscreen/dmmedef.

(All) #957

LISTBOX() would expand embedded TAB characters within items.

(All) #958

If you are on the last line of text in a memo, and that line wraps, and the end of that line (ie: the end of the memo itself) is visible as the last line in the memo window, and you start deleting text on that line such that the wordwrap now takes up less lines (ie: the word on the last line gets "pulled" up to the previous line), then filePro can crash.

(All) #959

autosshuf didn't run with a runtime-only license.

(All) #960

The default value for PFLISTSLASH has been changed from ON to OFF.

(*nix) #961

Under certain circumstances, the first time you accessed a licensed feature, filePro would freeze while doing the license check. This only happened if the license manager was not running.

5.6.04 Release Notes

(Linux) #940

Using LOCKED() in rclerk/rreport on Linux could crash filePro.

(All) #941

ifPSQL could crash if the query required 10K of output per record. (Default increased to 100K, and it now properly reports error if exceeded.)

(All)

Corrected some licensing issues with session and feature counts not being released in some circumstances.

(ODBC) #943

Define files fails on an ODBC data source when the table name has embedded spaces.

(All) #947

If you have a dummy field which has not been given a length/type in processing, and place it on a screen, and then have a popup screen over that field, and when the popup isn't active (ie: after a POPUP UPDATE, but before a CLEARP, do a SHOW "@...", the contents of that dummy field will bleed through to the popup.

(All) #951

If you have a fuzzy browse lookup with "show =pkeep", filePro would crash upon re-executing the lookup.

(All) #952

New systemarray -- @FUZZY[] -- contains information about the most recent fuzzy browse lookup.

Currently, the only info available is the record number, via @FUZZY.RECNO[subscript]

(All) #954

If you have any pending PUSHKEYed keystrokes, the filePro processing debugger didn't update the screen as you stepped through processing.

(All) #955

-SX flag to *report did not load the selection set.

(All) #956 Memo editor spellcheck crashes

Pressing F8/S in the memo editor to run spellcheck would crash filePro if you hadn't previously done a SPELLCHECK() function within processing.

(*nix)

On Unix/Linux, when not using the license server, if the license file was missing or was determined to be invalid for any reason, the filepro program would abort with a segmentation violation. This has been corrected and the filePro program now displays a licensing error message and exits cleanly.

(*nix)

On Unix/Linux, when not using the license server, filePro could incorrectly create many semaphores and shared memory segments.

5.6.03 Release Notes

(*nix)

fplmsrver wouldn't recover disconnected sockets in a timely manner.

(SCO OSR5)

fplmsrver would freeze when running a filePro program from within a SYSTEM command on SCO OpenServer 5.

(All) #939

PFNUMXBUILD was incorrectly documented as allowing up to 128000. The real maximum is 32767, which is now enforced by filePro. (Setting it higher than the maximum value will result in using the maximum.)

(Linux) #940

Using LOCKED() in rclerk/rreport on Linux could crash filePro.

(All) #941

fPSQL could crash if the query required 10K of output per record. (Default increased to 100K, and it now properly reports error if exceeded.)

(All)

Corrected some licensing issues with session and feature counts not being released in some circumstances.

5.6.02 Release Notes

(All)

PFNUMXBUILD was incorrectly documented as allowing up to 128000. The real maximum is 32767, which is now enforced by filePro. (Setting it higher than 32767 in previous versions could crash filePro.)

(*nix)

Some license server sockets in fplmserver were not being closed and released in a timely fashion.

(OSR5)

Nested sessions of clerk/report could cause an infinite loop in the license server on OpenServer 5.

(*nix)

Problems with the DEL/Ctrl-C interrupt on multi-threaded platforms could cause crashes.

(Linux)

filePro had problems with the shared libtermcap on 64-bit Linux platforms, so we now statically link to our own termcap library.

(All)

@LICENSE[] will now show a feature as licensed, even if the session limit has been reached for that feature.

(All) #924

PFNODFMMSG=OFF turns off ddefine's "PFNODF=ON" notice. (Default: ON)

(*nix) #925

Fixed a problem on *nix boxes with filePro not properly counting sessions if the license manager is not running, and one filePro program calls another.

(5.6) #928

RECV() and RECVLINE() now accept DECLARED variables, in addition to real and dummy fields, as the destination.

(Windows)

Index Maintenance (pmain) now has an F6 option to display a list of system-defined printers while in the "destination" field of the printer config screen.

(All) #927

Fixed a problem with node_addr and DKNF errors on indexes which are larger than 65536 blocks in size.

(5.6) #937

Added PFENTSELDISABLE=list to disable (and remove prompts for) a set of default behavior of *clerk at the "Enter Selection" prompt.

A question mark (?) represents HELP. Note that the HELP key cannot be disabled with this variable, although you can keep the new prompt from appearing.

The default value is "?", which causes the same prompts as before to appear.

Any "invalid" keys listed are ignored.

Note that any @KEY events are not disabled. Only filePro's default behavior for the keystrokes is affected. Also note that, if the HELP key label is longer than 3 characters, enabling all of the prompts will not fit in 80 columns.

(All) #938

MEMO handle EXPORT filename APPEND

BLOB handle EXPORT filename APPEND

The above would crash at runtime.

(All) #931

FIELDNAME/LEVEDIT/VAL didn't take an expression for the field number.

(ODBC) #932

dreport was unable to post new records to some high-level ODBC data sources.

(*nix) #933

If you were to have a variable/array/lookup/etc called "USER", then the parser would allow some invalid syntax, where part of an expression was missing, such as "aa = bb +".

(Linux) #934

There are two distinct, and incompatible, pty methods on different Linux boxes. fpdaemon now includes both methods in a single binary, and determines at runtime which method to use.

(All) #935

dxmaint would show "not available" on indexes that were in use, but it would still let you rebuild them

(All) #936

MEMO...SHOW now accepts the NOWRAP flag, just as MEMO...EDIT does.

(Linux) #926

fpcopy would fail with "cannot create new data file" on some Linux systems.

(*nix) #929

DOKEY would not work if you used a lower-case letter.

5.6.01 Release Notes

(*nix)

Problems with the DEL/Ctrl-C interrupt on multi-threaded platforms could cause crashes.

(Linux)

filePro had problems with the shared libtermcap on 64-bit Linux platforms, so we now statically link to our own termcap library.

(All)

[@LICENSE] will now show a feature as licensed, even if the session limit has been reached for that feature.

5.6.00 Release Notes

- [Data encryption supports various encryption standards.](#)

Blowfish, Twofish, AES, Rijndel, DES, Safer+, and Rc2

`result = ENCRYPT/DECRYPT(data,method,key [,nonce])`

- [64-bit file I/O breaks the 2 GB file size limit](#)

- [Nested Calls](#)

Calls within processing can now be nested, limited only by system resources

- [Windows UNC Support](#)

Store your data at \\machine\sharename

- [SPELLCHECK command](#)

Check your text memos or fields for spelling accuracy Custom dictionaries by user login

- [26 Automatic Indexes](#)

Now you can have 26 automatic indexes A-Z

- [Socket Access commands](#)

(Requires additional purchase & annual license fee)

Open and monitor TCP/IP ports

Pass information back and forth through the ports

- [License Protection](#)

System calls are excluded from the license user count.

"Free" evaluation software are available with expiration dates.

- [Increased the number of extended key/data segments from 3 to 6](#)

Functions Added

`RESET @FN`

`GETPID()` returns the process ID of the current process.

`GETPPID()` returns the process ID of the parent process under Unix/Linux. On Windows, this information is not available, and an empty string is returned.

`GET16/GET32/PUT16/PUT32`

`value = GET16(buffer [,offset [,byteorder]])`

`value = GET32(buffer [,offset [,byteorder]])`

`binval = PUT16(value [,byteorder])`

`binval = PUT32(value [,byteorder])`

The `GETnn` functions retrieve a binary value from a buffer, and the `PUTnn` functions convert the value into binary. The offset parameter specifies the zero-relative offset within the buffer where the value resides. The byteorder parameter specifies the byte order of the binary value, with "L" meaning little-endian, "B" meaning big-endian, and the default being the native order of the current system.

For example, to get the 32-bit little-endian value at offset 8 within the `MyBuffer` variable, you would use:

`GET32(MyBuffer,"8","L")`

To convert the `RouterHandle` variable into a 16-bit little-endian value, you would use:

`PUT16(RouterHandle,"L")`

Note that 8-bit values can already be read/written using the existing `ASC` and `CHR` functions.

`FIELDNUM()`

`FIELDNUM(lookupname,fieldname)` returns the field number of the given field name. If no such field exists, a null string is returned.

`WORDWRAP()/@WORDWRAP[]`

Used to manually do word-wrapping of fields.

`xx = WORDWRAP(field,width)`

`xx = @WORDWRAP[linenum]`

The `WORDWRAP()` function generates wrap information for the given field, as wrapped to the given width. It returns the number of lines that result. (The field need not be a memo field.)

The `@WORDWRAP[]` array returns information about the most recent `WORDWRAP()` call. It is an array of zero-relative values containing the offsets for the start of each line. Note that some values will be negative, meaning that the line break was caused by a hard return in the buffer just before it. (Also note that the hard return is still in the buffer and would need to be excluded from any printout.)

`@LICENSE[]`

Contains license information.

Subscripts:

- 1 = A comma-separated list of licensed features.
- 2 = Name
- 3 = Serial number
- 4 = Platform
- 5 = Product name
- 6 = Version
- 7 = User count

DOKEY

The new DOKEY function allows @key to trap a keystroke and then tell filePro to act on that keystroke as if where were no @key trap. Its syntax is:

DOKEY expr

where "expr" is the keystroke to perform. Note that only the first character is used, and it need not necessarily be the same as the current @key event.

Note that DOKEY does an implicit END.

For example:

@keyX

Then: msgbox "Are you sure you want to exit?","","YN"

If: @bk = "Y"

Then: dokey "X"

Then: msgbox "Exit cancelled"

Then: end

@keyF

If: UserCanPrint ne "Y"

Then: errorbox "You are not allowed to print forms!"; end

Then: dokey "F"

Also note that DOKEY does not interfere with FUSHKEY, and they can be used in conjunction. For example, to trap "B" and then load the "mybrowse" browse format:

@keyB

Then: pushkey "f|mybrowse[ENTR][ENTR]"; dokey "B"

@SBRKn event

"Start of break" event.

This event is triggered on the first record of a subtotal break. Unlike @WBRKn, this processing is run prior to the normal processing, and if multiple @SBRKns are triggered at the same time, they are executed from the outermost level inward. (ie: the opposite order of @WBRKn processing.)

SYSTEM() function

xx = SYSTEM(command [,noredraw flag])

Equivalent to SYSTEM and SYSTEMNOREDRAW, except that the exit value is returned.

If "noredraw flag" is zero or not supplied, the screen is redrawn after executing the command. If "noredraw flag" is "1", the screen is not redrawn (the equivalent of a SYSTEMNOREDRAW command).

Other values are not currently defined.

Note: See 5.0.15 Release Notes for bug fixes since 5.0.14

5.0.15 Release Notes

(All) #966

If you have a lookup file with memo fields, and do a COPY to that lookup, the memos won't be copied correctly.

(All) #963

If the cursor is on the last line of a word wrapped line of text, and is on the first line within the memo editor window, and the rest of the window is filled with additional text, then placing the cursor at the end of the text on that line and backspacing such that the now-shorter text would re-wrap to take less lines, taking the cursor to the text which has been re-wrapped to the previous line (ie: the one above the top of the window), can cause filePro to crash.

(All) #967

In the memo editor, using the up and down arrows, if the cursor is in a column on the source line beyond the last column of the destination row, the cursor may be placed in the wrong location.

(All) #949

The 2-digit-year systemdate fields resolved without an EOF marker in dscreen/dmroedef.

(All) #957

LISTBOX() would expand embedded TAB characters within items.

(All) #958

If you are on the last line of text in a memo, and that line wraps, and the end of that line (ie: the end of the memo itself) is visible as the last line in the memo window, and you start deleting text on that line such that the word wrap now takes up less lines (ie: the word on the last line gets "pulled" up to the previous line), then filePro can crash.

(All) #960

The default value for PFLISTSLASH has been changed from ON to OFF.

(Linux) #940

Using LOCKED() in rclerk/rreport on Linux could crash filePro.

(All) #941

fPSQL could crash if the query required 10K of output per record. (Default increased to 100K, and it now properly reports error if exceeded.)

(ODBC) #943

Define files fails on an ODBC data source when the table name has embedded spaces.

(All) #947

If you have a dummy field which has not been given a length/type in processing, and place it on a screen, and then have a popup screen over that field, and when the popup isn't active (ie: after a POPUP UPDATE, but before a CLEARP, do a SHOW "@...", the contents of that dummy field will bleed through to the popup.

(All) #951

If you have a fuzzy browse lookup with "show =pkeep", filePro would crash upon re-executing the lookup.

(All) #954

If you have any pending FUSHKEYed keystrokes, the filePro processing debugger didn't update the screen as you stepped through processing.

(All) #955

-SX flag to *report did not load the selection set.

(All) #939

FFNUMXBUILD was incorrectly documented as allowing up to 128000. The real maximum is 32767, which is now enforced by filePro. (Setting it higher than the maximum value will result in using the maximum)

(Linux)

filePro had problems with the shared libtermcap on 64-bit Linux platforms, so we now statically link to our own termcap library.

(All) #924

FFNODFMMSG=OFF turns off ddefine's "FFNODF=ON" notice. (Default: ON)

(Windows)

praint now has an F6 option to display a list of system-defined printers while in the "destination" field of the printer config screen.

(All) #927

Fixed a problem with node_addr and DKNF errors on indexes which are larger than 65536 blocks in size.

(All) #938

MEMO handle EXPORT filename APPEND

BLOB handle EXPORT filename APPEND

Would crash at runtime.

(All) #931

FIELDNAME/LEVEEDIT/VAL didn't take an expression for the field number.

(ODBC) #932

dreport was unable to post new records to some high-level ODBC data sources.

(*nix) #933

If you were to have a variable/array/lookup/etc called "USER", then the parser would allow some invalid syntax, where part of an expression was missing, such as "aa = bb +".

(Linux) #934

There are two distinct, and incompatible, pty methods on different Linux boxes. fpdaemon now includes both methods in a single binary, and determines at runtime which method to use.

(All) #935

dxmaint would show "not available" on indexes that were in use, but it would still let you rebuild them.

(All) #936

MEMO...SHOW now accepts the NOWRAP flag, just as MEMO...EDIT does.

(Linux) #926

fpcopy would fail with "cannot create new data file" on some Linux systems.

(All) #779

Executing a multi-line DIM would cause line 1 of the processing table to be executed for each of the continuation lines.

(All) #783

A non-dash lookup to the main file may clear non-".g" DECLARED variables. (See also #785.)

(All) #785

If you have DECLARED GLOBAL variables in the automatic and/or the input/output table, and then CALL another table which DECLAREs additional GLOBAL variables, such that the total number of such variables crosses a multiple-of-32 boundary, and then from within the main input/output table execute a non-dash lookup to the main file, filePro may crash. (See also #783.)

(All)

Add PFCLOSEPENDWARNING=OFF to disable the warning if you attempt to close an HTML tag when it was not open.

(All) #801

Swapped the memo editor extended functions T and I from

T - Toggle insert

I - Insert time

to

I - Toggle insert

T - Insert time

Use FFMEMOEDITOLDKEYS=ON to restore the old keys.

(All) #803

Increase the compiled tok code limit from 128K to 2MB.

(All) #808

When passing both "-Xh" and "-D" flags to *clerk, the "Press DEL" prompt from the index box wasn't cleared.

(fPSQL) #817

"SET QUALIFIER" was ignored in the network version of fPSQL.

(Solaris/AIX) #822

"GOSUB (expr) OF ..." could crash the prc compiler on Solaris and AIX platforms with a SEGV or bus error.

(Solaris/AIX) #823

A DKNF error could cause a SEGV or bus error on Solaris and AIX platforms.

(*nix) #827

The MSB of the color attribute bit originally meant "blink" on MS-DOS systems. Windows changed the meaning to "high-intensity background". The *nix version of filePro still used this for "blink". Moving color screens from Windows to *nix would cause a different appearance between Windows and *nix if this bit was used anywhere.

The *nix version of filePro now ignores this bit by default, as ANSI does not define a "high-intensity background" escape sequence.

FFVSBBLINK=ON will restore the old behavior of MSB meaning "blink".

(Quikstart) #784

If you have an array aliased to dummy fields and use "CLEAR arrayname", and run in read-only mode, filePro would report an "attempt to modify read-only file" warning, even though no real fields were modified.

(All) #707

Some old output formats would cause dmedef to crash.

(fileProG) #720

ddir/dprodir did not prompt to confirm deletions under fileProG.

(All) #763

Using a DECLARE LOCAL variable as an array subscript could give a false "subscript out of range" error in *report.

(fPSQL) #769

When running a query from the command line, and sending the output to a file (via SET OUTPUT), if no records are selected, the "no output generated" message box is no longer displayed.

(All) #777

A stray graphics character was printed at the very end of memos.

(Cosmetic) #780

The spacing of prompts in dscreen's F8/options screen was not consistent.

(All) #781

dxmaint did not allow indexes to be built on the 4-digit-year system date fields.

(All) #786

FORM now strips trailing blanks from the format name, eliminating the need to use FORMname{""}

(All) #790

Pressing F8/Options in dscreen would cause a screen's password to be removed.

(All) #791

When placing a memo on a subtotal/grand total line, the memo's height would be shortened.

(ODBC) #794

Due to a bug in Microsoft's runtime libraries, pmain would crash if the .prt file was in Unix format (LF line endings) rather than Windows format (CRLF line endings).

(All) #809

The filePro debugger would read PUSHKEYed keystrokes.

(All) #813

A lookup back to the main file on an index built on a key length of a multiple of 32 bytes could crash filePro. ("Debug error DAMAGE" on Windows, and SEGV on *nix.)

(All) #841

A "WRITE lookupname" where "lookupname" has been closed can cause filePro to crash.

(fileProG) #829

After executing the configuration editor in ddir, the display was corrupted.

(All) #833

A SHOWCTR or SHOWTOOL executed after a SHOW (w/o row and column) would be displayed centered on line 23 after doing a browse lookup.

(All) #854

LISTBOX and FIELD* functions would pass syntax check (but fail at runtime) if the close parenthesis was missing.

(All) #856

Marking text in the memo editor would sometimes highlight the wrong part of the memo.

(All) #857

An automatic index built on multiple associated fields could cause the wrong record to be read on lookups or index-by searches.

(All) #859

F7 in the memo editor when on the last word-wrapped line of a line of text would place the cursor on the last character in the text, rather than on the end-of-line marker.

(All) #867

Index scan (FFXS=ON, -j) would not correctly handle indexes with descending sorts on scans other than "EQ". (It would only find matching records.)

(All) #874

Hardcopies from pmain would print 1 more line per page than defined for the printer.

(All) #879

If you do multiple cross-reference hardcopies in *cabe in a single session, without printing the prc itself, the page number is not reset between printouts.

(All) #880

The cross-reference listing in *cabe would leave off the field numbers in lookup files if the field number was greater than the number of fields in the main file. (For example, "lookupfile[20]" would print with a blank number if there were less than 20 fields in the main file.)

(All) #704

fPCopy couldn't copy ODBC files.

(All) #865

A selection set with memo_field gt "" as a condition would select all records, rather than just those with the memo not blank.

(All) #866

If you left the menu item "title" entry blank, the item was not drawn, though you could still highlight it.

(All) #871

If you chose to xfer to a file by selecting the menu choice within xfer, rather than the "-lf" flag, it was possible to crash xfer if the record size was very large.

(All) #881

If you brought up the memo editor in READONLY mode, the prompts still included those options which would not be applicable.

(All) #882

IMPORT/EXPORT files were closed before @DONE was executed.

(All) #884

Browse lookups with "show=pkeep" would not retain the cursor position if the key length were zero.

(All ODBC) #793

Under MS-SQL ODBC server, in some conditions, *report and dxmaint would fail with an "invalid descriptor index" error.

(All) #847

Running *report with the "-ro" flag could cause a "file table full" error on some platforms.

(All) #885

Calling DLEN() with a string longer than 256 characters and of "just the right length", or SHOWTOOL with "just the right length" could crash filePro. ("Just the right length" being dependant on the particular platform)

(ODBC) #887

There is an undocumented limit to Microsoft's MFC ODBC library which restricts you to 256 columns in any given view.

(*nix fPSQL) #889

fPSQL's help engine didn't handle Windows-formatted text files on *nix systems.

(All) #892

If you modify a real field in automatic processing in *report, and an automatic index is built on that field, the index may not be properly updated.

(All) #896

DIM of an array name which started with a number did not give a syntax error.

(ODBC) #897

Given the right combination of data source, ID field type, and PFODBCCOMMITTYFE setting could cause dxmaint and *report to improperly sort the file.

(Windows) #904

Under Windows, dxmaint used to show a date of "00/00/00" for demand indexes.

(All) #909

If you have a browse lookup with "-n1" on a multi-key index, and pass a blank key, no records will be matched unless the entire key in the lookup file is all blank, rather than just the key specified in the lookup.

Changes to filePro behavior:

A null key passed to a browse lookup now means "find the lowest key", rather than "find an all blank key". If you specify "rlen=" on a browse lookup, and do not have PFBRWM=ON to make trailing blanks significant, filePro will not make the key shorter than the specified rlen length. (For example, a key of 8 spaces with "rlen=4" will result in a key of 4 spaces.)

(All) #718

With PFXS=ON, filePro scans could fail with an index built in descending order.

(ODBC) #799

ddefine would allow you to build default (empty) indexes on ODBC data sources that had data in them.

(All) #802

If you executed a CHAIN command from within an event which was triggered while updating a screen from a SCREEN command within another event, then when you save the original SCREEN, filePro would attempt to continue from the original SCREEN command, which is no longer there due to the CHAIN.

For example:

@UPDATE issues a SCREEN command. While updating the screen, an @WLF event is triggered. Within the @WLF, a CHAIN is executed. You continue updating the original screen, and press SAVE. At this point, filePro attempted to continue from the SCREEN statement above.

(Windows) #818

The Windows version of filePro used to permit PFDIR to include a drive letter, which should not have been permitted. filePro no longer allows this. Should you absolutely need to have a drive letter in PFDIR, then you can set "PFDSK=", and "trick" filePro into "working" again.

(*nix) #890

setperms and fp.list updated to not touch files placed in the filepro/filename directory which are not part of filePro.

(All) #898

The memo editor didn't have a way to cancel "mark" mode without actually doing a cut or copy. You can now cancel "mark" mode with "X".

(All) #899

Pressing up-arrow from a field on the first row on a screen without a cursor path would not move the cursor to another field.

(Windows) #903

If the last character on a screen was green-on-blue (attribute 0x1A), the screen format would get the last byte truncated, making the file invalid.

(All) #905

ddir/dprodir would give an error if you tried to delete the data from a non-filePro file that pointed to a non-existent file.

(Windows fPSQL) #906

If an error occurred and fPSQL displayed a filename in an errorbox, and that filename included backslashes (ie: "filepro\filename" as opposed to "fileprofilename"), the errorbox would be displayed improperly.

(All) #907

The index selection box in dxmaint would not respond to a keypress if that keypress corresponded to the currently-highlighted index.

(All) #911

dclerk would crash if you attempted to display an old-style screen which had been corrupted.

(All) #914

If you have the same variable DECLARED more than one in the prc file, then F6/D to display dummy fields and variable would show none.

(All) #917

If you have an uncast dummy field which has never had a value assigned to it, filePro could crash if you try to SHOW that field.

(All) #921

In dscreen and drmedef, importing a text file that didn't end in a new line would not import the last line of the file.

(All) #922

Browsing on an index built on an associated field would show the first instance regardless of the correct instance, for the top record on the screen after pressing "R" to reset the browse to the beginning.

(All) #923

A browse lookup on an index built on an associated field would show the first instance regardless of the correct instance, for the first record shown in the browse window.

Spooling for Native Windows filePro

Windows apparently doesn't spool print jobs sent by native windows console applications to local printer ports, the way that it does for MS-DOS programs that do the same thing. (That is, open "lpt1" as a file and write to it.)

We have added the necessary code to the native windows version of filePro to use the Windows printer routines (ie: OpenPrinter, StartDocPrinter, etc.) which do respect the Windows spooler. However, the spooler is also limited to those printers defined in the printer control panel. Therefore, we have made it a requirement that, in order to use the Windows spooler, you must prefix the filePro destination with "win:", as in "win:lpt1:".

The rest of the destination must be the exact port name or printer name as you have defined it to Windows. So, if you have a printer attached to LPT1 that is named "HP DeskJet 870Cse", you would use either:

win:lpt1:
or win:HP DeskJet 870Cse

If you have a network printer "\\server\printer" that is captured to LPT2, called "Bob's printer", and the Windows destination is [\\server\printer](#) then you would use either:

win:\\server\printer
or win:Bob's printer

You could not use "win:lpt2:" as "lpt2" is not the destination that Windows knows the printer by. (Though you could use "lpt2" without the "win:" and go directly to that port without the spooler.)

Remember: You can only use the exact port name or printer name that Windows uses. Anything else will result in a "the parameter is incorrect" error when filePro tries to open the printer.

5.0.14 Release Notes

(ODBC) #726

FFODBCCOMMITTYPE=n

Selects the open-commit-type to use for high-level ODBC data sources, where:

0 = "SELECT * FROMtablename" (default)

Very slow on some data sources with very large files, but uses nothing non-standard.

1 = "SELECT * FROMtablename WHERE id_field = nnn"

(Where "id_field" is the name of the ID field, and "nnn" is a valid ID.)

Usually faster, but may be slower on some systems, as filePro must first determine a valid ID to use.

2 = "SELECT TOP 1 FROMtablename"

Fastest version, but "TOP 1" is non-standard and not supported everywhere. Will cause ODBC failure on those DSNs that don't support it.

(fPSQL) #1

fPSQL now respects FFUMASK for output files it generates.

(All) #237

If BREAK OFF is executed in a CALL/CHAIN process, it now remains off upon returning. Also, it is restored to ON if you go to a new record.

(*nix) #285

fPCopy didn't copy formats with names longer than 14 characters.

(Network) #409

If FFQUAL is set to a non-existent qualifier, the network version of filePro gave an incorrect error message.

(All) #550

The page number was not reset before F8/hardcopy in *cabe.

(All) #634

DECLARED variables over 34 characters caused errors in *report.

(fileProG) #663

If you define buttons while displaying a record, go into browse, and come back, the buttons are gone.

(All) #666

WRITE was not blocked in a table CALLED from automatic.

(All) #681

FFCHECKLOCK didn't pick up some unprotected writes in browse lookups.

(All) #684

When defining processing from dmroedef, FFTOKSIZE was not respected if the file had no automatic processing table.

(fPSQL) #706

A query with a WHERE clause on a field with an automatic index built on it could fail if the index was built in descending order.

(All) #708

DECLARED variables defined without the ".g" attribute were not cleared between records.

(Linux) #711

Running "dcabe filename prcname" could crash on some Linux systems.

(All) #712

FFLOGAPPEND was actually LOGAPPEND. Changed to FFLOGAPPEND as per documentation.

(All) #717

Given the correct sequence of keystrokes, dmoedef could release its lock on the output format, thereby allowing more than one person to be updating it.

(All) #714

EXPORT ASCII with both "-a" and "-x" flags didn't work.

(All) #719

If you modify an index using the "save options without rebuild" and then tried to rebuild a different index, it wouldn't rebuild.

(*nix) #707, #722, #736

Some output formats saved under 4.8 dmoedef would crash 5.0 dmoedef.

(*nix) #724

Only root could delete files in ddir.

(All) #725

EXIT within a CALLED table did not exit, but acted like END.

(All) #727

"CLEAR arrayname" when the array is aliased to real fields did not cause the record to be written if that were the only modification made to the record.

(All) #728

ddir's "delete index" showed all index letters/numbers, even if those indexes did not exist.

(Linux) #729

ddir would crash when deleting indexes.

(All) #733

dxrmain's "save options without rebuild" could corrupt index.

(All) #734

When using *cabe's lookup wizard, using a non-existent qualifier in the filename would crash *cabe.

(All) #735

dscreen would resolve @FD to only 20 characters, when it's really 80.

(All) #740

Although @ONCE in *report is documented as being run prior to any output being done, it was run while sitting on the last record read during the sort/select process. Some people thought that this meant that it was sitting on a selected record.

@ONCE has now been fixed to be not sitting on any record. However, some people depend on their incorrect interpretation of the old behavior, so setting FFOLDONCE=ON will "revert back" to a modified version of the old behavior, where it will now be run while sitting on the last record_selected_ during the sort/select process.

Note that FFOLDONCE=ON may disappear in some future version.

(All) #747

If you were to DECLARE the same variable more than once, and then press F6/D in *cabe to show dummy fields, *cabe could crash.

(All) #750

When a print code requiring a space is on an output format, and you use F7 block functions to move it, the '%' marker is left behind.

(All) #752

When using F6 in praint's printer definition screen to change to printer type, if you select a type with a shorter name than the one currently there, the field w was not properly cleared.

(All) #753

filePro used to accept a period in variable names. This is now not allowed (it never should have been allowed in the first place), in order to permit enhanced functionalities. To revert to the old behavior and allow periods in variable names, you can set FFLONGVARDOT=OLD. Note, however, that this will disable certain features, such as access to ODBC and biometrics, which require that periods not be allowed here.

(All) #757

BLOB/MEMO functions did not work within lookup files.

(All) #766

*cabe could crash when accessing a zero-length prc file.

(All) #767

MINMAX functions did not properly handle expressions as parameters.

(All) #80

Printing of MEMO fields now available.

(Network) #191

The network version of ddefine did not create qualified indexes on new files.

(All) #709

EXPORT ASCIIWORD would always export the same number of fields, regardless of whether the fields were assigned to on each record, even if they were only referenced in a comment.

Now, filePro will only export the number of fields as the highest-reference field actually assigned.

For example:

```
if:
Then: out[1] = 1 ; out[2] = 4
if: xx = "y"
Then: out[3] = 3 ; out[4] = 4
if:
Then: ' out[5] = 5
```

filePro would previously always exported 5 fields. Now, if x="y" is true, it will export 4 fields, and if false will export 2 fields.

To revert back to the old behavior, set PFEXPORTALL=ON.

(All) #282

If you do a fuzzy browse lookup on a field that has an index built on it, the wrong instance will be shown in the browse.

Also, if you were in index mode on the main file, the wrong instance would be shown, even though @AF was correct.

(All) #770

If you perform a browse lookup with a "-n!" (must match) flag on an index built across multiple fields, and the major sort key is a date or time field, the "must match" portion was limited to the major sort key only, regardless of the mlen parameter.

(All) #758

When using a browse lookup on a multi-field index, with "-n!" (must match) across multiple fields, it was possible to get an erroneous "top of file" after scrolling down and then up again.

(fileProG only) #720

No confirmation was asked for when deleting formats in ddir/dprodir.

(All) #773

When accessing BLOB/MEMO fields from lookup files, it was possible for the data to get corrupted.

(All) #774

Under certain conditions, re-executing a fuzzy browse lookup could cause filePro to crash.

5.0.13 Release Notes

(SCO only) #???

Fields with extended-ASCII characters (≥ 128) might not compare correctly.

(All) #653

Browse lookups with only a single line could crash filePro.

(All) #662

DLEN() had an undocumented limit of 255 characters on the input string. The input is now unlimited, and the output is limited to 4095.

(*nix only) #667

Some *nix systems prevent a setuid program running with a real uid of root from executing child processes. The previous workaround (setting the real uid to "filepro" when running as root) causes some things (such as printer banner pages) to report "filepro" as the user.

Added "PFROOTFIX=OFF" to turn off the fix for systems that don't require it.

(Native windows) #???

@ID will now contain the current user name, up to the first 8 letters.

(All) #633

If you have a memo with a line so long that it doesn't fit within the memo editor window, and while in insert mode press Enter near the top of the memo (to split the line), the program may crash.

(All) #???

Enhanced the blobfix utility to do a better job at extracting data from a blob within a corrupted section of the file.

(All) #587

"lookup -pw" did not honor the "-w" flag.

(All) #656

FORM command within a CALL NOAUTOed processing table would use fields from the automatic table.

(Native Windows) #671

sitepwd.exe did not work under XP.

(All) #668

Output formats would not function properly if the height*width was more than 32K.

(All) #672

"ddir -k" did not respect the lockfile.

(Native Windows) #673

p.exe splash screen would fail if the licensee's name contained a lower-case "z".

(Sun and PPC Linux) #675

filePro would crash if TERM/PFTERM was set to an entry that contained a "tc=" value.

(All) #674

IMPORT/EXPORT using an expression for the filename would not compile properly in just the right circumstances. (Symptoms included adding or deleting lines with literal strings would make the problem go away.)

(GI) #644

Timing issues caused connections to GIservers over a satellite link to fail.

(GI) #651

Prompt buttons were missing with browse lookup windows.

(All) #670

autosuf would crash on some systems when adding fields.

(All) #676

dxmaint would not update the status screen while reading large sections of deleted records within the file.

(All) #680

filepro would crash on "-pq" if a printer was defined without any comment.

(All) #682

Dummy fields set in @ONCE did not hold their values in @BRKn and @WGT processing.

(All) #690

In just the right combination of circumstances (full description to come), filePro could crash in the index delete routine.

(All) #683

System fields didn't push left with "<" on the output form.

(All) #700

If @ONCE processing makes an assignment to a dummy field that is not defined in automatic processing, *report may crash.

(ODBC) #692

In some ODBC data sources, updating a record via high-level ODBC would cause a nul character to be appended to the data in some field types.

(All) #686

MESGBOX/ERRORBOX with long lines would drop 1 character at the end of each wrapped line.

(All) #695

*cabe didn't recognize the new @DV system field.

(Native Windows only) #698

EXISTS() would return "1" (true) if the specified path included an existing filename as part of the path. For example, if the file "c:/filepro/backup.zip" exists, then passing EXISTS() the name "c:/filepro/backup.zip/map" would return true.

(ODBC) #691

dprodir would crash attempting to display info on an ODBC file.

(fileProGI) #688

"MEMO ... EDIT TITLE ..." didn't display the title.

(fPSQL) #618

fPSQL did not recognize new 4-digit-year system date fields, nor PFSYSYR4=ON.

(Unix xfer) #687

The Unix version of xfer did not send screen/output formats that were longer than the old limit of 14 characters.

(All) #661

BREAK OFF wasn't honored within a browse lookup.

(SCO only) #677

If a file exceeded the ulimit file size, filePro would crash with a SIGXFSZ signal, rather than give the "file too large" error.

5.0.12 Release Notes

Never released

5.0.11 Release Notes

5.0.11 Windows versions includes the use of encrypted distribution files.

In order to install, you will need an Authentication Key file which you can get from our download site or can request from customer service. You will need the End-user Company Name in order to request/download this keys file

(All) #???

In just the right circumstances, it is possible for ddefine to crash in the middle of a restructure when adding fields off the end of the record. (Special "5.0.10a" release of ddefine made.)

(All) #654

If you execute a lookup r=free to the current file, followed by a lookup-dash by record number to that record, and there were no free records in the file, the lookup-dash will fail.

(All) #630

menupass now accepts menu name from the command line.

(All) #620

A protected lookup to the current file, to the current record, would unlock the current record, even in update mode.

(All) #647

If the very first thing you do in dscreen is attempt to create a new screen, dscreen would crash.

(Windows) #649

The initial splash screen now responds immediately to a keypress.

(All) #650

A menu commandline that included the sequence slash-p-space (as in "-h 'A/P menu'") wouldn't work.

(All) #646

Renaming a file in fpcopy might not remove the old directory.

(All) #357

BREAK OFF did not carry through call/chain.

(All) #648

dxmaint would not accept system fields on the command line.

(GI) #655

pmaint printer editor did not accept F5/F6 function keys.

5.0.10 Release Notes

5.0.10 Windows versions include the use of encrypted distribution files.

In order to install, you will need an Authentication Key file which you can get from our download site or can request from customer service. You will need the End-user Company Name in order to request/download this keys file

(All) #577

HTML :IN :ML generated "MAXLEN=n" rather than "MAXLENGTH=n" attribute.

(5.0) #582

The TVM_*(*) functions did not handle comma-type edits for input.

(ie: "12,345.67" was treated as "12".)

(4.9) #581

Add PFMEMOINSERTMODE=ON to set insert mode on by default in the memo editor.

(All) #585

PFCHECKLOCK does not report errors on "-p" lookups if you modify fields after a WRITE.

(All) #???

Enhancement to PFTMP for specifying temp directory:

If PFTMP is not set, then use TMP. If both are unset, use TEMP.

(All) #???

Tweak to the above: some versions of Windows allow for a list of directories in the TMP/TEMP variables, separated by semicolons.

In such instances, filePro will use the first directory listed.

(And, change the Unix version to do the same, should there be a version of Unix that allows a colon-separated list of directories.)

(All) #615

runmenu was using PFTEMP rather than PFTMP to store menu batch files.

(Quickstart) #616

SHOW statements were truncated to 255 characters.

(All) #617

Fuzzy browse lookups with processing caused non-global dummy fields to be cleared. ("Normal" browse lookups were okay.)

(All) #619

Fix case where "node_addr(x!=y)" error would be generated.

(All) #621

If you have an unprotected lookup to the current file, and find the current record, do a GETNEXT, modify that record, and CLOSE the lookup, the modified record will remain locked.

(All) #628

If you have a lookup dash in @ENTSEL processing on a file with no automatic processing, it is possible to get DKNF errors displayed.

(Note that the indexes aren't actually corrupted in this case.)

(All *nix) #???

Pressing "N" or "I" in the monochrome attribute editor of dscreen caused the display to fill with text.

(fileProGI) #???

Shrinking a memo in the GI memo editor caused the tail end of the original memo to remain.

(Some *nix) #???

Running menu items with the "no return" flag ("##") could cause the stty settings to get confused.

(All) #275

Using F8/Options to set the processing password in *cabe would

cause subsequent tables loaded in the same session to have the same password assigned.

(All) #588

A "lookup - r=nn" which fails due to an EOF condition would cause the current record to be unlocked.

(All) #601

Using block functions to copy processing lines from beyond the last one could cause garbage to be copied.

(All) #602

fPTtransfer now copies "map.new" files as a map.

(All) #604

Pressing F6 in ddir does not bring up the config editor if the highlighted name is not a valid filePro file.

(All) #610

PFLOOKWIZPROT=ON was not recognized properly.

(All) #612

ddir did not empty the blob file on kill data / retain formats.

(All) #613

Lookup aliases in the format "letter-number-others" (ie: "d4a") would fail syntax on assignments.

(All) #614

You can now do CContains comparisons on memo fields.

(fPSQL) #618

fPSQL did not recognize 4-digit-year date fields, nor PFSYSYR4=ON

(UnixWare only) #640

Some configurations of UnixWare do not allow a setuid program when running as real-uid root to execute child processes.

(UnixWare only) #641

An incompatibility in UnixWare running OpenServer binaries caused SLEEP to be ignored.

(All) #635

dxmaint -M "" did not override PFQUAL environment setting.

(Linux only) #642

Fuzzy searches on Linux did not return results consistent with other filePro systems.

(All) #637

Extended selection sets didn't work if comparing to a value with a colon in it.

(All) #591

RAND("-l") didn't properly seed the random number generator on some systems.

5.0.09 Release Notes

All Platforms

#489

Backed out change in behavior regarding browse lookup prompts being displayed even if xkeys specified.

Added "-DL" flag to *clerk to tell filePro to display the prompts, even if xkeys are specified.

#547

Shrinking an existing memo/blob to zero bytes would corrupt the blob file. (Note that this is not the same as deleting the object.)

#214

If you use ddir/dprodir to empty a file, and have not yet added any records nor rebuilt indexes, it is possible that *clerk/*report will still see the indexes as empty if another process has added records to the file while you are still in your *clerk/*report session.

Note that the records will be correctly seen -- it is only the indexes that may still be thought to be empty. (The symptom is "no matches found" or failed lookups.)

(No task #)

Add a new end-of-line option for printing -- backslash+LF. Mostly, this is for the RTF printer table, which requires the backslash at the end of every line for a continuation marker.

#566

If PFOLDIX=ON and you build a single-key index in dxmaint with a descending sort, it erroneously built an old-style index, which do not support descending keys.

#561

Change dxmaint to prevent building of indexes on BLOB/MEMO fields.

#501

Repeated use of F6/D/L to display long variable names in *cabc caused corrupted list to be displayed.

#562

Add PFFIXNOLock=OFF to back out change related to automatically locking future executions of a lookup which was modified without a "-p".

#536

DECLARE GLOBAL name(len,type,g) fields were not properly initialized in quikstart.

#252

Assignment/concatenation operators ("=", "<", "{", and "&") now work with memo fields.

eg: memofield = memofield & @TD
field = memofield
memofield = field

#515

If you have an array aliased to real fields, and the only modification to the record in output processing is "CLEAR arrayname", the record would not be written without an explicit WRITE.

#534

LOGTEXT would log some filePro debug information into the log file, in addition to the LOGTEXT items.

(No task #)

Add MOUSE PATH [ON|OFF] as an alias for CURSOR PATH [ON|OFF]

(No task #)

Add PFFIXEDLISTSIZE=ON to prevent filePro from shrinking selection lists. This allows screen readers for the blind to be programmed with fixed screen locations for such lists.

#569

CLOSE() did not return zero on success. (Return value was undetermined.)

(Windows network only) #570

It was possible that, if one process was accessing an index for the first time and another was reading/writing the head of that index at the exact same time, the read/write process would get a Windows locking violation (Windows error #33).

(fileProGI) #573

filePro will now break up queued BUTTON_OP commands into 50-command chunks, to prevent any buffer overruns in the fileProGI client.

(*nix only)

When using the "+ENV+" command syntax in a user menu, it was possible to get an error that "exec" was not a valid command/filename.

(No task #)

Add the ability to put "%varname%" and "\$varname" in the title of user menus, and have them displayed at runtime.

(No task #)

Add PFSHOWROWCOL=OFF to turn off the row/column display in programs like dscreen, dmoedef, and *cabe. It can confuse screen readers for the blind, as the numbers are read every time you press a key.

(No task #)

Add PFINSERTMODE=ON to set insert mode in by default in *cabe/*clerk.

(Native windows, network only) #517

Native Windows network version did not respect PFDIRFILTER=ON.

#576

If you executed a record-number lookup with "-p", and the lookup failed due to the record being deleted, the record was locked anyway.

(No task #)

New flags for dxmaint:

-LY/-LN Include/Exclude index from lists.

-KY/-KN Save / don't save changes without rebuilding.

(fileProGI) #479

Short selection didn't set date field types, preventing GI from properly displaying calendars.

(fileProGI) #522

Under GI, only a single click of the cancel button is required, even if the server is Unix. However, this would cause problems if PUSHKEY "[BRKY]" is used. Change so that double-break is still needed under Unix if they are sent via PUSHKEY to make text and GI consistent.

#508

When rebuilding indexes with -r/-ra, the hidden flag was lost.

5.0.08 Release Notes

(Native windows)

On some Windows systems, p.exe couldn't execute "/fp/progname" command lines. (It would work with "\fp\progname" or just "progname".)

(All)

On some systems, the following expression would generate undefined results:

```
UncastVariable = mid(UncastVariable,start,len)
```

(All)

The following code could leave the record being updated unlocked:

```
@wfl  
lookup self=filename k=1 i=a -nxp  
m = self(@m)  
lookup - r=m
```

(ie: a protected lookup to the current file, and then a lookup-dash to that record.)

(All)

Upon returning to a user menu with "@command", garbage might appear on the screen before the "Press Enter to return to menu" prompt.

(All)

Add a title to the dxmaint main menu.

(G)

Garbage could appear on the screen following the location for file name input.

(G)

Make sure that pty device is in raw mode before sending PROCESS_INIT message. (Could cause lockup if PID/PPID happened to contain a 0x0A byte.)

(G)

Allow pmaint's printer configuration screen to accept mouse input.

(G)

If an edit definition was changed, leaving existing data that would fail the new edit, using the mouse to move around the screen in *clerk could freeze filePro as it attempted to display the "edit failed" message.

5.0.07 Release Notes

(GI) #2

Strip out / expand backslash codes from select lists.

(GI) #478

When returning to user menu with "@command", leave the raw-mode screen rather than the filePro splash screen, when waiting for Enter.

(All) #480

"COPY lookup1 TO lookup2" didn't force write of lookup2, unless it was a free-record lookup, or some other change was made to lookup2.

(GI) #481

showbutton/brwlook problem where button goes away

(All) #489

If exitkeys are specified on a browse lookup, filePro does not display filePro's browse lookup prompts. (It assumes that you have placed your own prompts to go with the exitkeys.) Now, if you specify the new "-DE" flag (suppress filePro prompts only at @entsel), filePro will still display the browse lookup prompts even with exitkeys.

(All) #490

-DE flag browse text should not disappear

(5.0) #498

"MEMO lookupfile[fdno] EDIT" didn't write the updated memo.

(Native windows) #499

There is a bug in Windows' console application support that would cause filePro to sometimes see two spaces for a single keypress. This bug has been worked around.

(All *nix systems) #502

Close security hole in mkdir.

(GI, UnixWare server) #503

Fix problem with default pseudo-tty port settings, to force a mode that filePro can use.

(GI) #521

Make sure that filePro doesn't display end-of-field markers under GI.

(GI) #523

Fix problem with certain byte sequences in data stream causing the stream to get corrupted.

(All) #540

REPEAT() would crash filePro ("out of memory" error) if passed a negative length.

5.0.06 Release Notes

#N/A

xfer now includes the necessary "-m xxx" flag in the buildix script to rebuild qualified indexes.

#N/A

The "PageUp/PageDown/F5" prompts are now shown in dxmaint/*report on the sort screen's field listing.

#39

F9/goto in config editor will now accept line number as well as text.

#173

If you scrolled down in the config editor while just viewing, and then entered update mode, you were returned to line 1.

#259

When the number of print codes was increased to 9999, pmaint no longer allowed you to type "END" for F9/goto.

#264

Add F8/Options to *cabe, to allow setting of the processing password.

#N/A

When using F8/Save, the new prc table will have the same prc password as the source table.

#249

fPcopy didn't copy blob files.

#252

You can now do a CContains compare on memo fields.

#57

HTML:TX can now take a memo field.

#183

"READONLY" flag added to MEMO nnn EDIT:

```
MEMO nnn EDIT [ (row,col,height,width [,startrow,startcol] ) ]  
[ WRAP | NOWRAP ] [ READONLY ]
```

#N/A

When creating a new file in ddefine with BLOB or MEMO fields, the blob file wouldn't get created.

#279 (native windows only)

On some systems, the text-mode mouse cursor would be enabled while running filePro full-screen.

#101

Temporarily remove the F8/options box from dxmaint when building demand indexes, as the options don't work (yet) with them.

#83

*cabe's F9/Goto will now find multiple occurrences of a string on the same line. (And fixes the "not found" problem introduced in .05K1)

#123

Duplicate DECLAREs are now caught in *cabe.

#140

Browse lookups which had both "k=(expr)" and "b=(expr)" would not work correctly. (Internally, filePro got the "k=" and "b=" values backwards.)

#34 (native windows)

When using "WIN:printername" as a destination, FORMM left the spooler in limbo.

#N/A

Add PRINTER FLUSH command, which flushes any printer buffers within filePro.

#N/A

New system array, @@UNAME[], which returns the system uname() info:

[1] = sysname
[2] = nodename
[3] = release
[4] = version
[5] = machine

For Windows systems, the following values are returned:

sysname: One of: "Win9x", "WinNT", or "Windows" (if type cannot be determined)
nodename: the value from GetComputerName()
release: major.minor.build (for example "4.10.1998" for Win98)
version: the szCSDVersion from GetVersionEx()
machine: blank

For *nix systems, see your O/S manual for details on each fields meaning.

#280 (native windows only)

Restructure of files in ddefine would fail at the freespace check if PFDSK was set to more than one drive.

#268 (native windows only)

SHOW on row 25 did not display anything.

#30

Variables DECLARED in sort/select processing were not properly retained for the output phase in quickstart.

#269

rreport could lose some printer setup information after executing a PRINTER FILE command. (Symptom: no end-of-line generated at page boundaries.)

#83

After doing an F9/text-search in *cabe and finding text that's not at the beginning of the line, using F9/goto a line number (or "END") would not position the cursor at the beginning of the line.

#298

MESSAGE SEND would crash filePro.

#284

@PD and @PC increased to 80 characters, to match pmaint.

#301

If a CALL has been executed, then using "E" expressions in the processing debugger may not recognize all variable/lookup names. Only occurs if

the CALLED processing table has fewer symbols than the table being debugged.

#297

Within automatic processing, when scrolling through a file, memo fields would always compare to null, even if filled in with text.

(Native windows) #300

"WIN;printname" syntax caused the printer init code to be sent with each form printed to FORMM command.

#39

Configuration editor F9/search enhanced so that you can go to a specific line by typing its line number, and you can search for numbers by typing a quote as the first character. (Same as *cabe's search.)

#N/A

Add PFLOOKWIZPROT=ON to change the lookup wizard's "protect record" default to "Y".

#N/A

New STATUS object allows you to save/return the status of: break, cursor, video, escape, and background. Allows subroutines to enable or disable these items, and then restore them to their original state.

handle = NEW STATUS()

STATUS handle GET

STATUS handle SET

#355 (native windows only)

Windows ME and/or 2000 appears to not beep when a ctrl-G is output from a console application. Change the native windows version to use the MessageBeep() API call instead.

#N/A (fileProGI only)

A new TITLE verb is added to the MEMO EDIT and MEMO SHOW commands, to allow a title to be specified for the window. Currently, only the fileProGI client will display the title.

Example:

MEMO memofld EDIT (row,col) TITLE "Last updated: " < @UD

#N/A

The "-pq" list no longer interprets backslash codes, so that you can have printer comments like "Send to \\server\printer".

#N/A

Adding more than 100 lines to a memo field in the non-wrapping editor could crash filePro.

#358

PFSKIPPEDLOG=filename would crash *report.

#219 (fileProGI only)

Cursor path is now enforced within the GUI environment.

PFFORCECURSORPATH=OFF turns off forced cursor pathing logic.

#N/A (native windows only)

PFSHOWWINERROR=ON shows value of GetLastError() when system error occurs.

#N/A (native windows only)

Some system errors didn't display the correct error message text.

#N/A

PFREFRESHRATE=nnn sets the screen refresh rate during sort/select and output phases in dxmaint/*report to once every "nnn" seconds. (default=1)

#304

If you CHAIN from a processing table with a DECLARE GLOBAL, and then CHAIN back, the value of that variable was lost upon return.

#N/A (fileProGl only)

New command "CURSOR PATH ON/OFF" to give programmer ability to turn off forced cursor pathing in GUI.

#404

If you are building an index, and you have a duplicate key that spans several blocks, and the last block is exactly filled, and you insert a new key that comes between that multi-key and the next key, it was possible for dxmaint to crash. (Though it appears to only have crashed if running under fileProGl.)

#104

Pressing F2/DEL at the end of a line in the memo editor now joins the lines together by deleting the end-of-line marker.

#16

If you have a memo that is smaller than the memo editor window, and you start inserting blank lines with Enter in insert mode, then when the memo reached the height of the editor window, it would crash.

#N/A

New *clerk flags:

-de Turns off @entsel prompts, but leaves update mode prompts.

-du Turns off update mode prompts, but leaves @entsel prompts.

#N/A

New environment variable PFNEWSK=ON (default: OFF) allows new @SK values to be seen by processing. Specifically, the only one right now is "MOUS", which will be seen as "ENTR" otherwise. (Lots of processing tables depend on certain values in @SK in order to function. If an unknown value is set, the cursor won't leave the field or will behave in undesired ways.)

A new system field will be assigned later, which will always contain the "real" keystroke value.

#N/A

Manipulating memo fields within a lookup file could crash quikstart.

#476

In *clerk, if you press F6 on a field with no browse lookup, and the press BREAK when asked if you want to create one, you were taken out of update mode, rather than just cancel the brwlook create.

#87

If you have an index with a duplicate key that requires more than one block in the index, and add a record whose key comes immediately after that duplicate key, and then continue adding records with the duplicate key, while not adding any other key between it and the first key of the next node in the tree, so that it no longer fits into the same block, the parent node's pointers were not properly updated.

This could lead to a "deleted key not found" error.

#249

DOS/Windows version of fpcopy, when renaming a file with blobs, would fail to move the blob file to the new directory.

(Native windows) #499

Native windows version would sometimes see two spaces when you pressed the spacebar once.

#477

Hardcopying a screen in *clerk, when the printer destination was set to the screen (ie: with "-pv") would produce a blank screen.

5.0.05 Release Notes

(All)

*clerk can now go to the end of a demand index via F7. (It used to give a "can't do bxhigh() on demand index" error.)

(Native windows)

SYSTEM() will execute via command.com if a full path is not given on the command line. (This was backed out while working on letting fPclient run scripts, but now that we're using pipes rather than sockets, that fix is no longer necessary, and running via command.com is necessary to run any built-in commands or to respect I/O redirection.)

(All)

Blob files can now be placed on drives other than PFDATA (just as key/data/index files can).

(All)

Increase width of ddefine's "create screen 0"/"create default report" dialog box, as messages are now wider due to showbuttons.

(All)

*cabe - immediately flush buffers after writing processing table, in attempt to cut down on false "truncated processing table" reports.

(Quikstart)

CREAT() was not accepted as an alternative to CREATE() in rcabe.

(All)

Add PFHELDIR=path to set alternate help file directory. If help not found there, filePro will then look in \$PFPROG/fp/lib as well. (This only affects the filePro fp/lib help files. Application help files are not affected.)

(All)

Turn off expiration date. Programs will no longer expire.

(All)

Change locking logic for new-style automatic indexes. The new logic will allow multiple read-only accesses to the index to occur simultaneously. Only index-update accesses are now single-threaded. On systems with many users doing many index searches on the same file at the same time, the load on the system is dramatically decreased and response time is dramatically increased with the new logic.

(All)

If you had a duplicate label with a DECLARED variable and a processing line, and the prc line was empty except for the label, filePro could get in an infinite loop attempting to display the syntax error.

5.0.04 Release Notes

Restructure of a file with data in it, to change between text and memo didn't handle the qualified blob file.

While someone was in the memo editor, other people would be locked out of accessing any memos/blobs from the same file.

(Native windows version only)

DOS commands built into command.com would not execute via SYSTEM command

5.0.03 Release Notes

After using "!menuname" to nest menus, returning from the nested menu, and executing a menu command line, the wrong menu (the previously-nested one) would be displayed.

Due to the change in 5.0.02 of re-reading the user menu, menus with passwords started asking for the password every time you returned from executing a command.

Changing a BLOB/MEMO field to a "regular" field in ddefine caused the restructure to crash.

(Unix only)

dmakemenu would crash upon saving menus.

5.0.02 Release Notes

Using the syntax "MEMO fld EDIT ()" or "MEMO fld SHOW ()" would crash *clerk/*report programs on some systems.

Enhance the "There are no filePro files" errorbox to include the directory where filePro is looking.

Help screens lost the ability to show attributes in 5.0.00K5. Fixed.

Add PFHELPAUTOGOTO=ON to automatically force F9/goto upon entering help.

Fix memo/blob problem with deleting a memo and then lengthening a memo (which happens to reside in the same block in the blob file) causing corruption.

Adding more than 100 lines to a memo field in the wrapping editor could crash filePro.

dxmaint F8/options screen didn't allow blank to represent "no".

Memo editor F8/options didn't have any text after "F10" in "F10 - help".

Memo editor respects "-d" flag to not display prompts.

runmenu re-reads the menu upon return, allowing the command to update the menu and have the changes reflected on return.

5.0.01 Release Notes

Word wrap within memo editor. Also, the MEMO EDIT command is enhanced to take an optional WRAP/NOWRAP flag. The default is currently WRAP, but that may be configurable later.

MEMO field EDIT [(row,col,height,width,startrow,startcol)] [WRAP|NOWRAP]

(Native windows version only.)

Don't turn on mouse cursor while in full-screen mode.

Typing more than 100 characters at the end of a line in the memo editor while not in insert mode, could crash filePro upon exiting the editor.

Fix buffer-overflow problem from increased max line within help.

Fuzzy browse lookups didn't work if helped by an index, if that index was in 4.1-style format.

Last remaining known "deleted key not found" problem fixed.

If you had a duplicate key that spanned exactly 9 blocks in the index, and the ninth block contained exactly one record, and you deleted a record stored in the first block, the eighth (and now final) block was sometimes not properly marked as end-of-chain.

Fix problem with opendir/nextdir truncating filenames at a space.

Fix single-break problem in *cabe under Unix/Linux.

Fix problem in word-wrap memo editor, where deleting lines could leave a ghost of the last line at the bottom of the window.

Fix problem in word-wrap memo editor, where pressing Enter in insert mode would still display the old line in its entirety.

Change logic of word-wrap editor, so that whitespace will wrap to the next line if necessary.

If typing off the end of a line, in overstrike mode, every 100th character might not be stored properly.

5.0.00K5 Release Notes

Increase max line within help file from 132 to 512, to allow for multiple backslash-coded items.

Fixed MEMO DELETE to release memory location after delete.

Memo editor now keeps track of the desired column when moving vertically through shorter lines.

Having an extraneous comma within a DECLARE statement would cause an erroneous syntax error at the first reference to a literal. ie:

```
DECLARE variable1, variable2,  
or DECLARE variable1,, variable2
```

MEMO EDIT / MEMO SHOW can now leave off the (row,col,...) entirely, and a default location/size will be used.

Fix *cabe DEL problems under Unix/Linux.

5.0.00K4 Release Notes

ddefine allows restructure of to and from memo/blob field types.

Deleting the last line of a memo could cause a crash upon save.

If the only changes made to a record were via MEMO EDIT, those changes would not be saved.

Fix *cabe DEL problems under Unix/Linux.

Memo editor now requires double-break to cancel.

Memo editor - when you marked text end-to-front, cut/copy would crash.

Memo editor didn't find text at end-of-line.

"Toggle insert" added to memo editor F8/options menu.

Repeated use of the sequence MEMO DELETE, MEMO IMPORT, MEMO EDIT while remaining within the processing, could corrupt the blob file.

5.0.00K3 Release Notes

Never Released.

5.0.00K2 Release Notes

MEMO EDIT would crash at 101 lines or any increment beyond a per 100 line entry before the 100 line buffer established was refreshed.

Added SAVE and BRKY references to the prompts of MEMO EDIT windows

PFNOBOX=ON created a piece of box that displays in lower right quadrant of MEMO EDIT box.

Changed MEMO EDIT Extended Functions for Find to respect the PFDLGENTER setting

Fixed [field] copy to a lookup[field] of a MEMO field

Fixed lookup[field] CLEAR after a MEMO lookup[field] SHOW

Fixed Native/DOS saving proper CR/LF of a MEMO TEXT EXPORT after MEMO EDIT

Fixed Memo Paste Function which would not work at the End of File

5.0.00K1 Release Notes

New Extended Features (F8) in MEMO editing which includes

Mark, Cut, Delete, & Paste

Find & Find Next

Insert Current Date

Insert Current Time

Environment Variable for *NLX platforms to set the Graphics character for PFSHOWF6ARROW is now GY

fPTransfer has been update to handle blobs as binary files.

Many cosmetic and messaging enhancements

5.0.00 Release Notes

Commands for MEMO management

To Display a text MEMO field with allowing edit

MEMO field SHOW (row,col,height,width)

To Remove a text MEMO field display-only window

MEMO field CLEAR

PFLOOKUPNOFILE=ON

In *cabe, if you define a lookup to a file that does not yet exist, and this is set, you will no longer get a "No or invalid map" error.

Default is OFF.

Spooling for Native Windows filePro

Windows apparently doesn't spool print jobs sent by native windows console applications to local printer ports, the way that it does for MS-DOS programs that do the same thing. (That is, open "lpt1" as a file and write to it.)

We have added the necessary code to the native windows version of filePro to use the Windows printer routines (ie: OpenPrinter, StartDocPrinter, etc.) which do respect the Windows spooler. However, the spooler is also limited to those printers defined in the printer control panel. Therefore, we have made it a requirement that, in order to use the Windows spooler, you must prefix the filePro destination with "win:", as in "win:lpt1:". The rest of the destination must be the exact port name or printer name as you have defined it to Windows. So, if you have a printer attached to LPT1 that is named "HP DeskJet 870Cse", you would use either:

win:lpt1:
or win:HP DeskJet 870Cse

If you have a network printer "\\server\printer" that is captured to LPT2, called "Bob's printer", and the Windows destination is "\\server\printer" then you would use either:

win:\\server\printer
or win:Bob's printer

You could not use "win:lpt2:" as "lpt2" is not the destination that Windows knows the printer by. (Though you could use "lpt2" without the "win:" and go directly to that port without the spooler.)

Remember: You can only use the exact port name or printer name that Windows uses. Anything else will result in a "the parameter is incorrect" error when filePro tries to open the printer.

To submit 4.8 bug reports

1. FAX them to (317) 826-0932 clearly marking them as 4.8 bug reports
2. EMail them to fpsupport@fileproplus.com including the text 4.8 Bug Report in the subject line
3. Call the customer support number (317) 802-0138

4.8.12 Release Notes

Changes from 4.8.10 to 4.8.12

All

Various Index Fixes

Fixed Incorrect Prompt Display in Help

Correct a different result between d & r clerk in the handling of Global variables.

Fixed an error in debugging CALL's when long expressions

Erroneous data display on Associated Fields in Browse Lookups

Bug in F9 GOTO search on *cabe on same line duplicates

Native

Fixed sending init code on every occurrence of FORMM

4.8.11 Release Notes

All
DECLAREd variables that have the ".g" flag were not initialized to blank.

Native95
A version of native95 went out in the past that had an internal debugging flag set, causing @SK to contain the internal hex value of any keystroke that wasn't a "normal" special key. (This would cause @SK="" to fail in tests.) This was corrected, but some people commented that they liked that feature, so a new config variable PFSKHEX=ON will turn it back on.

All
*cabe didn't recognize -pn/-pc command-line flags.

All
If you build a demand index on a zero-length field, *clerk would freeze if you attempted to go into index mode on that index.

Native95, plus only
If you have multiple import/export statements in a processing table, with a variable-named import/export followed by a hardcoded-filename import/export, dclerk/dreport would crash upon loading the prc table.

All
Fix problem with HTML :CE for <CENTER> which would cause a superfluous <BLOCKQUOTE> tag to be generated.

DOS versions (NO LONGER USED)
If a file used all 16 automatic indexes, and you did a dxmaint "-ra" on the file, it would fail on index.P (unless you increased PFFILES).

Native95
Native windows *clerk, when attempting to delete a record immediately after finding the record by index mode, would display the confirmation message in DialogNormal rather than TextNormal.

All
INPUT POPUP for some reason set a minimum width at 10. Change to 1. (If there was a reason, I don't recall what it was.)

All
READ()/READLINE() now accept long-named variables as the second parameter.

All
HTML :DI <DIV> and :SP produced junk close-tags.

All
In some instances where a subtotal-with-pagebreak contained a single record to be printed, the page's heading wasn't printed.

All
When doing a browse lookup "-" and there were more than approximately 250 exact matches, and using the new 4.5-style indexes, the browse would not place the cursor on the last match. (The cursor would position to the last match listed within the first index block.)

All
The following would crash on some systems:

```
xx(12,.0) = "0"  
yy(11,rj0) = xx
```

Unix
Fix *cabe DEL problems under Unix/Linux.

All
Having an extraneous comma within a DECLARE statement would cause an erroneous syntax error at the first reference to a literal. ie:

```
DECLARE variable1, variable2,  
or DECLARE variable1., variable2
```

All
Fuzzy browse lookups didn't work if helped by an index, if that index was in 4.1-style format.

All
"Deleted key not found" problem fixed.
If you had a duplicate key that spanned exactly 9 blocks in the index, and the ninth block contained exactly one record, and you deleted a record stored in the first block, the eighth (and now final) block was not properly marked as end-of-chain.

All
Fix problem with opendir/nextdir truncating filenames at a space.

4.8.10 Release Notes

4.8 Unix

Fix problem with "-pq"/filename attempting to execute filename, rather than output to it.

Native95

Fix problem of occasionally missing release of Alt when CapsLock down, causing Windows CapsLock bug workaround to be disabled.

Native95

Fix problem when pressing shift-numpad key twice in a row would act as a Break.

All

Fix @FN length problem, where it was truncated to 10 characters in some places in filePro.

All

Fix problem with SQR(0) returning "/D0". Now returns "0" as it should.

All

Fix ddir problem emptying indexes when nodesize > 1K.

All

Fix fuzzy browse lookup problem when scanning on associated field, where first instance would always be used.

DOSNative95 Network

EXPORT ASCII -A truncated file on network version.

All

If filePro cannot read the full lockfile, an error is now generated.

All

DECLARE will now correctly give a syntax error if you try to declare a variable that starts with a digit.

All

If an invalid edit name was given to DOEDIT(), the function returned garbage. It will now return a null field.

All

If an error occurs in automatic processing (one that would normally cause the "A system/filepro error has occurred" message to appear) while in add records mode, no error is reported, and a new record is added. (If this error occurs repeatedly, you will be stuck in an infinite loop.) Note that the error must be in automatic and you must be in add records mode.

Native95

Under certain conditions, Windows will return ERROR_WRITE_FAULT or ERROR_IO_DEVICE, rather than ERROR_NOT_READY, for a printer-not-ready condition. These are now trapped as a not-ready error, rather than reporting "error -1"

If you continue to get "error -1" errors on printing, you can set PFNTPRTERR=ON, and filePro will display the internal Windows error number, which we can use to determine the cause.

Unix

If PFREADONLY=ON, "invalid argument" errors could occur if you access a 4.5-style automatic index.

Unix

On some systems, ASC() would return negative numbers for characters greater than ASCII 127. (Specifically, it would return the proper value, minus 256.)

All

If you have a `getnext` loop, and within that loop you have another lookup to the same file, and are posting new records (or changing existing records' index key) such that the new index key value is in close proximity to the current `getnext`-loop index key, it is possible for the `getnext` loop to malfunction. (Skip records, or duplicate records.)

Unix

If a help file was in DOS format (ie: includes `"^M"` for end-of-line) then `filePro` would freeze when reading that help file.

All

If `PFCHECKLOCK=ON`, the error is now reported only once per lookup, rather than on every assignment.

All

`Dxmaint` would display demand index dates after 12/31/99 as "mm/dd/100".

Native95

Added ability to use the Windows spooler:

Spooling for native windows filePro

Windows apparently doesn't spool print jobs sent by native windows console applications to local printer ports, the way that it does for MS-DOS programs that do the same thing. (That is, open `"lpt1"` as a file and write to it.)

We have added the necessary code to the native windows version of `filePro` to use the Windows printer routines (ie: `OpenPrinter`, `StartDocPrinter`, etc.) which do respect the Windows spooler. However, the spooler is also limited to those printers defined in the printer control panel. Therefore, we have made it a requirement that, in order to use the Windows spooler, you must prefix the `filePro` destination with `"win:"`, as in `"win:lpt1:"`. The rest of the destination must be the exact port name or printer name as you have defined it to Windows. So, if you have a printer attached to LPT1 that is named `"HP DeskJet 870Cse"`, you would use either:

`win:lpt1:`
or `win:HP DeskJet 870Cse`

If you have a network printer `"\\server\printer"` that is captured to LPT2, called `"Bob's printer"`, and the Windows destination is `"\\server\printer"` then you would use either:

`win:\\server\printer`
or `win:Bob's printer`

You could not use `"win:lpt2:"` as `"lpt2"` is not the destination that Windows knows the printer by. (Though you could use `"lpt2"` without the `"win:"` and go directly to that port without the spooler.)

Remember: You can only use the exact port name or printer name that Windows uses. Anything else will result in a "the parameter is incorrect" error when `filePro` tries to open the printer.

4.8.09K Release Notes

4.8 Unix

Fix problem with "-pq"/filename attempting to execute filename, rather than output to it.

Native95

Fix problem of occasionally missing release of Alt when CapsLock down, causing Windows CapsLock bug workaround to be disabled.

Native95

Enhance video I/O by enabling video buffering as under Unix.

Native95

Fix problem when pressing shift-numpad key twice in a row would act as a Break.

All

Fix @FN length problem, where it was truncated to 10 characters in some places in filePro.

All

Fix problem with SQRT(0) returning "/D0". Now returns "0" as it should.

All

Fix ddir problem emptying indexes when nodesize > 1K.

All

Fix fuzzy browse lookup problem when scanning on associated field, where first instance would always be used.

DOS/Native95 Network

EXPORT ASCII -A truncated file on network version.

All

If filePro cannot read the full lockfile, an error is generated.

All

DECLARE will now correctly give a syntax error if you try to declare a variable that starts with a digit.

All

If an invalid edit name was given to DOEDIT(), the function returned garbage. It will now return a null field.

4.8.09 Release Notes

Changes from 4.8.07 to 4.8.09

All

If PFWGT0=ON, then report will now print headings and grand totals.

All

Change HTML :BA for <BASEFONT> to HTML :BF, as :BA is already used for <BASE>.

Unix

Fix Unix code for HTML and JSFILE :CR when checking for leading "/".

(Would prepend \$PFSEVRROOT even if it started with "/".)

All - QuikStart (rcabe)

It was possible for an array reference to return "bad assignment at position indicated" even on valid references, at runtime.

All

If you are inserting a duplicate key, in a record higher than any of the other records with that key, and the block in the index that stores that key is continued into another block, but it is not this key that is continued, the key is inserted into the continued-key value rather than the correct key value. This can cause "deleted key not found" errors.

Unix

Fixed Unix OpenDir()/NextDir() problem of not returning filenames in the same format as the DOS version.

Unix

Fixed Unix OpenDir()/NextDir() problem of not having the dates line up correctly.

Unix (SCO/iBCS2 only)

Fix occasional "too many open files" on systems with NFILES > 64

All

Add PFUFLAG=ON to *report to force "-u" operation.

All

Add PFBLANKOV=ON to cause math arithmetic involving blank dates to return "/OV" (which is apparently how the Sun version used to work).

All

If you are browsing on a demand index, and use a selection set which causes no records to be selected, pressing up-arrow generated an "invalid index" error.

All (cosmetic)

dxmaint's list of automatic indexes overwrote the filename/indexname info on line 20. Move dialog box up.

All

If you have a shared index block that is continued into the next block, but the continued key has only a single instance of the key within this block, and you delete that key, the rest of the continuation chain is left dangling. This can cause "delete key not found" errors.

All

New variable (since 4.8.01, but missing from the prior readme) called PFCONFIG=pathname allows you to specify the full path to an alternate filePro configuration file.

4.8.07 Release Notes

Unix

Fix problem with Break/DEL messing up screen on some systems.

4.8

Still had a bug in GETNEXT if "k=(expr)" was used in lookup.
(Would get "invalid field" error.)

All

Fixed fuzzy browse lookup problem where duplicate key values would display the first record repeatedly, if using a 4.5-style index to assist the browse.

4.8

Fix HTML :DI and HTML :SP crashes.

All

Fix problem in dxmaint if demand index info was too wide, dialog box would wrap around screen.

Native95

Fix keyboard problem where some keystrokes would cause a second (invalid, keyval=1) keystroke to be seen. (Would cause "press any key to continue" and WAITKEY to return prematurely.)

4.8.06 Release Notes

Native95

Handle Ctrl-Break correctly, in addition to Ctrl-C.

Native95

Fix directory read problem in xfer, so can now transmit as well as receive.

DOS

Change "BREAK" to "Ctrl-C" in prompts.

All

PFBLXBUILD=2 didn't handle date field correctly if you mixed "/" and "-" and separators.

All

Add PFF6PROMPT=OLD to turn off the fix of taking @WBL processing into account when displaying the F6 prompt in *clerk.

Native95

*report wouldn't accept ENTER at the index prompt if PFME=ON.

Native95

System command could leave keyboard in funky state, depending on the program executed.

Native95

*cabe didn't wait after displaying duplicate field definitions.

Big-endian systems (IBM RS/6000 and Sun Spare)

Deleting the last key from an index leaf node could corrupt the key count in the parent node, resulting in either "deleted key not found", "IOT trap", or "segmentation violation" on the next access to the same parent node.

All

Fix problem with ADDMONTH() not returning "/OV" if 2-digit-year result is out of range.

All

PFBLXNODESIZE=n will have dxmaint build indexes with a nodesize of nK bytes (1 <= n <= 63) rather than the default value calculated by filePro.

4.8

PFBLXBUILD=2 had a bug that would build a node incorrectly if there was exactly enough room at the end of the block for 1 new non-dupe key.

All

If a processing table had more than 32K of literals, references to those beyond the 32K boundary weren't accessed correctly.

4.8

PFPOSTPRINT=cmdline will execute "cmdline filename" after any printout or hardcopy, when printing to a file.

Native95

Microsoft's keyboard input routines for native console apps do not support Alt+numpad entry. Added code to allow users to use this method.

All

Don't lock demand indexes in *clerk's index menu, unless the user selects that particular index.

All

Add HTML :FN to set attributes. Optional value. Flags: S11CO1

SI SIZE
CO COLOR

If the optional value is given, it is output as text, and a closing
 is automatically generated. The following generate identical
output:

```
HTML :FN :SI "+2"  
HTML :TX "This is big text."  
HTML :FN-  
and  
HTML :FN "This is big text." :SI "+2"
```

both generate

```
<FONT SIZE="+2">This is big text.</FONT>
```

Add HTML :IM to set attributes. No value. Flags:
SR IAT1AL1HI1WI1BO1HSI VSI USI1SI

SR SOURCE
AT ALT
AL ALIGN
HI HEIGHT
WI WIDTH
BO BORDER
HS HSPACE
VS VSPACE
US USEMAP
IS ISMAP

Add :DT flag to HTML :CR to add DOCTYPE tag:

```
<DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
```

Add HTML :IS for <INDEX>. No value. Flags: PR1

PR PROMPT

Add HTML :BA for <BASE>. No value. Flags: HR1

HR HREF

Add HTML :ME for <META>. No value. Flags: HT1NA1CO1

HT HTTP_EQUIV
NA NAME
CO CONTENT

Add HTML :LN for <LINK>. No value. Flags: HR1RL1RV1TI1

HR HREF
RL REL
RV REV
TI TITLE

Add flags to HTML :FO <FORM> tag. New flag: EN1

EN ENCTYPE

Add flag to HTML :HR <HR> tag. New flag: NS0

NS NOSHADE

Add flags to HTML :AN <A> tag. New flags: RL1RV1TI1

RL REL
RV REV
TI TITLE

Add HTML :AD for <ADDRESS> tag. Optional value. Flags: none

If value given, its text is used, and </ADDRESS> automatically closed.

Add HTML :PR for <PRE> tag. Optional value. Flags: W11

W1 WIDTH

If value given, its text is used, and </PRE> automatically closed.

Add HTML :DI for <DIV> tag. Optional value. Flags: AL1

AL ALIGN

If value given, its text is used, and </DIV> automatically closed.

Add HTML :BQ for <BLOCKQUOTE> tag. Optional value. Flags: none

If value given, its text is used, and </BLOCKQUOTE> automatically closed.

Add HTML :UL for tag. No value. Flags: TY1CO0

Add HTML :OL for tag. No value. Flags: TY1ST1CO0

Add HTML :DL for <DL> tag. No value. Flags: CO0

TY TYPE

ST START

CO COMPACT

Add HTML :LI for tag. Optional value. Flags: TY1VA1

Add HTML :DT for <DT> tag. Optional value. Flags: none

Add HTML :DD for <DD> tag. Optional value. Flags: none

TY TYPE

VA VALUE

Add HTML :CE for <CENTER> tag. Optional value. Flags: none

If value given, its text is used, and </CENTER> automatically closed.

Add HTML :SP for tag. Optional value. Flags: none

If value given, its text is used, and automatically closed.

Add HTML :MA for <MAP> tag. No value. Flags: NA1

Add HTML :AR for <AREA> tag. No value. Flags: HR1SH1CO*NO0AL1

NA NAME

HR HREF

SH SHAPE

CO COORDS

NO NOHREF

AL ALT

4.8.05K Release Notes

Changes from 04.08.05 to 04.08.05K

Native95

Handle Ctrl-Break correctly, in addition to Ctrl-C.

Native95

Fix directory read problem in xfer, so can now transmit as well as receive.

DOS

Change "BREAK" to "Ctrl-C" in prompts.

All

PFBLXBUILD=2 didn't handle date field correctly if you mixed "/" and "-" and separators.

All

Add PFF6PROMPT=OLD to turn off the fix of taking @WBL processing into account when displaying the F6 prompt in *clerk.

04.08.05 Release Notes

All

If you define a fuzzy browse lookup, and tell filePro to display less matches than there are lines in the browse window, it used to fill the top of the window with false matches. Now it will simply leave the bottom of the window blank, as it should.

All

If you use the lookup editor and specify a non-existent qualifier in the filename, an error message appeared with erroneous filenames.

Native95

drumenu only allowed 8 characters for "set/change filename".

All

Browse lookup wizard didn't leave enough room after "popup screen name" field for row and column prompts. (Wasn't noticeable unless screen name was >6 characters. More noticeable in 4.8, which allows 25-character screen names.)

All

Fix "read-node 0" message if getnext done on read-only lookup.

[04.08.04K3]

Native95

CURSOR=LINE now works in native95 versions.

Native95

CURSOR=nnn/PFCURSOR=nnn sets the size of the cursor as a percentage of character height. "nnn" must be from 1 to 100. CURSOR=LINE is the same as CURSOR=10.

All

Configuration editor had limit of 256 lines. Increased to 1000.

All

Attempt to use aggregate functions (TOT, AVG, etc.) in automatic processing could crash filePro.

04.08.04 Release Notes

Unix *clerk/*report

READSCREEN()/READOUTPUT() always returned null.

All

Fixed index corruption problem. (File with hundreds of duplicates of the lowest key in the file, with a 2-level b-tree, and adding a record with a lower key, requiring that the index become a 3-level b-tree at the same time, could cause a 'read_node address 0' error.)

All

Memory leak in index routines fixed. (Did not affect 4.5)

All

Lookups with "... r=(expr) -n" could get erroneous syntax errors.

Native95

Filenames can contain "."

All

POPUP command would crash filePro if screen name longer than 10 chars.

All

(Cosmetic) Debugger would wrap processing table name if longer than 14 chars.

All

Protected lookups in CALLED processing tables would not automatically be released when called from normal input processing. (If called from when processing, such as @key or @update, the locks were released.)

All

If the default printer wasn't one of the first 9, pmaint would reset PFPRINTER to the first defined printer.

04.08.03 Release Notes

Memory Leak

It is IMPORTANT to re-index all filePro files after updating to this release.

04.08.02 Release Notes

Lookup wizard

If a non-existent filename (or "(xx)" format) is specified, pressing F6 to display a list of indexes would crash the program.

clerk/report

Don't blow up on "HTML:CR" if file cannot be created.

New processing statement

"If: NOT HTML"

Tests if the last HTML/JSFILE processing statement failed.

New processing function

xx = HTMLERRNO()

Returns an error code for the last HTML/JSFILE statement.

Zero means the statement succeeded. "1" means that the specified document-id is not an open document. A negative number is the system error number.

Processing debugger

Debugger no longer affects @SK.

Install

Wouldn't read free space correctly if PFDIR was set but PFDATA was not.

dreport/rreport

Wouldn't allow long filenames for output formats.

Native Win95 programs

Eliminate busy-wait problem in keyboard input routine.

DOS/Network

Display correct text for "0x20 error" and "0x24 error" messages. (As well as all other errors from 0x12 to 0x24)

0x20 = "Sharing violation"

0x24 = "Sharing buffer overflow"

dclerk/rcclerk

Check for @WBL processing when deciding which "F6" prompt to display.

New flag to lookup statements

"-w" flag will cause protected lookups to locked records to fail rather than wait for the record to be unlocked.

New processing statement

"If: LOCKED(lookupname)"

Tests if the specified lookup failed because the record was locked. (Requires new "-w" flag on lookup, in addition to the "-p" flag.)

File name selection

The list of filenames will have the column width shrunk to the longest existing filename. This will permit more filenames to be displayed if you don't use the longer names.

Unix ddir

"Kill data / retain formats" set the execute bit on indexes.

PFNOBOX=ON

Wasn't respected everywhere.

New environment variable

PFDIRFILTER=ON

Turns on the filter that verifies that only directories appear in the filePro filename list. Some Unix users had serious slowdowns with the filter enabled. (Default: OFF)

dmocdef

In box move/copy mode, display prompts for what to do.

Unix

Executing SLEEP multiple times in processing could cause filePro to crash with "alarm call".

dcabe/rcabe

Pressing F10/Help now keeps the cursor position upon return.

dcabe/rcabe

Pressing F6 within the lookup editor will respect any qualifier already specified in the filename.

04.08.00 Release Notes

NEW Environmental variable that didn't make it to the docs.

PFIXGT=ON

This will allow *clerk to do a next-greater-than if no exact match is found when selecting through Index Selection.

PFSCC=ON

This will enable the "!scc" shell-escape within dclerk and rclerk, which has been disabled by default.

-ro and PFREADONLY=ON

These features have not been fully tested and may not work for your specific application.

04.05.08K6 Release Notes

PFBACKGROUND=OFF

Turns off *report/dxmaint "-bg" and "!g" to enter background.

PFBLXBLANK=OFF

Tells filePro that a null key in a lookup on a 4.5-style index should find the lowest key, which was the default 4.5 behavior until now. Default of ON matches an entirely-blank key, which is the behavior of 4.1-style indexes.

DOS Fixed PFKEYTAB=DOS use of Ctrl-Up and Ctrl-Down. (Actually, any key combination whose scancode \geq 0x80.)

UNIX Fixed flooding the keyboard buffer in *clerk browse mode with 'N' and 'P' could cause the graphics line at the top/bottom of the screen to be corrupted.

All Platforms Fix ddir crashing on files with large number of formats. (More than 12 screens full.)

All Platforms ddir will now display alien file info even if alien file doesn't exist.

All Platforms Lookup editor -- If you have backslash codes in the browse lookup, the editor's display was corrupted.

DOS dxmaint and *report will recognize and ignore "-bg" and "-bs" flags.

All Platforms When prompting for a filePro file name, only show subdirectories of the filepro directory, rather than all items (including non-dir's) in the filepro directory.

All Platforms When doing a browse lookup with "-nl" and giving a key less than the lowest key in the file (or "-ng" with a key above the highest key) you were placed at the wrong end of the file. ("-nl" placed you at the highest key, and "-ng" placed you at the lowest key.)

All Platforms fPCopy - Would crash if more than 255 files in .../filepro/filename directory.

DOS fPCopy - Recognize lowercase filenames on systems that allow DOS programs to see lowercase names. (Win'95 workstation networked to an OS2 server. Unknown if network software a factor.)

DOS dscreen would give a permission denied error if you tried to delete the current screen format.

All Platforms If you have a variable-name lookup, and the variable you use to specify the name does not exist, no syntax error was reported, and invalid code was generated.

ie: lookup foo = (aa) i=a k=ky -nx

...and there is no "aa" anywhere else on the prc table.

04.05.08 Release Notes

Fixes for 4.5 style index problems - primarily resulting from changes to index keys inside large (over 10 index blocks) duplicate key sequences.

Fixed bug related to SHOW(row,col) when the message started with @

Expanded processing editor to 9999 lines maximum.

04.05.07 Release Notes

Extended selection. After entering an "xxF" relationship, you could not press DMAP or HELP until there was a valid field number in the "value" column.

If a CALLED process had a syntax error, it was possible for filePro to crash rather than report the error.

If you do a lookup, specifying a qualifier, and filePro needs to rebuild the freechain on that file, it would build it using the default qualifier, not the one specified in the lookup.

The OPEN() function, when passed the "C" flag to create the file if necessary, could create the file without write permission.

The CLOSE() function wouldn't free the closed handle, reducing the available open files.

The LSEEK() function required the (optional) third parameter to be specified.

New environment variable:

PFKEEPPIXVAL=OFF When returning to the "index by" prompt in *clerk, the value previously entered will be cleared.

When returning to the "select index" prompt, the highlight is now back at the last index selected.

If the highlight was not on index "A", you could not press "A" to select it. (All other valid letters would work.)

Demand indexes can only search on the major key. (This has always been the case.) Change the prompts to reflect this.

The "kill data / retain formats" could corrupt 4.5-style automatic indexes.

Numerous fixes for 4.5-style index "deleted key not found" and "node_address" errors.

The PRINTER TYPE command closed the printer, rather than just changing the type.

When entering sort information, the popup window showing the fields could be missing the sides of the border.

When creating a new qualifier in ddefine, 4.5-style automatic indexes would not be created correctly.

The latest version of SCO Unix changed the syntax of the "doscp" command. Xferdos now handles the new syntax. Added "PFXFERDOS=OLD" to allow it to run on systems with the old syntax.

04.05.06 Release Notes

Numerous fixes for "deleted key not found" errors.

Especially for indexes with large numbers of duplicate keys.

(Unix) fPCopy was missing the "rename" program.

(rclerk) The following combination of events could cause rclerk to crash:

- * rclerk is run without a filename on the command line.

- * File has automatic and input processing

- * Do not do anything that displays a record on the screen.

- * Press <break> from the rclerk menu.

(dedef) If you copy edits from another file, you could not test the new edits until you entered update and re-saved the edits table.

(ddir/dprodir) After killing the key/data files, the displayed record count would not reset to zero.

(ddir/dprodir) If you manually edit the fp/lib/config file to contain a line longer than 80 characters, the filePro configuration editor may crash. Length increased to 200. (Printer definitions may require longer than 80.)

(DOS network, dmoedef) You could not delete the output format currently displayed in the editor.

(dxmaint) Dxmaint would not recognize 4,5-style indexes that had been sent via fPTransfer.

04.05.05 Release Notes

If you broke out of update mode on a non-filePro file in *clerk, the record was deleted.

dreport now recognizes "-tf" flag. rreport now recognizes and ignores "-ty" and "-tf" flags.

If index.A is built on multiple fields, and user presses ENTER on one of the secondary fields at the "select index" prompt, clerk would crash. Other indexes were okay.

FIELDNAME(), FIELDLEN(), and FIELDEDIT() can now take associated fields, rather than just real fields. [ie: FIELDNAME(lookup,"a0")]

If you CALL a table with several lookups, and the table has a syntax error, filePro could have crashed.

An invalid demand index would crash ddir.

Multiple index fixes for "deleted key not found" and dxmaint speed.

If you break out of creating a script file in dmakemenu, you cannot break out of the menu.

Script file may not be made executable under Unix.

VIDEO OFF did not always work under Unix when used with PUSHKEY.

04.05.03 Release Notes

- * (SCO Unix) Count the multi-screen console as a single user for licensing.
- *
- * (Unix/Xenix) Fixed problem of @ID/@CB/@UB returning a number rather than the name if there are too many open files.
- *
- * Demand indexes built with associated field as a non-major key did not set @AF correctly.
- *
- * (Unix) PUTENV command prevented further use of the SYSTEM command
- *
- * The F6 popup list of indexes in dcabe's lookup editor wrapped across the right side of the screen.
- *
- * There was no space after the field name in the "index by" dialog if the field name was 40 characters long
- *
- * Browse using a demand index or a pre-4.5 automatic index would repeat the first record indefinitely.
- *
- * (DOS/LAN) If you SHOW a message starting with "\rK" (no spaces around the "\r") the program would abort with a "DOS/4GW error" message.
- *
- * (DOS/LAN) If you install filePro into a subdirectory several levels deep (such as D:\PROGRAMS\DOSAPPS\FP) and use forward slashes in your menu command line ("/fp/dclerk") the program will abort immediately with a "DOS/4GW error (2001): exception 0Dh" message. This is a bug in the DOS/4GW startup code.
- *
- * (Unix) F8/Options dialog in dmoedef required a triple-break to cancel.
- *

04.05.02 Release Notes

- * 04.05.02 12/03/96
- *
- * Adding records to a non-filePro file could result in erroneous
- * "deleted key not found" errors.
- *
- * In add records mode, the freechain was locked during execution of
- * automatic processing. If user input was needed, this would lock all
- * users out of adding records until the input was supplied.
- * Automatic processing is now run after the freechain is released.
- *
- * It was possible to get an "error 2001 (page fault)" under DOS or a
- * "segmentation violation" under Unix if all the following conditions are met:
- * 1 - You have an index built on an associated field in the current file.
- * 2 - You are in index mode on that index.
- * 3 - You do a lookup to yourself based on the same index.
- * 4 - This is the first time executing the lookup.
- *
- * DOS/LAN The screen did not redraw correctly if you canceled the options
- * dialog in printer maintenance.
- *
- * The column offsets could be improperly calculated in a popup window
- * if the field length caused the field to extend past 80 columns.
- *
- * Deleting the highlighted record in a browse lookup window with the pkeep
- * option set could cause the new highlighted record to be displayed twice.
- *
- * Using inverse video codes around a key code at the start of the message
- * in a show statement could result in a "page fault" under dos or a
- * "segmentation violation" under UNIX. This same bug existed in 4.1, but
- * the 'damage' was undetectable.
- *
- * There was a token table incompatibility between tables produced by rcabe in
- * 4.5 and 4.1 if there was more than one import or export statement in the
- * table.
- *
- * If you moved an index to a different drive, everything was okay until
- * you went to dxmaint to rebuild it. dxmaint would build it back on the
- * default drive (and not get rid of the one on the other drive).
- * This caused "duplicate file" errors. Dxmaint now finds and rebuilds
- * the existing index.
- *
- * -bg -bs flags functionality restored in UNIX *report
- *
- * fixed cosmetic problems in index search dialog
- * also now use index comment if only one field in index
- *
- * modified index search dialog
- * if PFDLGENTER is TRUE
- * ENTER now performs a SAVE only if exiting last field
- * or if field just exited was empty, otherwise is does a
- * CDWN (cursor down)
- *

Automatic Processing Tables

When compiling processing tables that rely on dummy fields defined in an automatic processing table having a name other than " automatic " (UNIX) or " auto " (Windows), you need to specify the automatic processing table name with the " -y " flag when defining the processing table.

Example:

If you are using dummy field " aa " to keep track of a subtotal in a report1 and you are going to run rreport with a different automatic processing table named " autotot " (which defines dummy variable " aa "), then you must compile the report processing table as follows.

```
rcabe filename report1 - y autotot
```

Or, put another way:

If you are using "aa" to subtotal, and are going to run rreport with an automatic processing that does not define "aa", then you must also compile with an automatic table that doesn't define "aa". (Or vice versa.)

Of course, your best bet is to simply compile with the same automatic table.

See PFZEROLENWARN to turn off assigning to a zero length field warning

Blinking Text

Windows does not support blinking text for native console applications, such as the native Windows version of filePro. Instead, the attribute bit that DOS uses for blink is used for high-intensity background color by windows.

Note: DOS can set the video to either mode, but it defaults to blink. Windows will set it for high-intensity background, disabling the blink feature for use by filePro.

Browse and Creation Password

Keep in mind that a browse will not display fields in the current file if the field is not included on the screen when the filePro file has a creation password.

Button Problem

When using fileProGI, buttons either disappear or are hidden when using the " cls ". This only occurs with user prompts for browse lookups, and only appears to be a problem if you start the processing with:

```
cls("something") ; show "string with button definitions"
```

The simple workaround is to split the cls() into its own line:

```
cls("something")  
show "string with button definitions"
```

Note: "show" can also be "show (row,col)", "show ctr", and any of the other show variations.

DOS4GW

***** IMPORTANT For DOS and LAN users (non-Native versions) *****

You must make sure the supplied DOS4GW.EXE is in your PATH. We are also including with this distribution three utility programs for DOS4GW as well as a short text file DOS4GW.DOC which describes their use. These files will be placed in your program directory by the installation program. You should move them to whatever location you desire.

Because of the way the DOS4GW 386 DOS extender works - the maximum allowable filePro command line has been reduced from 127 characters (the maximum allowable under DOS) to 120 characters. This involves all command lines, whether run from a DOS prompt, or system command, or menu, or batch file, which run filePro versions 4.5 through 4.8 programs. It does not affect command lines that run other programs.

Note: The above does not apply for Windows Native versions of filePro. If you are using these versions, the DOS4GW program and related documentation can be removed. Non-Native distribution was discontinued with version 5.0

Emulation Problems on LINUX

In some cases, graphics characters and colors may not be properly displayed for filePro. This is due to system configuration and can be corrected by doing the following.

Edit this file `/etc/sysconfig/i18n` to look like this

```
LANG="en_US.iso885915"  
SUPPORTED="en_US.iso885915:en_US:en"  
SYSFONT="lat0-sun16"  
SYSFONTACM="iso15"
```

Logout and then back in and make sure your term is linux.

fppath

Contents of this section

Description

Environment Variables [[link](#)]

Path Environment Variables

fppath

Description:

FilePro uses many environment variables to alter the way it functions. These variables are recognized by the various filePro programs. They are stored in the environment as would be any environment variable. FilePro environment variables can also be stored in a special configuration file you can maintain with an editor found in filePro. This file is called "config" and it lives in the "fp/lib" (fp/lib) directory. It is edited with the Configuration Editor found within the FilePro Directory choice on the filePro Plus Main Menu.

Path Environment Variables

There are four filePro environment variables that cannot be set in this file. (Actually, PFDSK may be set in the filePro config file, but you would do well to treat it like the other 3 PATH variables and set it outside of the config file.)

These 4 variables are loosely called the filePro PATH variables since they tell filePro programs where to find the filePro program directory "fp" (the filePro executables and configuration files are in this directory), and the "filepro" directory (your applications and data are in this directory).

PFDSK	Sets the filesystem
PFPROG	Sets the program directory (by appending "fp" or "fp").
PFDATA	Sets the mount point of a filesystem
PFDIR	Sets the data directory (by appending "filepro")

To locate the filePro program directory (fp):

```
Unix - $PFPROG/fp
Windows - %PFPROG%\fp
```

To locate the filePro data directory (filepro):

```
Unix - $PFDATA$PFDIR/filepro
Windows - %PFDATA%%PFDIR%\filepro
```

Then filePro will scan for key/data/index files by looking in the directories listed in PFDSK, substituting each entry for PFDATA above.

/etc/default/fppath

Under Unix, there is a special filePro default file (/etc/default/fppath). Under Windows, the fppath file will be located in the /fp parent directory. This file contains four lines and may look something like the following:

```
/u/appl
/u
/appl
9eaNE%WWFYbfeL
```

This file represents the three variables, PFPROG (first line), PFDATA (second line) and PFDIR (third line). If they are set, the environment variables themselves **OVERRIDE** this file's contents, but in their absence, this file dictates where filePro programs look for their own libraries and where these programs look for your data files. Again, the first line is where the programs live, the combination of the second and third line show where the filePro directory lives. You append "fp" to the first line and "filePro" to the second/third lines separated by a /.

The fourth line of this file is an encoded site password. This password is set or unset on the filePro Utilities Menu. ("uti"). If this line is blank, there is no site password.

NOTE: When setting a site password, keep in mind that this will affect your processing tables so make sure that you don't lose the assigned password and when moving from one system to another that you set the same site password.

Hide Indexes

5.0 Enhancement - When using the index maintenance program, you can specify which indexes you want to hide from users while in the " *Index Selection* " menu for " *Inquire/Update/Add* ". After selecting an index, press [**F8**] to access the extended features " *Index options* " popup screen. You should see a screen as depicted in figure 1.



Figure 1 - Index Options

Press " Y ", [**ESC**] to hide the index for " *Inquire/Update/Add* ", or press " N ", [**ESC**] to unhide the index.

This may help if you find that indexes are rebuilding slowly.

To speed up index building, set the following config var:

`pfnumixbuild=10000` ($10 \leq n \leq 128000$)

`PFBIXBUILD` shouldn't be required to be set

`PFNUMXBUF` can also be set but isn't usually required.

`PFOLDIX` should not be set

pfdisk

This is a path environment variable that should be set in all cases to avoid searching all available disk drives and to avoid errors when searching for filePro files. Refer to the Environment Variables section of the manual for syntax. By setting "pfdisk", you can avoid searching the floppy drives.

License Manager

Version 5.0.15 / 5.6.00

License Manager is implemented to allow for better control of user counts and to address licensing requirements for host and mirror/backup servers.

Environmental variable PFLMHOST tells filePro where the license manager resides. The values of PFLMHOST must match the register license (licfp.dat) for the machine that it resides and runs on.

Syntax

PFLMHOST=address:port

Most of the licenses will use the default port of 6556. If the address is the local machine and the port is the default, PFLMHOST does not need to be set.

Example:

Machine 192.168.0.1 is the host and license server. PFLMHOST need not be set.

Machine 192.168.0.2 is a backup/mirror machine.

Set PFLMHOST in the starting environment or the config file for filePro of machine 2 as follows.

PFLMHOST=192.168.0.1:6556 (on the backup/mirror machine)

filePro will operate on each of the machines using the license server on machine 1.

NOTE: User counts per product will be accumulated as if all programs were running local. Should the host server (machine 1) be "out of service" machine 2 will run in a seven (7) day grace period using the backup/mirror (machine 2). If you install a new machine, you will need to request a new license with new machine information. By using the license manager, you can utilize one (1) license on both the host and backup/mirror machine and still be in compliance with the filePro licensing policy of one license per server since the 2nd machine is a mirror of host machine and not an active machine. Should the backup/mirror machine be put into an active non-mirror/backup status, a valid license must be obtained from www.fptech.com website or by calling our sales office at 1-800-847-4740.

License Server Program

The license server will look for the license file %pfprog%/fp/lib/licfp.dat. Make sure that you have downloaded and saved the license file in the right location before attempting to start the license server program.

OS	Program Name
NIX	fplmserver
Windows 98	fplmserver.exe
Windows NT	fplmservice.exe

Starting the License Server Program

You can start the license server program from a terminal window command prompt on NIX and Windows systems by changing to the directory where the license server program resides and starting the license server program.

Examples:

AIX / LINUX / UNIX

```
cd /u/app/fp
nohup ./fplmserver&
```

Windows

```
cd \app\fp
fplmserver
```

Although the above will start the license server, this would only be used for testing or troubleshooting to make sure that the license server will start and that you have a good license in the right location e.g. %pfprog%/fp/lib/licfp.dat.

LINUX/UNIX

For LINUX and UNIX systems, you will typically want to autostart the license server by creating a script that can be launched during the system boot process or by including the required lines in an existing script that is called during the boot process.

```
cd /u/app/fp
./fplmserver
```

A sample script named "startfplmserver" is included in the filePro distributions for UNIX and LINUX systems in the ~fp directory containing the above lines.

Modify the first line of the sample script to change directory to your filePro program directory and then call this script from a boot startup script. The location and filename of boot startup script varies depending on the Operating Systems version and particular distribution. Refer to the operating system documentation for details on how to autostart scripts. Some typical locations/filenames for autostart scripts are provided in the following table.

Redhat/Fedora/SuSe	/etc/rc.d/rc.local
Gentoo	/etc/conf.d/local.start
SCO Unixware	/etc/rc2.d/userdef
SCO OSR5 and OSR6	/etc/rc.d/8/userdef

Windows NT, 2000, 2003 and XP

Although you can use "fplmservice.exe" to start the license server on Windows NT type operating systems, the license server is normally installed as service name "fplm_service" during the filePro install and uses the "fplmservice.exe". Since the license file "licfp.dat" will normally be copied after the installation is complete, the service will not automatically start after the installing filePro on new installations. Once you have placed the license in the appropriate directory e.g. ~fp/lib/licfp.dat, you can start the license server by going to the Windows Control Panel, Administrative Tools, Services. Find the fplm_service and click on "start". You should see "started" in the service status.

Installing the filePro License Service from a command line using "sc.exe"

In some instances, you may have a need to install the fplm_service as a service on your server without doing it during the filePro install. This is true for NETWARE installations where the service would be installed on a Windows box that serves as your filePro License Server or in cases where you are temporarily using a Mirror/Backup server as your filePro License Server.

Also, some anti-virus software will prevent the license service "fplm_service" from being created during the install. Use this section to install the service when the "fplm_service" is not shown as a service in the Windows "Service" panel.

Microsoft provides a command line utility named "sc.exe" that allows you to start a service.

To install the filePro License Server from a command line, do the following.

```
sc create "fplm_service" binpath= "c:\fp.56\fp\fpplmservice.exe c:\fp.56\fp"
```

where "fplm_service" is the service name and "binpath" contains the full path to "fpplmservice.exe" executable and the path parameter "c:\fp.56\fp" identifies the location of the \lib directory containing the license file "licfp.dat".

Important: You need a space after each equal sign when using the sc.exe utility program

The "sc.exe" allows you to manage the service based on the parameters that you pass to it. When executing sc.exe without parameters, the syntax and acceptable parameters are identified for things like *Stopping, Pausing, Removing* and *Querying* the service.

Workstations - Connecting to a Windows NT License Server

Once the License Server is running, you will need to identify the server in your workstation startup file(s). The PFLMHOST variable is used to identify the server name or the server IP address as follows.

```
set PFLMHOST=servername:6556
```

where servername is the Windows computer name where the filePro license server is running and "6556" is the reserved port number.

```
set PFLMHOST=xxx.xxx.xxx.xxx:6556
```

where xxx.xxx.xxx.xxx is the statically assigned IP address of your server and ""6556" is the reserved port number.

Recommend using the "servername" rather than IP address since the license server will be found whether the server IP address is either statically or dynamically assigned.

Stopping the License Server Program

LINUX and UNIX systems

%pfprog%/fp/stopfpplmserver contains the following.

```
kill `cat /u/app/fp/lib/licfp.pid`
```

Note: Gentoo LINUX, add stopfpplmserver to file /etc/conf.d/local.stop to allow you to stop the server.

Windows NT

Go to Control Panel, Administrative Tools, Services and select fplm_service. Click on "Stop" to stop the service.

License Server Logs

Located in the %pfprog%/fp directory is a directory called logs. In logs is a file for server logging configuration.

```
servlog.cfg
```

Default:

```
# type, level [,filename] [,max_size]
# type could be CONSOLE,FILE,ROLLING,or DAILY
# FILE,ROLLING, and DAILY require a filename
# ROLLING requires a maximum size
# level could be FATAL,ERROR,WARN,INFO, or DEBUG
# lines beginning with # are comments
# blank lines are skipped
# keywords are case-insensitive
# forward slashes (/) must be used for path separators
CONSOLE,FATAL
# ROLLING,DEBUG,server.log,60000
NEWFILE,DEBUG,server.log
DAILY,WARN,server_warn.log
```

ROLLING indicates that file will be renamed to a backup name (using the current timestamp) when it exceeds the specified size (60000). A new file with the specified name will then be opened.

NEWFILE indicates for server to remove the file and create a new log file when the license server is started.

By changing the name of servlog.cfg file to servlog.cfg.sv the server will eliminate creating any log files.

IMPORTANT: An update or upgrade of filePro will overwrite the servlog.cfg file or reinstall it if it does not exist or has been renamed.

Turning off Logging

Once you are comfortable with the licensing you can turn off the logging by commenting out the DEBUG line in the log file. You should leave the WARN line though. You can also change the logtype from NEWFILE to DAILY. NEWFILE causes the server to restart the log file each time the server is started. Daily causes the server to restart the log file each day when the server is continuously running.

Take these steps to isolate a license problem when running filePro on Windows.

Go to the services of windows to stop or make sure the service is stopped.

Start - Control Panel - Administrative Tools - Services

Look for the entry called fplm_service

If the service is not already stopped, Right Click on fplm_service and select Stop then proceed to the next step.

Go to the ~/fp/logs directory where filePro is installed.

Edit the file called servlog.cfg

Make sure the lines towards the bottom are not commented out

```
NEWFILE,DEBUG,server.log
```

```
DAILY,WARN,server_warn.log
```

Save the servlog.cfg file

Then go to the services of Windows to try to start the license service.

Start - Control Panel - Administrative Tools - Services

Look for the entry called fplm_service

Right Click on fplm_service and select Start

If the service does not start, read or send the logs that are located in the ~/fp/logs directory of your installation. These logs will tell us what is not proper about your license or configuration.

Map the drive of the server to the workstation. C on the server may be F on the workstation.

Copy the FullDev.bat located in the c:\appl directory on the server to WS.bat for use by workstations.

Edit the WS.bat and Add after the first line

```
set PFLMHOST=88.118.118.88:6556
```

NOTE: Replace the IP address with the correct one for your server.

This tells the Workstation to use the machine at that IP address as the server license manager and to connect to port 6556 (the default).

Also change all references of the server Drive letter to the new Workstation Drive letter. C on the server may be F on the workstation.

Change the short cut (or create) to use the WS.bat instead of the FullDev.bat

When a Windows workstation cannot connect to the License Manager, it means one of 3 things.

1. Make sure the license server is running on the server. Does filePro run from the server properly?
2. Make sure the bat file that launches filePro on the workstation has the appropriate environment setting of 'set PFLMHOST=serveraddress:port' The port is set to 6556 on all filePro licenses.
3. Make sure that the port (6556) is not blocked by either the operating system or by a firewall.

On the system where the license manager is running, do:

```
telnet localhost 6556
```

If it connects (screen clears), use ctrl-] (control right-bracket) to get to the telnet prompt, and then "q" to quit.

Then, from the workstation system, use:

```
telnet <p_address> 6556
```

If both these telnet tests work properly, then the workstation connection should be made to the license server.

Syntax

Contents of this section

- Description*
- Clear Screen Example*
- Variables*
- Literal Values*
- Use of []*

Description:

Syntax is the set of rules by which a program is made to operate properly. Any function, command or feature can be considered a " program " that needs to be operated correctly - by its rules.

Clear Screen Example

For example, the syntax for the "CLEAR SCREEN" function used on filePro processing tables is:

cls	Clear the entire screen.
cls(s)	Clear the screen starting from line # "s" to the bottom of the screen.
cls(s,n)	Clear the screen from line # "s" for "n" lines.

In other words, cls("10","5") would clear the screen starting at line # "10" through line # "14".

Variables

Generally, variables are used to indicate places where you will substitute your values. For instance, the " s " represents your " starting line " and the " n " represents your " number of lines to clear ". You can usually determine what syntax instructions mean using your common sense and intuition. They are not meant to be obscure, their purpose is to be very clear.

Literal Values

In general, within filePro processing tables, literal values are placed inside quotes. If you mean to use the number " 10 " as in the above example, it must be in quotes. If you mean to use the contents of field 10, you would just use 10. All filePro syntax is written with this as a basic assumption.

Use of []

Items within [] s are optional, i.e., the function will work with or without your substituting values for these items.

Example:

mdy[y][/] shows that "mdy", "mdyy", "mdy/", or "mdyy/" are valid constructions.

by
Jim Asman
Spectra Colour Svcs, Ltd.
Vancouver, BC

jim@spectra.winsey.com

The filePro directories "ljet" and "widths" described in this article are on The filePro Bible CD.
They are in the directory ...vpfiles, and must be copied to your system in order to be used.

Introduction

The modern laser printer can add much to filePro output but, as the "hplaser" print code table is ostensibly having the laser emulate a line printer, we really need a new table. Aside from deciding on a structure to make some logical sense out of the 256 codes in filePro 4.1, programmers must take a whole new approach as to what the printer can do for them.

This article is an attempt to build a useful print code table for both Hewlett-Packard LaserJet III and LaserJet 4 printers (while keeping an eye toward future developments). Because the LJ4 has many more internal fonts than the LJ3, and the LJ4 fixed pitch fonts are now scalable with different typeface numbers than those on the LJ3, I have worked up separate tables for each printer. This article is NOT intended to be a tutorial on the use of the LaserJet, as HP has material to do that very well, and the reader is well advised to get the PCL Technical Manual, if that hasn't been done already. You can use it quite effectively on a "need to know" basis, and get into it as deeply as you wish.

It is probably safe to assume that the majority of filePro applications running on a laser printer today are using either the Courier, lp.16, or another monospaced font. The printers, though, are capable of much more, and with a little diligence you can make your filePro reports and forms look more like they came from a word processor than a database. In the case of the LaserJet Series III and more recently the LaserJet 4, using the internal scalable fonts creates possibilities that we would never consider using a dot matrix printer, and further, the added graphics capabilities by themselves can change the "look" of the output entirely. If you want a fat line, a skinny line, a double ruled box, just name it and you can probably do it without very much effort. The HP-GL/2 plotter language can simplify drawing boxes and other graphic shapes, but that will have to be a topic for another time.

First Things First

Because of the sheer volume of useful printer codes (and maybe that should be rephrased to the codes that you do use), an orderly definition of the printer codes in the table needs to be established. In using a line printer, most of us probably never embrace more than a dozen or so printer codes in our day to day doings, but with the laser, the current limit of 256 codes on the table with filePro 4.1 is already restrictive, and that will only get worse as new printer features become available.

To this end, abandoning the 54 "universal" codes and starting over from scratch may be worthwhile just to maintain structure in the table itself, but recognizing that many people have built existing applications relying upon the supplied tables, I haven't changed the functionality of any of the codes at the beginning of the "hplaser" table, other than the *printer init* (code #3), which now resets the printer to the control panel settings, sets the PC-8 symbol set for both the primary and secondary fonts, and nothing else.

It makes sense to lay out the printer code table in categories, much like Hewlett-Packard has done with their reference materials. So let's go through it in some order.

Job Level

There are any number of printer codes that we may need to issue to the printer before the first character is ever printed and which are generally only sent once during the print run. I guess we have always thought of this as the printer initialization, and there are a lot of possibilities. We need to reset the printer first, define the page size, number of copies, margin sizes, page orientation, initial font, etc., etc.

All of this can be amalgamated into a single filePro code, but because the number of permutations becomes so large, I suggest that we use a generic *printer init* (code #3) and then get into the specifics with the *form init*. An implicit assumption made through all of this discussion is that we are not relying upon the lp interface script to establish any of the printing parameters, and obviously, if we have a printer reset code in our filePro *printer init*, the goings on of the interface script become academic. Certainly, any printer setup that you can use from the interface script is something that filePro doesn't have to deal with, but I see that script as something largely hidden and soon forgotten as to what it's doing.

Logically, then, we build our *form init* code by assembling other codes from the table. This need not be complex at all, and probably would only use a collection of a few codes put together most of the time. The printer reset code sets all the printing parameters to the values set at the printer control panel. Unless you are the only person using the printer, and maybe not even then, it is wise to assume that the values at the control panel may have been modified by yourself or others and should be set in your initialization of the job. Has anybody set the number of copies at the panel to some number other than one, and not reset it before the next job was printed? I know that I have. To be safe then, our *form init* should contain codes to set the number of copies, symbol set, font selection, paper source, page layout, and simplex/duplex printing, if you have a duplex printer.

Page Layout

```
Take a look at code 55. $1b &12a0o6d3e60F
| | | | |
$1b Sequence start-----| | | | |
2a Letter size paper-----| | | | |
0o Portrait orientation-----| | | | |
6d 6 lpi-----| | | | |
3e Top margin of 3 lines-----| | | | |
60F 60 Print lines-----| | | | |
```

With this code we are telling the printer we have letter size paper, want to print 6 lines per inch, a top margin of 3 lines (1/2 inch), and a text length of 60 lines (10 inches). The PCL-5 language has no definition for the bottom margin as it is implied by the size of the top margin and text length. In this case, starting with an 11 inch sheet, we use 1/2 inch on top followed by 10 inches of text, thus we are left with 1/2 inch at the bottom. So, the bottom margin is really defined by the text length and top margin. Look at the other codes for 8 lpi in portrait and 6 lpi & 8 lpi in landscape mode, and study them until you understand the numbers. It's quite simple really.

Given the relative ease of formatting the page on the printer, I think that all filePro forms and reports should be defined as having the same number of lines to print as there are lines per page. The printer can do the page formatting quite nicely, and it saves filePro from having to send blank lines to fill the page.

Font Selection

Before discussing the selection of fonts, let's define just what a font is, to avoid confusion. Many of us, myself included, will use the words *font* and *typeface* interchangeably but, as the terms are used with a laser printer, they are not the same thing. In fact, the typeface is just one of several characteristics that describes a font. Quoting the HP Technical Manual...

"A font is a set of characters that have similar characteristics. A font has an assigned name, typeface, and is further described by its spacing, height, pitch, style, stroke weight, symbol set, and orientation."

While this may seem apparent, it is important to understand that if we want, for example, a Times 10 point medium upright proportional font, we would need a print code:

```
$ 1b (s1p10v0s0b4101T
| | | | |
Sequence start-----| | | | |
Proportional Spacing---| | | | |
10 pt-----| | | | |
Upright(not italic)-----| | | | |
Medium stroke(not bold)-----| | | | |
Typeface(4101 is Times)-----| | | | |
```

Then later, to get the same thing in bold, we would only have to issue the "bold" code \$1b (s3B). By giving the bold command, we haven't asked the printer to just modify the existing font, but are actually requesting a totally different font. So changing any of the font select parameters is changing the font and, if the printer doesn't have that font, you may get some surprises.

A font is selected by the printer based on its characteristics in the font select command, and the printer will do its best to oblige but, if there is no font that matches exactly, then there is a pecking order of the characteristics that will determine which font is picked. In order of highest priority, the determining characteristics are as follows:

- Symbol Set
- Spacing(proportional or fixed pitch)
- Pitch(cpi--valid only for fixed pitch)
- Height
- Style
- Stroke Weight
- Typeface
- Location
- Orientation

This is all covered quite well in the technical manual, and it is worth the time to study it.

In our printer code table, codes 88-97 contain font selects for all of the LaserJet fixed pitch fonts, while codes 101-128 are font selects for 10 point proportional fonts for all the internal typefaces supported by the printer. Codes are provided for both the primary as well as the secondary font. Primary what?

The printer can maintain two distinct font select tables, with one of them being active at any given time. If you look at the font select codes in the table, you will see the codes for both primary and secondary. They differ only in the start of the code sequence. What you can do is have the two fonts designated and bounce back and forth between them by alternately sending codes 99 and 100.

These two tables needn't have anything in common, but they can. The secondary font could be the same as the primary, but italic instead, or could be completely different in every respect. Do whatever is most advantageous to you.

Symbol Sets

Although many of the LaserJet fonts contain well over 500 characters, we can only access a portion of them at any one time. The symbol set that is selected determines which characters are available and their order at any given time. Symbol sets are only a maximum of 255 characters in length, so you can see that if you only ever used one set, you'd be missing half the fun. The PC-8 set, though, has most of what you are likely to need in day to day applications.

As the symbol set has the highest priority in font selection, you want to make sure that the font you are selecting supports the symbol set that you've chosen, or quite likely you'll get something other than what you had expected. However, if you are using the LaserJet internal scalable fonts, you probably won't have to worry about that, as they support many different sets. On the other hand, the LaserJet internal bitmapped fonts (i.e., courier and lp.16 on the LJ3 and just the lp.16 on the LJ4), support far fewer, and if you were printing a bitmapped font and changed to a symbol set not supported by the font, you will end up with another typeface. This is all in the technical reference manual, but just to raise the level of paranoia ...

Using "vi" or your favorite editor, type in the following, which selects the lp 16 pitch font with the PC-8 symbol set and then changes midstream to the Desktop Symbol Set. For the purposes of this text, I will use "^[" to indicate the escape character, but keep in mind that you would use chr("27") in processing, or \$1b when defining a printer code. To insert the escape character using "vi", type "CTRL-V" followed by the escape key. When entered, it will appear as the two character sequence "^[" on the screen, but it is just a single character in the file. Send the file to your printer and see what you get.

```
^[ (10U^ [ (s0p16.67h8.5v0s0b0T
Carrots are divine
You get a dozen for a dime
```

^[(7Jit's maaaagic.

...B. Bunny

Food for thought, yes?

If you find you have an application where you regularly want to change back and forth from one symbol set to another, set it up with your primary and secondary fonts, leaving the other font attributes the same.

Just an aside here: in the print code table, codes 32-54, which are on the universal table, are various odd characters and others for drawing lines and boxes, some of which are not in the PC-8 set. If you look at them, you will see that the codes for a few, change symbol sets for the one character, and then change back to the PC-8 set. Of course, the assumption is that you are using PC-8 to begin with. If you are using any other set and call one of these codes, you could be in for some surprises after the code is finished; or in the case of those that are in the PC-8 set, you would almost certainly get a character other than the one you planned.

The Problem

Unfortunately, most of the goodness of the laser printer implies the use of proportional fonts and precise positioning of text on the page, but filePro output is designed for a fixed pitch font. I know that I didn't buy a laser printer to print courier fonts. filePro expects that any character position, let's say character 23 on the line, will always be in the same physical location along the line, regardless of what may have preceded. This is fundamental in output formats if fields are to line up in columns. That assumption certainly cannot be made when using proportional fonts. Even mixing 12 and 16 pitch monospaced fonts within a box has unwanted ramifications. So, in all likelihood, any existing output format that you now have which uses a fixed pitch would fall apart at the most fundamental level; i.e., to get the fields themselves to line up on the left, when using a proportional font. Our task is largely to align the fields and to take care of any alignment required within the field. Centering headings and the like also requires some attention.

Hewlett-Packard's PCL-5 printer control language provides us with all the tools we need to make use of the printer's many features, and it is, of course, the basis for the filePro print code table. At this point, let's have a look at why output formats can have problems once we start using the generally nicer looking proportional fonts.

Line Level

First, on a given line, we define any number of fields to be located at fixed positions on the line and, when a report is printed, these fields will all line up on the left, leaving us with columns on the page. Pretty standard stuff. We know that when this scheme is used with a proportional font, the columns won't line up, so rather than rely upon the amount of space previously consumed on the line to establish where a field will be printed, we will issue a code that positions the field at a specific location on the line, independent of what else may be printed on that line.

Field Level

There are three types of fields from our standpoint: *left justified*, *aligned*, and *right justified*. *Left justified* is our typical field where everything is pushed to the left: names, addresses, and the like. An *aligned* field usually would be a numeric field that is aligned on a decimal point or some other character: a dollar amount with a ".2" edit, perhaps. As the name implies, in a *right justified* field, all data is pushed to the right: integer numerics and perhaps text that for whatever reason you want to line up at the end of the word rather than the beginning.

All fields in filePro with fixed pitch fonts are really left justified, and any alignment that is required is taken care of by the "edit" that appropriately pads the field on the left or right side with spaces. It is, though, important that you conceptually understand the differences between them to make it work with the Laserjet.

Using Printer Codes

The LaserJet makes use of a wealth of PCL-5 printer codes to position the printer "cursor", select fonts, draw graphics, and on and on. The thing that makes it really interesting for a filePro programmer is that, aside from the escape character, chr("27"), that precedes every HP printer code, the rest is pure ASCII text. Actually, that is not quite true; the codes to activate the primary/secondary fonts are also non ASCII, chr("14") and chr("15"). This means that we can very easily put a printer code into a variable and integrate it right into our data, if appropriate, put it right on the output format as text, or in a code on the filePro printer table to achieve our goal.

Cursor Positioning

You can consider an 8 1/2 x 11 sheet of paper to be a 2550 x 3300 matrix of addressable dots, which translates to our 300 dpi but, as there are some margins imposed by the printer, our workspace isn't quite that large. We can direct the printer to start printing at any one of these locations, so you can see that cursor positioning is very precise.

Absolute Positioning

Before we go any further, let's get one term or acronym defined. CAP. This stands for "current active position", which simply means the place on the page that the next character will be printed, and is very analogous to the current location of the cursor on your monitor.

The code to position the cursor works like this.

```
^[*p200x600Y
```

This instructs the printer to move the CAP to a point 200 dots along the horizontal axis from the left margin and 600 dots down from the top margin. We can move only to the horizontal position by using ^[*p200X or only to the vertical position with ^[*p600Y. Note that only the last letter in the code is capitalized and that, when the command is combined, characters in the middle of the string are lower case. Further, these are absolute positioning commands, i.e., the position 200X means 200 dots from the left margin and 600Y means 600 dots from the top margin, **regardless** of where the CAP was at the time the command was issued. So ^[*p200x600Y is a unique position on the page. If you issue ^[*p200X, only the horizontal position is changed and the vertical location remains the same, and ^[*p600Y only affects vertical positioning.

Relative Positioning

We can modify the previous example to ^[*p+200x+600Y, which will direct the cursor to move relative to the CAP 200 dots further to the right on the X axis and 600 dots further down the Y axis, or ^[*p-200x+30Y would back up 200 dots to the left and move 30 dots down the page. The only difference is the inclusion of the "+,-" signs in the command. Relative positioning commands can be positive or negative, meaning that you can back up on the current line and overprint, if you wish to do so. It is all very flexible. I should point

out here that, if you issue a positioning command that is beyond the bounds of the printable page, the new CAP will be at the printable margin.

Now how does this help us position our filePro fields? We only have to create filePro printer codes to create tab stops at regular intervals along the X axis. Ideally, we could have a predefined code at every 10th dot or so, but there simply isn't room on the print code table. To cover a 10 inch wide line with codes every 10 dots would by itself require 300 codes, so that is obviously out of the question at this time. The table presented here has a group of codes in 50 dot increments. While a tab at every 1/6th inch would seem adequate, you may find that positioning fields onto a preprinted form may require a finer resolution. When we define the output format, the appropriate printer code to position the field is buried under the first character of each field. For this purpose, we want absolute positioning on the X axis and, if you look at the codes beginning at #200 in the filePro print code numbering scheme, you will see there are enough codes to cover a 9+ inch wide line. As the positioning codes begin at 200, it is fairly easy to work out what code you need without consulting the table.

With the codes defined every 50 dots (which is 1/6 inch), when you need a code that will position your field 3.5 inches from the left margin, you will know that code 221 ($6 \times 3.5 + 200$) is the right one. We don't really need a code for a field that begins on the left margin, as the margin is automatically tabbed as it were, but you will find from time to time that you want to insert a couple of print codes before a field that is on the margin. With the positioning code, we can insert the spaces without changing the field's location on the printed page. That is really all that is required to line up the fields on the line. You may find that you want to change these values some to narrower tab stops, if necessary, to put the fields exactly where you want them on the line, but this is a place to start. If our print code table could be of unlimited length, then we could have a tab at every dot position!

Even though the fields are now lined up into columns, we've still got a real problem with right justified text and numeric fields. Using a fixed pitch font, filePro simply left pads the field with spaces and then treats it like any other field. For positive numbers, this would work with proportional fonts if the width of the space were the same as a numeral but, unhappily, they are different on any of the LaserJet fonts I have seen. There are two general approaches to get numeric fields aligned.

If there are no characters other than a space or a numeral before the alignment point in the field (that means no commas or leading minus sign), then there is a relatively easy way. This means 1000000, 100000-, 1.00, 1.00- are all OK, but 10,000,000, -1000000, -1.00 are not. Given that restriction is acceptable, then we can treat it as a left justified field. There is still a problem, in that the width of the space is different from the numeral, but HP has conveniently provided us with something called the Horizontal Motion Index or HMI.

The HMI is the mechanism within the printer that defines the pitch of a fixed pitch font. You could be printing a 12 pitch font and, by changing the HMI value, have the same font print at 8, 10, 18, or some other pitch. I don't know why you'd want to do this, but you could. Now back to our story. When you are using a proportional font and issue a new HMI value, only the width of the space is affected. So all we need to do is make the width of the space equal to that of the numeral and then our numeric fields will print just fine. The HMI command goes like this: `^&k###H` (where ### is the desired width of the space in 1/120ths of an inch and is valid to four decimal places). If we know the width of the numeral, some simple math will give us the correct HMI value (numeral width in dots * .4). Only the Times and Univers fonts were considered when the codes were created. It is quite likely that many of these codes will coincide with some of the additional LJ4 fonts.

The numeral widths come from a width table for each font and point size which we will get to shortly, but in the meantime, you can look at printer codes 184-199 in the table that make reference to the HMI and indicate the font(s) and point sizes where the numeral and space character widths will be equated.

Be warned that after you have issued an HMI command to the printer, it will revert back to its default setting when you subsequently change any font characteristics, such as bold on/off, for example. To use the HMI command in your output (and this assumes that you have the HMI values in your filePro printer code table), you would put the appropriate HMI printer code under the character just before the numeric field, as the first character in the field already has a cursor positioning code there.

If you must have commas (etc.) in your numeric field or you want to right justify a text field, then you will have to have filePro consult a character width table to establish the printing width of the field and then issue an appropriate cursor positioning command. All of this will have to be calculated in processing, as the data contained in the field will directly determine the positioning values needed for proper alignment. Essentially, we read the widths of all the characters in the field up to the alignment point, total them, subtract that amount from the dot position on the line where we want the fields to align, and then have the field print beginning at that point. For example, if you want to align some text fields on the right at dot position 2100, you would first measure the length of the field, assume real field #6, which works out to be 622 dots, let's say, then create a dummy variable that contains `^[*]p1478X{6`, and that dummy field would be the field on the output format. Note that the dummy variable contains both the positioning code as well as the field's contents. For numeric fields with decimal values, you would establish your alignment point at the decimal, rather than the end of the field. It isn't as complex as it probably reads, as you will see in examples presented later.

Character Width Tables

The width table is but a list of the character widths for each font and point size. For the HP scalable fonts, we only need to have one table for each of the font attributes, i.e., upright, bold, italic, and bold italic, etc. The character widths scale in direct proportion to the point size, so we can derive the width for any point size from a single table.

Hewlett-Packard was kind enough to send me the font metric files for the LJ3 internal scalable fonts. From that data, I have extracted into a filePro database the character widths of all characters for all of the symbol sets that the LJ3 supports. This is a lot of info, and the key file is on the order of 250K and is really quite easy to use in your programming. Similarly, the font metric files for the LJ4 were uploaded to the HP forum on Compuserve when the printer was announced, and those have been put into a filePro database as well. With a significant increase in the number of internal fonts and supported symbol sets, the key file for the LJ4 is just under 2 meg.

If you want this information, you'll have to get the "cookBook" disk for this issue, as there is simply too much there to print, and I think it unlikely that anyone would have the inclination to key it in anyway. More on the width tables and symbol sets in a bit.

I should probably interject here for those of you that are thinking about getting a laser printer, that Hewlett-Packard, aside from producing a fine printer, is really a good company to do business with. Their customer support is absolutely outstanding. If the person on the phone doesn't know the answer, they will find out and call you back. Their technical manuals are very well written, and it is clear that a great deal of care is taken in their preparation. I would personally need a very good reason to go to another brand. The LaserJet 4, with its 600 dpi and increased font selection, looks pretty attractive right now. I recently bought one to have at home.

For non scalable fonts, you will have to have a separate table for each of the font attributes and point sizes that you want to use. If you have some other fonts for which you don't have the character widths, send me some email, and I'll try to tell you how to measure them yourself, if I don't have the information somewhere.

filePro Output Formats

It is useful when inserting PCL codes into your output format, whether the code is integrated into one of your filePro variables or in a traditional filePro printer code, to consider what is actually sent to the printer. Take a "form" that is defined as 75 characters wide and 60 lines long.

In theory, if you didn't put anything on the form, filePro would direct the printer to print 60 lines of 75 spaces, yielding a blank page. In fact, when filePro detects that there is nothing else to print on the line, it immediately sends a linefeed.

Now, if you put a heading or other text on the form, that text simply replaces the spaces that would have been there otherwise. Similarly, if you put a field on the form, e.g., *1, the

spaces on the form are replaced by the contents of the field, beginning at the character position of the asterisk. It is important to understand that if you have a second field on the same line that is too close to the first field, i.e., you haven't allowed enough room for the first field, then the first field will get truncated and overwritten by the second field. Remember this. Its importance will become apparent later.

A filePro printer code, on the other hand, is inserted into the byte stream to the printer just before the character position it appears at on the line, in which case, we get our byte count effectively lengthened for that line, assuming, of course, that the printer code is non printing. A printer code that actually prints a character and is defined that way in printer maintenance will replace the "space" at the appropriate character position.

The point that I am making here is that any filePro output, be it a form or a report, is a long stream of bytes which arrives at the printer, and that by inserting various printer codes into this stream at the appropriate spot, we can get our output formatted as desired. Ultimately, it makes no difference if the print code got into the stream, as filePro code, as a variable, or directly on the output format itself. Things will be much tidier on your output format itself, however, if you can make use of a filePro printer code, but this isn't always possible.

An Example

Before getting into the actual implementation of these concepts in filePro, we will create a raw text file using "vi" or your favorite editor and print it out as a demonstration. Enter the text in Figure 1 into a new file. The grid showing the character numbers on the line shouldn't be entered, as it is only there for clarity.

The first section of the text prints data that is typical of (but abbreviated from) what many of us print in our day to day accounting. The very first line is the printer setup and selection of the courier 12 pitch font. The fields begin at the margin for the name, character position 31 for the invoice number, and at position 43 for the amount. Note that the amount field is left padded with spaces to fill a (8,2) filePro field. This prints as we would expect. Refer to Figure 2.

```

1...+...10...+...20...+...30...+...40...+...50...+...60...+...70..
^[\E^[\&16d3ef0P^[\&0p10v12b0s0b3T
Acme Consulting Inc.      12345      382.26
Condor Gold              12842      1684.91
Standard Building Supply 12965      32.07

1...+...10...+...20...+...30...+...40...+...50...+...60...+...70..
^[\&1p12v0s0b4101T
Acme Consulting Inc.      12345      382.26
Condor Gold              12842      1684.91
Standard Building Supply 12965      32.07

1...+...10...+...20...+...30...+...40...+...50...+...60...+...70..
Acme Consulting Inc. ^[\&p750X12345 ^[\&p1050X 382.26
Condor Gold ^[\&p750X12842 ^[\&p1050X 1684.91
Standard Building Supply ^[\&p750X12965 ^[\&p1050X 32.07

1...+...10...+...20...+...30...+...40...+...50...+...60...+...70..
^[\&k10.08
Acme Consulting Inc. ^[\&p750X12345 ^[\&p1050X 382.26
Condor Gold ^[\&p750X12842 ^[\&p1050X 1684.91
Standard Building Supply ^[\&p750X12965 ^[\&p1050X 32.07

1...+...10...+...20...+...30...+...40...+...50...+...60...+...70..
^[\&{#08
Acme Consulting Inc. ^[\&p750X12345 ^[\&k10.08^[\&p1050X 382.26^[\&{#08
Condor Gold ^[\&p750X12842 ^[\&k10.08^[\&p1050X 1684.91^[\&{#08
Standard Building Supply ^[\&k10.08^[\&p750X12965 ^[\&p1050X 32.07^[\&{#08
^[\E

```

Figure 1

Acme Consulting Inc.	12345	382.26
Condor Gold	12842	1684.91
Standard Building Supply	12965	32.07

Acme Consulting Inc.	12345	382.26
Condor Gold	12842	1684.91
Standard Building Supply	12965	32.07

Acme Consulting Inc.	12345	382.26
Condor Gold	12842	1684.91
Standard Building Supply	12965	32.07

Acme Consulting Inc.	12345	382.26
Condor Gold	12842	1684.91
Standard Building Supply	12965	32.07

Acme Consulting Inc.	12345	382.26
Condor Gold	12842	1684.91
Standard Building Supply	12965	32.07

Figure 2

Now, the second section begins with a font change to the Times 12 point proportional font. You can see that our output at this point is far from satisfactory. Let's get the fields in line first. The left margin is OK but the other two fields need some work.

The invoice number begins at character position 31, which at 12 pitch translates at 300dpi to dot position 750, so, if we insert the printer code `^[*p750X` immediately before the field value, those fields will be aligned. If you don't understand where the 750 came from, we'll work it through. We want to know at which dot position the 31st character begins. At 12 pitch each character is 25 dots wide (300/12). The first 30 characters then consumed 750 dots (30*25), but as the PCL coordinate system begins at position 0, our 750 dots are filling positions 0-749, conveniently starting char position 31 at dot 750. The same is true for the amount field at character 43, which translates to dot position 1050. The third section now has the fields aligned on the left, although the amount field doesn't look like it. The problem there is, of course, that the space and numeral characters have different widths.

By inserting the HMI printer code `^[&k10.0H` just before the beginning of the section, we have forced the width of the space to be the same 25 dot width as the numeral. The fourth section then looks like what we are after. It is purely coincidental that the Times 12 pt numeral happens to be the same 25 dots wide as a 12 pitch monospaced font character. There is one thing, though, which probably isn't apparent in this example, and that is that, when you have the space widened to the numeral width, it can appear visually too wide in normal text; and, as mentioned earlier, if you change a font attribute, the HMI value reverts back to its default value. So what we can do is, instead of issuing the HMI code right at the beginning of the text, we will insert it just before each numeric field's positioning code, and then just after each of the field's contents, issue an effectively null font attribute code. In this case, it was `^[s0B`, which has no effect, as we are already printing non bold. As we are already printing at 12 pt, the command `^[s12V`, which would select 12 pt, would have achieved the same thing, namely, to restore normal spacing without changing any other of the current printing characteristics.

The fifth section should then represent our original text, now aligned perfectly, just like the Courier 12 pitch, but now using the Times proportional font.

I strongly recommend that you enter the text into your own computer and get it to behave as shown. At that point, experiment a bit with changing the dot positioning values to move the fields around. Put some extra spaces just before the dot positioning command, and you will see that it has no effect on where the field prints, as we are giving the printer absolute positions on the line to place the text. Try reversing the 750 and 1050 values, and you will see that the fields print transposed from their locations in the text. I can't think of a reason that you might want to do this, but it's possible.

Actually, if you had two or three numeric fields on one line that were separated by some text fields, you could place them side by side on the form with a beginning HMI code and a trailing null font select code, and then, if your positioning commands were correct, it would all be located properly on your output, but would have only required one HMI command. It would certainly make the output format confusing to the uninitiated, but could make the format design a bit easier.

Enter some HMI values that are wrong to see what happens. If you fully understand this example using raw text and inserting the printer codes manually, as it were, you will find all the rest of it will come very easy.

Referring back to the previous example, let's put the detail lines into a filePro report. To simplify the discussion, we will ignore any header or subtotal sections and just look at the detail lines. We will define the fields as follows.

Field No. Description Length/Edit

- 1 Name (24,uplow)
- 2 Invoice No. (5,*)
- 3 Amount (8,.2)

Now refer to Figure 3 to see how this is laid out on the report format relating the filePro printer code numbers to the supplied table.

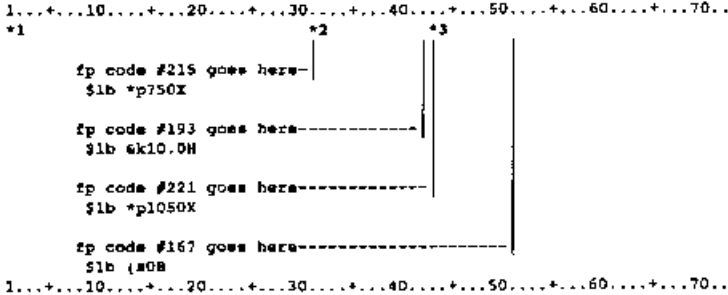


Figure 3

All that we have done in defining the report format is to insert the appropriate printer code from the table into the format at the right place, and that is really all there is to it. Presumably, the output from our report format will be essentially identical to what we previously did by hand. Note that the positioning codes are buried under the first character of their respective fields, and this is an absolute requirement. If there are any spaces or anything else between the printer code and the field's beginning, then our alignment goes right in the toilet. In the case of the HMI code, it is only required that it arrive at the printer sometime before the field is to be printed and sometime after a previous field that may be adversely affected by the code is encountered in the format.

Undoubtedly, we will never have enough room on the printer code table to meet every requirement, and further, some of the values required for the code may have to be established at runtime; so what we can do is to put the printer code into a variable and put the variable on the output format or, more than likely, integrate our filePro data into one variable that contains both the printer codes as well as the data. This is all done in processing.

There are two different approaches we can take in putting the printer codes into filePro variables. First, we can maintain our concept of fields on the report, and simply replace real fields on the format with dummy variables that contain both the printer codes and the data in the real fields; or we could have one long variable that contains everything we need to print on a single line. The former is probably less complex in processing, and the latter makes the output format definition an absolute breeze. The advantage, if you can call it that, of the printer codes being contained within a dummy variable is that you don't have to rely upon any predefined values from your printer code table. All things being equal, though, your life will be quite a bit simpler if you can make do with the printer codes off of the table.

Once we start embedding printer codes into variables, we totally lose any concept of the variable's length as it relates to the length of the field as it is actually printed on the output; and, as mentioned previously, filePro will truncate any field that is overwritten subsequently by a field to the right of it on the line. If you have done your duty and slogged through the original exercise, you will understand that the physical location of something on the output format only need have a general relationship as to where it will appear on the printed page, namely on the same line, and not even there necessarily. Because, at the point of designing our format, we don't really know how wide our lines may prove to be including both data and printer codes, define the format as being arbitrarily wide, maybe 200 char or to the maximum of 255 char. There isn't, at least from observation, any performance penalty to speak of from defining the format from being overly wide.

Let's go back to our original report with the three fields. This time we will keep the semblance of the fields on the output format. The first field needs no special treatment of any kind as it is on the margin and is a left justified field. We just print it as filePro delivers it. Field 2, however, needs to be positioned on the line at dot position 750. Rather than put field 2 on the format, in processing, we will create a variable that contains both the positioning code as well as the field data. A processing line similar to the following will do the deed.

```
aa(12,*)=chr("27"){ "*p750X" {2
```

Similarly, field 3, the amount, could be defined:

```
ab(29,*)=chr("27"){ "&k10.0H" {chr("27"){ "*p1050X" &3{chr("27"){ "s0B"
```

Note the use of the "&" join character in the variable "ab". Because the width of the space is being manipulated to achieve alignment, the use of a "{" join would squeeze out the spaces and destroy the alignment.

So then our output format could have:

*1 at position 1 on the format, *aa at position 27 on the format, and *ab at position 40 on the format. So, although the variable *ab eventually only prints 8 characters on the output, its length (including printer codes contained within) is 29, and if you wanted to print another field to the right of *ab, it had better be located 31 characters or more further to the right. The key thing to remember in this is that you have to leave enough room on your format for the entire variable, even though many of the bytes will be consumed by the printer and not appear on the page, and, of course, the format must be wide enough to accommodate the length of the final variable on the line.

Taking this a step further, we could define yet another variable:

```
ac(66,*)=1{aa{ab
```

In this case, the output format would only need to have *ac appear at position 1 of the line. This makes your output format rather tidy.

Finally, there is yet one other method that could be used to get our printer codes inserted into the output, and that is simply to put the code physically onto the format itself. We still need a printer code for the escape character (code 98), but as the balance of the HP codes are ASCII, we can put them directly on the format itself, leaving enough room for the fields, of course. See Figure 4.

Putting codes directly onto the format is quite often the most practical approach. First, it makes them immediately visible when reviewing a format on the screen, and it saves the need from defining obscure codes, particularly specific cursor positioning codes that in all likelihood would only be used on that particular format. Many times, you will find that you have to position the cursor at a specific dot on the line, perhaps to right justify some text, and then a "custom" printer code on the form is in order. You will see examples of this later.

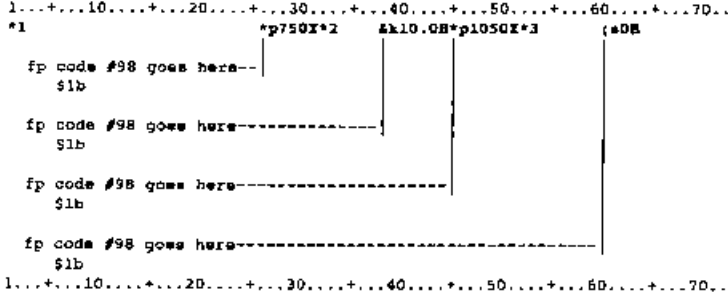


Figure 4

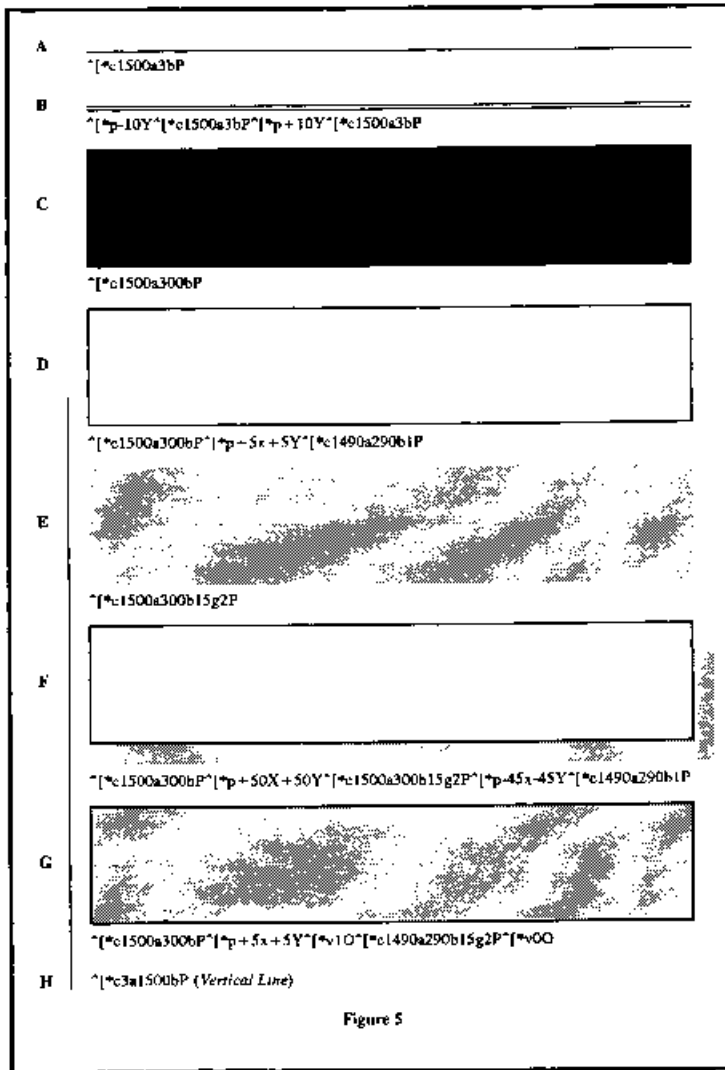
Really, it should be a matter of personal preference as to how you get the codes into the output, and I don't see any particular superiority to any method. The nature of the output may very well determine the appropriate approach.

Graphics

We will limit the definition of graphics in this discussion to drawing and filling boxes. While it sounds somewhat simplistic, much can be done that you would never have considered with filePro before. Refer to Figure 5 as you read through this part of the text.

Again, think of our printed page as being a matrix of dots or pixels, if you prefer, and we "turn on" the appropriate dots to print black and the collection of dots that are "turned on" represents our printed output. When we place a character on the format, the printer turns on the dots that will create the character at that location on the format.

Similarly, the boxdrawing printer codes turn on the appropriate dots to draw the box, as per the specifications in the code. We have codes to draw a solid black box, a solid white box, and boxes filled with various patterns. A good question immediately arises. Why would you want to draw a solid white box on an already white page? Well, we draw white boxes to "turn off" or reset dots that have previously turned on. You see, if we draw a solid black box and then decide that we would like the center of the box filled with a halftone pattern, let's say, sending the code to print the halftone pattern would turn on its dots; but because we are printing onto an area that already has all its dots turned on, there would be no effect. Printing a pattern only turns dots on and it doesn't reset any. So if you want a halftone or some other pattern, you would, in this case, have to reset all the dots in the particular area before you laid down the halftone pattern.



The statements in the last paragraph are not always true. There are a couple of options in using fill patterns. The default is that the pattern is applied to the destination transparently, which means that only the black dots in the pattern are turned on and the white dots in the pattern have no effect on the destination. A good analogy to this would be the use of one of the yellow felt pens often used to highlight text in a book. Although the yellow covers the printing, you can still see the text through it and read the text, but if you covered the same text with an opaque yellow oil paint, it would cover the printing so it was totally hidden. If we send the code `^[*v10` to the printer, and then our fill pattern, both the black dots and white dots are copied from the pattern to the destination, i.e., any existing black dots on the page that coincide with white dots in the pattern will be turned off, effectively overwriting anything that was "undemeath the pattern". Figure 5-G shows a box drawn this way. The situation will dictate which is the appropriate approach. It is probably wise to return the pattern transparency back to 0 when you are done (`^[*v00`). Box drawing works like this. The following code will create a solid black box that is 1500 dots wide and 300 dots high.

```
^[*c1500a300b0P
```

The value that precedes the "a" is the width, and the value preceding the "b" is the depth. The 0 preceding the "P" is calling for a black fill pattern, but note that the 0 need not be present in this case, as the printer will presume a black fill in its absence. The box is drawn down and to the right of the CAP, but the CAP is NOT changed by the box drawing.

In most cases, a solid black box isn't terribly useful, but a really skinny long black box is. We call it a line.

```
^[*c1500a3bP
```

or try

```
^[*c3a1500bP
```

You can see then that we can draw lines any width and any thickness in increments of 1/300 of an inch. In Figure 5-B, you can see that by using cursor positioning commands we can draw a "double" line. Varying line thickness and using a double rule here and there is ideal for separating sub total sections and the like in reports. A General Ledger is a natural for this kind of treatment.

Back to the original example. Let's extend the code some.

```
^[*c1500a300b0P^[*p+5x+5Y^[*c1490a290b1P
```

Now what we have done is drawn our original box, then moved the cursor in 5 dots on both axes and drawn a white box that is 10 dots smaller. The "1P" part of the code represents a white fill or reset mode. We are left with a ruled box 1500 x 300 dots with 5 dot thick lines. If you want a fatter rule, move in a little further and draw an appropriately smaller white box. There are other fill patterns besides black and white. "2P" represents a shaded fill and "3P" represents a cross-hatch fill. Because both of these fill patterns have their own variations, we end up with another parameter in the code, e.g., to specify the level of the halftone shading. I will only deal with halftone shading and leave it to the reader to deal with cross hatching. The game is the same, the numbers are different, and it is all covered quite well in the tech ref manual.

Although the boxdrawing codes do not change the CAP, in the last example, we actually moved the cursor to draw the white box, and the CAP would now be at the upper left hand corner of the white box. As we go further, I will represent the last example with an ellipsis, to keep from getting bogged down repeating the same code.

Continuing now,

```
...^[*c1490a290b15g2P
```

The last addition to our code fills our "white" box with a 15% shading pattern. You can see that we now have a "2P" fill pattern and have picked up a new section in the code "15g", which is the degree of shading. There are actually only eight levels of shading available, and they are defined by ranges in shading percentages, and whatever range your value falls into will determine the actual density of the shading you get. At "15g" here, we are right in the middle of the "11-20" percent range. So, if we had used "11g", "15g", or "20g", the result would have been the same. If you want a darker or lighter shading, change the value that precedes "g" in the code. Again, get the tech ref manual.

If you want to create a box that has a drop shadow, draw a second box that is shaded unruled, offset by 50 dots or so from the "real box". It's easy. See Figure 5-F.

Boxes don't have to start out as black and be worked over. If you want a shaded unruled box, the new part of our code will do that by itself, as you will see in a more real application later. None of this is restrictive at all. One thing: when you draw a box using the PCL codes, that box is being deposited on the page at an absolute position, specific dots are turned on, and none of your text is going to change its location. Boxes drawn by filePro in graphics mode, on the other hand, are merely text characters that, when aligned, will create a box, but die swiftly if you are printing proportional text or if you change pitch within the box.

If you find that you are getting into some minor difficulties using the filePro generated boxes, you may find a simple fix in putting a cursor positioning code under box end characters. If the box were defined at 12 pitch and the right hand edge of the box was at character position 50, just put ^[*p1225X under the right edge vertical box characters and they will always print at the correct spot on the line.

One other thing that should be brought up is a further definition of the CAP as it relates to boxes and text. The CAP, when printing text, is located at the bottom left of a capital "M". This may not be 100% accurate, but is close enough for our purposes. It is on the baseline, i.e., the line that represents the bottom of capital letters and most lower case letters. From any CAP, a letter that is subsequently printed will ascend above the CAP, and many characters with descenders (g, p, q, etc.) will go below, so to put text into a box, you will either have to make some effort in the positioning of the box or the text itself. Once you play with it a bit, you should be able to establish your own guidelines.

If you start mucking with the CAP to position a box vertically on the page, you want to at some point return the cursor to its original position, in the interest of getting the CAP back in sync with filePro's idea of where lines are positioned; e.g., if, for some reason, you have issued ^[*p-25Y and don't subsequently do a ^[*p+25Y, then filePro lines that follow are going to be (at 6 lpi), a half line off. It could be a big deal or maybe nothing, it depends.

In the course of drawing some graphics, if you have cause to reset some areas back to white, then the order in which things get executed becomes important. If you draw a ruled box like the one in our earlier example and want to put text inside it, you absolutely must draw the box BEFORE you print the text, otherwise the text will get erased during the box's creation. Just the other night, I must have spent 15 minutes trying to figure out what was wrong in a processing table because of missing bits on my output. The processing was fine, my data was being erased by the graphics.

One extension of the box filling business is to use the different fill and shade patterns to print text in something other than solid black. Look at Figure 6 and refer to the "Print Model" section of your tech manual. The codes that produced the graphics are underneath each of them. A pattern filled character is most effective at larger point sizes, as the pattern doesn't scale with the text, and some patterns read better against a black background. In Figure 6, the code at the top right positions the cursor and draws the black box, and is prepended to the codes that have the box. Note that in the examples neither the pattern transparency nor the selected pattern are returned to their default values after "filePro" is printed, and in your programming you should make sure this is done.

Figure 7 shows the effect of first printing black text to create a shadow and then moving the cursor some and overprinting with an opaque pattern. The creation of the shadow is a good demonstration of the push/pop printer codes. To do this, we issue the code to create the shadow pattern, if it isn't the current pattern, and, in this case, our shadow is black. But before "filePro" is printed, we save the address where the word starts with "^[&f0S". After "filePro" is printed, we restore the address with "^[&f1S", which puts the cursor right back to where it was before the "f" was printed. At that point, we issue a cursor move command, in this case, ^[*p-10x-10Y, select an opaque pattern and overprint the black with the pattern. There are all sorts of themes and variations, a few of which are shown here. Note that in the third one we only have shaded the word "file". Let your creative juices flow! Obviously, any of these code sequences could be assigned into a filePro variable and put anywhere on your output.

^(�S^[*p-15x-196Y^(*700a2506P^]&#f1S...

file

^[*c15G^]*v1o2TfilePro

filePro

...[*c15G^]*v1o2TfilePro

filePro

^[*c25G^]*v1o2TfilePro

filePro

...[*c25G^]*v1o2TfilePro

filePro

^[*c50G^]*v1o2TfilePro

filePro

...[*v1o1TfilePro

filePro

^[*c1G^]*v1o3TfilePro

filePro

...[*c1G^]*v1o3TfilePro

filePro

^[*c3G^]*v1o3TfilePro

filePro

...[*c3G^]*v1o3TfilePro

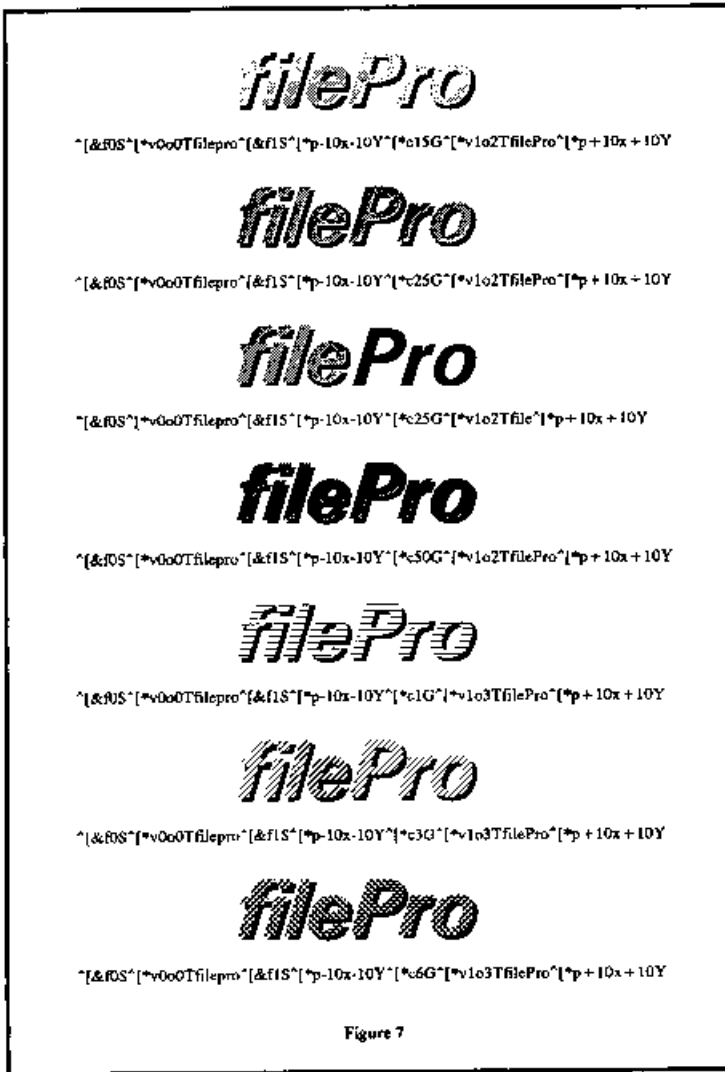
filePro

^[*c6G^]*v1o3TfilePro

filePro

...[*c6G^]*v1o3TfilePro

Figure 6



This type of thing tends to be somewhat of a gimmick and trite, but there are situations where it is quite appropriate. Don't use shadowed lettering because your printer can do it, use it because it adds something visually to your layout. You see it all the time on TV. When a new technology process evolves, every beer ad, car ad, and aspirin ad jumps on the bandwagon. The current one is the "morph".

I generally find it most efficient to develop a graphic sequence with a text editor, and once the codes are worked out, then worry about how to get it into the filePro application.

Practical Applications

I have created a filePro database called "ljet", which is a bare bones order entry file, purely for demonstration purposes. Within "ljet", there are a couple of output forms that implement many of the LaserJet features. You will see that the output formats are using "hp3" as the printer, and if you install your printer as "hp4", you will want to change this in the format "options".

First, is a form named "spine", and it could reside in any file, as the form supplies its own data. This is a small routine that prints a label for a binder which provides a clear sleeve down the length of its spine for titles. This is quite handy for labelling manuals for public domain software. You could modify this to make labels for your video tapes or signs for your store, etc. Get some Avery full page labels, stick one down on a piece of card, and you're done. We are going to draw some box shapes and center text supplied by the user at runtime in both normal and reverse type. As we are centering text that is supplied when the form is printed, the character width table supplies the text width information we need to center the text on the fly.

Have a look at both the processing table and output format for "spine." The processing is reasonably well commented as to what's going on. You will see that all of the work is completed in the processing table and the format has but four variables. There are no filePro print codes on the spine format. You could repeat any one of these labels by placing it on the form more than once. You do have to pay attention, though, that you don't put them so close together that they overlap.

The size of the labels reproduced here was construed to fit on the "cookBook" page and is really a bit small for a regular binder, so there is a second format on the disk called "spine1" that creates labels of a more appropriate length for use in a binder.

It is pretty straightforward. The first 9 lines establish our working variables and get the character widths into an array, and lines 10 through 16 create the label, which is moved into the variable "nl" for printing. Lines 17-24 print a similar label, but in reverse type. The remaining two labels are the same as the first two, except they are a bit deeper. Note at line 5, there are variables created for setting normal and reverse (white on black) text. Make sure when you go to white on black mode that you return to "normal" printing when finished or you may find yourself in possession of some blank sheets later on. In the processing here, the variable "bt" is tacked on the end of the print string in Line 24.

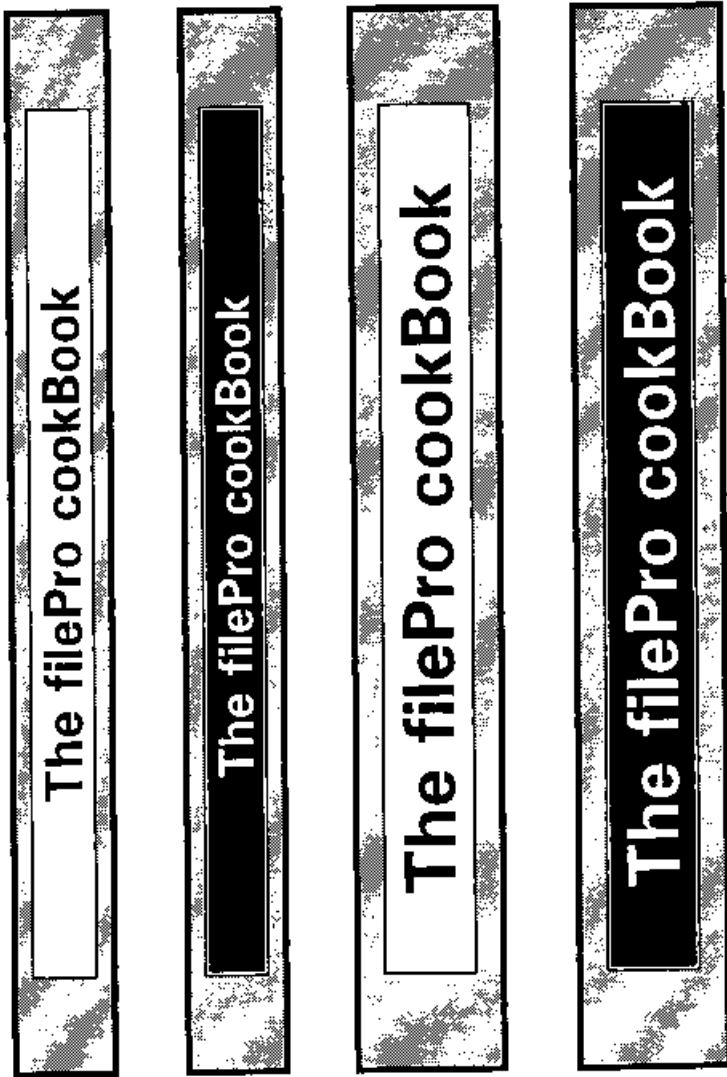


Figure 8 (ljet/out.spine)

Calling the Width Table

The only thing that really needs to be brought up here is the use of the character width table. If you have the disk from this issue of the "cookBook", you will find a filePro file called "widths". Each record contains 265 fields, the first 255 being 1 point character widths in ASCII order. To use the table, you must first "lookup" the correct record for the font and symbol set in use and move the lookup record into an array. (See lines 6-7.) Fields 256-259 also contain 1 point font information in the same format as fields 1-255, so define your array with 259 elements. Field 256 has the default space width, 257 the recommended line spacing in dots, 258 the Caps Height, and 259 the "lowercase" height. Look at the "widths" map for a description of the balance of the fields.

IndexA is built on field 261, which may require a bit of explanation. The index field is 8 char wide with the format, "ty ty wt st sy sy sy", where "ty" specifies the typeface, "wt" the stroke weight, "st" the style, and "sy" the symbol set. See the accompanying table, which gives mnemonic definitions for the typeface, stroke weight, and style. The final four characters are simply the call letters of the symbol set you are using. Some symbol sets only have a two character call sequence, in which case your search key would only be six characters in length, while others have three, and at least one on the LJ4 is four characters. In line 2, our search key (sk) is defined as "unbu10U". This lookup will find the width table for "Univers bold upright typeface PC-8 symbols".

Width Table Mnemonics

Typeface Mnemonic Typeface Mnemonic

Albertus al Omega om
 Antique Olive ao Symbol sy
 Arial ar Times ti
 Clarendon cl Times New Roman tn
 Coronet co Univers un
 Garamond ga Wingdings wi
 Marigold ma

Stroke Weight Mnemonic Style Mnemonic

Medium m Upright u
Bold b Italic i
Extra Bold x Condensed c
Condensed Italic k

Look at line 9 in the processing table. You will notice that our input string is assigned to the variable "l"(ell) which was never declared. If you squeeze out the spaces from both ends of an undeclared variable, filePro's length for that variable won't include any trailing or beginning spaces, and that is quite handy if we want to center text.

Centering Text Horizontally

We center text as we would normally using a fixed pitch font, i.e., to divide the whitespace evenly on either side of the text, but in this case we are working in dots, so the process is quite precise. As mentioned before, when a box drawing operation is completed, the cursor remains at the upper left hand corner of the box.

The text length is measured in the subroutine "getwid" starting at line 41 in the processing table. We simply step through each character in the string using a "mid" statement and get the ASCII value of each character, which is stored in variable "cn". As our width array is in ASCII order, the width for that particular character is the "cn"th element in the array. Well, not quite.

Remember, the widths in the array are for a 1 point font with 5 decimal places, and we need to multiply the newly found width by the current point size to get the character width in dots. The character width is an integer value. Our actual character width is stored in the variable "cw", which is declared as an integer, and filePro handles the rounding perfectly. FYI, the printer rounds up on a fractional part of a dot .5 or greater, otherwise the fraction is ignored.

If you look at line 45, you will see that the widths for each character are accumulated into the variable "aw". There is a test at line 40 to see if we have reached the end of the text and, if that is the case, we drop down to line 48 to work out the necessary cursor move to center the text. Note at line 44, there is a test for chr("0"). The printer makes no cursor move upon receipt of chr("0") but, as we don't have an element wid("0") in the array, we must insure that we don't go looking for one. I don't know how a chr("0") would get into filePro output, but ...

Speaking of illegal characters, should you try to print a character not supported by the symbol set in use, i.e., the symbol set has no character for the particular ASCII value, the printer will print a space. The width table will get the width right but, of course, you won't get what you wanted.

In line 48, we subtract our text width from our box width and divide the result by 2 " $xa=(bw-aw)/2$ ", thus "xa" now contains the distance we need to move the cursor to center the text along the X axis. Although this move will center us left and right, the cursor is still sitting at the top of the box vertically.

Centering Text Vertically

The vertical centering of the text is a bit trickier in a narrow box such as this, not so much in the implementation, but more in deciding what to center on! Character heights vary, e.g., Aag, so you have to decide whether you want to center on the Caps height, lowercase height, or Caps height plus descenders, lowercase plus descenders, etc., etc. In practice, you really only have to decide between Caps and lowercase, with lowercase being the most common choice. It's purely a visual thing.

The more white space you have to play with, the less of an issue it becomes. This time, we will center on the Caps height. In the width table, field 258 contains the 1 pt Cap height and field 259 contains the 1 pt "x" height (lowercase). If you multiply these fields by the current point size, you will get the uppercase or lowercase character height in dots. Line 41 on the table puts the Caps height into the variable "ch". To get our vertical cursor move, we first have to get our text into the box. Presently, the bottom of our text is at the very top of the box. If we move the text down by the amount of the Caps height, all the whitespace would be below the text. To center it vertically, then, we want to move down further half the amount of the whitespace, which, by formula, would be $(bh-ch)/2$, and, at line 41, you see $ya=ch+((bh-ch)/2)$. "ya" contains the cursor move we need to center the text vertically.

Finally, in line 48, the variable "h" contains the complete printer code to center our title in the box, both horizontally and vertically. The variable "h" is integrated into our print string immediately before the text and we are done.

Backing up just a bit, I would like to demonstrate how you would use the width table to align columns on a decimal point. Assume that the appropriate width table is already in the array "wid", the point size is in variable "ps", and that we would like to align real field 11 at dot 1819 on a report.

```
Then: pt="" { 11 { ""; ap="1819"; gosub getwid
...
...
getwid
Then: cn=""; aw="0"; cw="0"; pn="0"
loop
Then: pn=pn + "1"

if: mid(pt,pn,"1")="."
Then: goto dowid

if: mid(pt,pn,"1")=chr("0")
Then: goto loop

Then: cn=asc(mid(pt,pn,"1")); cw=wid(cn)*ps; aw=aw+cw

Then: goto loop
dowid
Then: pp=ap-aw; cp=chr("27"){ "*p" { pp{"X" { 11 { ""; return
```



```

5
6 *hi
7 *hj
8
9
10 Invoice *p1373XInvoice No.: *5 &l8D
11
12 *p1540XDate: *@dt
13
14 Sold To: &l6D *p1270XCustomer P. O.: *6
15
16 *1
17 *2
18 *3
19 *4 Attn: Accounts Payable
20
21
22 *bx
23 *p351x-34YQty*p916XDescription*p1610XUnit Price*p1887XExt. Price*p+34Y
24 *11 *21 *31 *41
25 *ia
26 *12 *22 *32 *42
27
28 *13 *23 *33 *43
29 *ia
30 *14 *24 *34 *44
31
32 *15 *25 *35 *45
33 *ia
34 *16 *26 *36 *46
35
36 *17 *27 *37 *47
37 *ia
38 *18 *28 *38 *48
39
40 *19 *29 *39 *49
41 *ia
42 *20 *30 *40 *50
43
44 *bg
45 *p1383x-50YSub Total *p1823X*51
46 *ib
47 Terms: Net 30 Days *p1381XSales Tax *52
48
49 *p1302XInvoice Total *p1823X*53
50 *ib
51
52 *p+50Y
53
54
55
56
57
58
59
60
.....E.N.D...O.F...F.O.R.M.....

```

Figure 10

Line: Pos-Code, Pos-Code, ...

- 1) 1-100, 2-101, 3-116, 4-156
- 4) 1-206
- 6) 1-147, 2-171
- 10) 1-154, 2-173, 3-206, 10-134, 11-151, 43-98, 63-99, 70-98
- 12) 43-98, 50-100, 56-99
- 14) 1-100, 2-206, 11-98, 43-98, 66-99

- 16) 1-208
- 17) 1-208
- 18) 1-208
- 19) 1-208, 15-216
- 20) 1-136, 2-146, 3-171
- 22) 1-206
- 23) 1-98, 11-100, 14-98, 31-98, 48-98, 65-98, 71-151,
72-99, 73-195
- 24) 1-207, 6-167, 7-210, 58-195, 59-232, 68-237
- 26) 1-207, 6-167, 7-210, 58-195, 59-232, 68-237
- 28) 1-207, 6-167, 7-210, 58-195, 59-232, 68-237
- 30) 1-207, 6-167, 7-210, 58-195, 59-232, 68-237
- 32) 1-207, 6-167, 7-210, 58-195, 59-232, 68-237
- 34) 1-207, 6-167, 7-210, 58-195, 59-232, 68-237
- 36) 1-207, 6-167, 7-210, 58-195, 59-232, 68-237
- 38) 1-207, 6-167, 7-210, 58-195, 59-232, 68-237
- 40) 1-207, 6-167, 7-210, 58-195, 59-232, 68-237
- 42) 1-207, 6-167, 7-210, 58-195, 59-232, 68-237
- 45) 1-167, 4-98, 15-100, 25-99, 58-195, 61-98
- 47) 1-167, 2-173, 3-100, 4-206, 10-171, 12-99, 33-98,
40-100, 50-99, 67-195, 68-237
- 49) 1-167, 4-98, 11-100, 25-99, 58-195, 61-98
- 52) 1-98

Figure 11

The next thing of interest on the format begins at line 10, where you find "Invoice No.:", and similar entries on lines 12 and 14. These three lines have text that we want to align vertically on the right. Similarly, down at lines 45, 47, and 49, we want "Sub Total", "Sales Tax", and "Invoice Total" to align on the right. Our friend the width table comes to the rescue. Back to line 10. I decided that "Invoice No.:", "Date:", and "Customer P.O.:" should be aligned at dot 1675 on the line. So, to achieve this, we have to establish the exact width of the text in dots, and position the cursor at "1675-textwidth" before we print the text. The grunt is really to get the text width and the rest is easy. In the input processing for the "widths" filePro file, there is an "@keyw" routine that will prompt you for the "fontspec", the point size, and the text you want measured. Once you enter the text, you are given its width in dots. The processing does check for a valid point size, i. e., a positive number in .25 increments. The left-arrow key is used to "back up" to the previous input prompt, or to eventually exit the routine. For convenience, the code is also duplicated in the "ljet" input processing table.

In the case of the address/telno line in the heading, I had decided on the text and measured for an 8 pt width. I knew I wanted to fill 1569 dots, and at 8pt, I was a 150 dots or so short. At 9 pt, the resulting width was within a dot or two. Bingo!! If, for some reason, 8 pt were required, then the space between the address and phone number section of the line would have to be widened or the text lengthened a tiny bit. Remember, the fonts are capable of being scaled in .25 point increments, and that can be useful if you have to shoehorn some text exactly into a given area. There is ALWAYS a way to cheat.

"Invoice No:" etc. are printed in Univers 13pt bold and work out to be 302 dots, 135 dots, and 405 dots respectively. Then, by subtracting those numbers from 1675, we know where to position the cursor to get each of the "fields" to line up at dot 1675. You can do the math to reconcile the codes on the format. The "Sub Total" group are 242, 244, and 323 dots respectively and are planned to align at dot 1625. You could calculate all of this in processing at runtime, but, as these are static things on the form, it is best to do the math only once, beforehand.

Also, at the end of line 10, there is a printer code that changes the line spacing to 8 lpi. This is just to close up the next few "double spaced" lines. I went to double spaced 8 lpi as the 13 pt text is too tall for single spaced 6 lpi. I bring this up to point out that the LaserJet does NOT count lines when deciding it's time for a new page. When you do the page setup so it works out to 60 lines at 6 lpi, the printer establishes the page boundaries in dots, and a subsequent change in the lpi does not change these boundaries. Sixty lines at 6 lpi translates to 3000 dots, and the printer will only do a page eject when a linefeed or half linefeed moves the cursor past dot 2999 (our 3000 dots are numbered 0-2999). At line 14, there is another visible code that changes back to 6 lpi. Through the printing of the 4 lines at 8 lpi, we have lost 50 dots in the vertical cursor advance, which is made up at line 52 in the format.

The variable "bx" at line 22 on the format draws the ruled box that contains the invoice line items and titles. This is built on the processing table in lines 12-17. Then, at line 23 on the

format, we put the line item titles into the box. You will see, to get the titles to appear in the center of the boxes vertically, the cursor had to be backed up 34 dots vertically, and the 34 dots were put back after they were printed. All of these titles were measured using the width table and then centered in their respective boxes. They did center perfectly the very first time. Not a tribute to my expertise (ha!), but a demonstration of how a little bit of planning can save you a lot of time.

Next, take a look at the variable "ia", defined at line 18 on the processing table. It is simply a 15% shaded box that is 2 lines high at 6 lpi. As we need it to print above its physical location on the format, there is a cursor positioning command included for that purpose. Also, note the use of the ^&f0S ^&f1S cursor push/pop commands to get the cursor back in sync with filePro's lines. We could have used the push/pop mechanism with the line item titles previously, rather than using a final ^[*p+34Y sequence. Further, the shading is printed transparently, so as not to erase the vertical lines in the box.

All of the numeric fields in the line item and total section of the invoice are aligned by cursor positioning and setting the HMI to 27 dots, which makes the space the same width as the Times 13 point numeral. Finally, on the form we have "bg" that draws the box for the total part of the invoice, and "ib", that provides the shading.

This invoice is probably as complex a format as you'll ever get into. Reports, by their nature, have far less designing required, as most of the page is repetitive line items.

Look at the form initialization codes for these forms. Both "spine" forms use code 57 which selects a landscape format at 6 lpi, while the invoice uses code 55, which is for a portrait orientation at 6 lpi. On the invoice, all of the font selections are done on the form itself, and none in initialization.

An Inexact Science

When using proportional fonts in a text field such as the "product description" field on this invoice, you can't really predict how much physical space may be required on the printed page to accommodate the field's contents, even though the maximum number of characters is known. It is almost certain on the invoice that we could enter a text string into the field that would print wider than the space allocated on the form.

If you considered each character position on average to be the same as a numeral, you'd probably never get into problems, but that may be too generous. A bit of experimentation should yield some decent guidelines. Numeric field widths can be calculated quite accurately.

Fudging the Margin in Reports

As mentioned earlier, the printer only does a page eject when a linefeed or half linefeed put the cursor beyond the bottom margin. We can still print past the margin if the cursor arrives there via a positioning command.

It is a rather trivial task on a report to put the page number at a fixed position at the bottom of the page. Essentially, we create a variable that will be placed in the heading area of the report format, which contains "push cursor { goto address { page number { pop cursor".

Say we have a report format and the page is defined with a 1/2 inch top margin and 57 lines of text. At 6 lpi, this would leave a bottom margin of 1 inch. Now we introduce a new cursor positioning command, whose unit of measure is lines or rows as opposed to dots, ^&a####R, where #### is the row number down from the top margin. As the row numbering scheme begins at 0, the 57 lines in our format are labelled 0-56, and we want to print the page number down in the margin at line 58, which allows for a blank line.

In processing we can do something like this:

```
Then: ec(1,*)=chr("27"); sc(5,*)=ec{"&f0S"; rc(5,*)=ec{"&f1S"
```

```
Then: pn(6,.0)=@pn + "1";
```

```
Then: pp(35,*)=sc{ec{"*p1125X"{ec{"&a58R"{ "Page"<pn{rc
```

```
Then: end
```

The variable "pp" is placed anywhere on the format heading, but it will be printed below the bottom margin. For some reason, if you pick up "@pn" in processing and put it in a variable on the heading, the number is off by "1", thus we set pn=@pn + "1".

Positioning the cursor vertically by "lines" rather than "dots" is preferable here, because then you don't have to calculate the exact dot position where line "58" is located. When the cursor is on the first line of a report or form, the CAP is NOT at dot position "0" vertically, but rather on the baseline of the first line, which is defined as being down from the very top 75% of the value of the current line spacing. So, the exact position of the CAP is dependent on the line spacing. Positioning by row number precludes us from having to translate the row number into dots.

This is particularly handy if you are using preprinted forms. Keep in mind still, the printer isn't counting lines. When given a row positioning command, it calculates a dot position, based on the current lpi setting, or, if you prefer, the "Vertical Motion Index" - VMI.

This technique could be used to put any kind of information at a fixed position at the bottom of the page. Page totals, maybe? If it is not possible to put the variable containing the data in the heading and it has to go onto one of the data lines, then you will have to make some creative use of "@lc", etc., to load the variable with the data and positioning values at the appropriate time. Play with this a bit, it shouldn't be difficult.

Print Code Table Review

Now, let's take a quick tour through the print code table. As stated before, the first 54 codes are functionally unchanged from the original hplaser.prt file supplied with filePro. A few of these codes have been duplicated further down the table in the interest of getting all symbol set definitions, for example, for both primary and secondary fonts, together.

Code 55-75 are intended to be the area where any user initialization codes are to be inserted. Supplied, there are four, 55-58, that do basic page formatting, while 73-75 set the number of copies. If you have a duplex printer (IIID, IIIsi), you will want to set up codes here for simplex/duplex operation and to select the bin the paper is to be drawn from. I know there are a few. There isn't much space available for additional codes, so use it wisely.

You might want to make up a couple of codes that select fonts for both the primary and secondary if you are using the same combination repeatedly. Hopefully, you can build any new code you need by nesting existing codes from the table, e.g., to create a code that made the primary font Courier 12 pitch and the secondary Times 10 pt, the new code would be defined as "%89 %101". This new code might include the symbol set, as well. Any further mention of codes that involve font attributes implies a provision for codes for both the primary and secondary fonts.

There is, though, a nasty bug in filePro that causes an error if you try to nest a print code with a value higher than 127, which precludes half the table from being nested. The problems don't appear when you define the code, but rudely show up when you try to use the code. Small Computer knows of this, and I have been told that it "will be fixed." Soon, I hope.

Continuing down the table now, 76-87 are reserved for symbol sets. Four of each are already defined but can be changed if you wish. Codes 88-97 select the fixed pitch fonts. Code 98 is simply the escape character, and you will use this one quite often if you are building your own codes directly on the output format. 99 and 100 activate the primary and secondary fonts, respectively. These are mutually exclusive, i.e., if you call code 100 for the secondary font, the primary font is inactive, and it won't subsequently be active until you issue code 99.

Selection of proportional fonts is in codes 101-128. For the LaserJet III, we only have two typefaces, so there is a bit of free space here, but for the LaserJet 4, it is pretty much full. Any of these codes select the 10 pt medium upright version of the typeface. As these fonts are scalable, we have to make the point size selection with another code (129-158). Originally, I tried to put in codes that gave you the same typeface in a variety of point sizes, but I gave up on that idea as it became apparent that there wouldn't be enough room on the table if you added a few fonts. The LJ4 adds more than a few. As our point size codes (129-158) are > 127, we can't use them nested, which is a real drag.

Similar to the point size codes, 159-166 select the pitch for fixed pitch fonts. These aren't really needed for the LJ3 internal fonts, as we have all of the possibilities covered in font selection. On the other hand, the IIIsi and the LJ4 both have scalable fixed pitch fonts, and this is the area in the table to predefine some.

This brings us to bold type, etc., at 167-175. When we use a line printer, we only think of turning bold "on or off". While this may be true with a line printer, laser fonts can have weights other than bold or medium, so we aren't dealing with an on/off situation. The codes are here for medium, semi bold, bold, and extra bold. There are others, but as we are out of room on the table, I didn't include them. These four variants will cover all the fonts on the LJ4.

The codes for font style at 176-183 show that there is more to the typeface style than italic or upright. Like the typeface weights, we have four different styles defined and these aren't all of the possibilities. They are upright, italic, condensed, and condensed italic. The LJ3 only has upright and italic fonts, while both the IIIsi and LJ4 have condensed and condensed italic fonts.

Codes 184-199 are a collection of HMI codes that relate to different numeral widths for the Times and Univers fonts at various point sizes.

Finally, the balance of the table is filled with our horizontal positioning codes. I wish we had room for more of them, but there you have it.

Omissions

One area that I have totally ignored is the "Printer Job Language" (PJM), which is used to select PCL or Postscript with the LJ4 or LJ3si printers with the Postscript SIMM installed. I don't have Postscript on my LJ4 so, consequently, my PJM Reference Manual remains untouched. If this is of concern to you, you probably want to put PJM commands in both the printer initialization and termination commands to specify PCL for use with the print code table.

There are still many, many PCL codes that are not on this table. In most cases, these require runtime values, e.g., drawing a box or other similar things that in all likelihood will be defined in processing. On Unix systems, if your version of filePro does not give you the ability to map LF to CR-LF, then you will want to put the code "\$1b &k3G" into your printer initialization in code # 3. If you can see something vital that should be included, let me know.

LaserJet III

Although the printer code table printed here is complete for the LJ4, the tables for both the LJ3 and LJ4 are on the cookBook disk. If you just can't wait to get the disk, make the following modifications to the hp4.prt table.

The only mandatory changes are in codes 88, 89, 93, and 94 where you want to change "4099" to "3". Those of you who have the IIIsi should NOT make this change, as your Courier font is scalable.

The following codes are not needed: 80, 81, 86, 87, 90, 91, 95, 96, 103-111, 117-125. Leave those codes blank on your table.

LaserJet 4

From the standpoint of the print code table, the only thing with the LJ4 that should be brought up is font selection. First, the Courier and Letter Gothic fonts are scalable. The Courier typeface on the LJ3 has a typeface number "3", while Courier on the LJ4 is "4099". The Clarendon Condensed font, codes 105 and 119, is only available in condensed bold. If you have it selected and then issue any other stroke weight or style change, you will almost certainly get another typeface.

Be sure to set the printer to 600 dpi resolution at the control panel. It appears that some characters at certain point sizes scale to different widths at 300 dpi vs. 600 dpi, and the width table expects 600 dpi.

There are no font select codes for the Wingdings and Symbol fonts on the LJ4 table. As each of these fonts has a unique symbol set, simply changing to the appropriate symbol set (codes 80-81, 86-87) will get the font into action.

Width Table Bloat

While I thought it was prudent to generate the character width tables for all fonts and symbol sets, I think it is very unlikely that any one of us individually needs them all. You could reclaim a substantial amount of disk space by deleting the tables for symbol sets that you will "never" use and then restructuring the file. Study the symbol sets carefully, and then perform surgery at your own risk.

What Next

We really need a bigger print code table, one that supports maybe 1000 codes. Even doubling the number of codes to 512 would help a lot. Right now, a LaserJet 4 with an added font cartridge is already out of space on the table. I asked Small Computer if it was a possibility and was told that there would be a couple of technical issues. Primarily, an extended table would not be backward compatible with existing tables and formats, and further, memory limitations could be a concern on some systems. We can always hope.

If we are able to get a larger print code table that supports 500-1000 codes, at that time the table presented here would want to be rewritten to open up space within the table to maintain its structure and grouping of codes.

For those of you who are still using filePro version 3.0 or earlier and want to play with this, the best advice that I could give is to upgrade to 4.1., otherwise, you have a very tedious task in front of you to enter the hex codes into your print code table. You can make it work, but you may find yourself putting more codes into processing than you would with 4.1, as the 3.0 and earlier versions have fewer codes and severely restrict their length, as well.

In conclusion, I would like to point out that there is nothing definitive in the way that I have approached utilizing the PCL codes in filePro. Given the flexibility of both filePro and the HP printers, there may very well be better ways to get the job done. If you have any problems in making it work, send email or call me at work during business hours (9-5 Pacific time).

Introduction

When installing filePro Plus on a multi-user *NIX systems such as AIX, LINUX and UNIX, it creates a termcap (terminal capabilities) file in the /appl/ftp directory. This file includes "tables" of keys available on the terminals you could be using. The filePro programs look for the applicable keys in the tables and uses them for screen prompts and to determine what sequences should be sent for the keys that are pressed.

For example, since NCR 7901 terminal's cancel key is [Rub Out], the message at the bottom of the screen in filePro update mode appears as "Press Rub Out to Cancel".

However, since we cannot do the same in the manual, we use IBMs name for the keys such as **[Ctrl][Break]** instead of **[Del]**. To translate, see the key tables in the next topics of this reference. To display the keys used for your terminal press **[Esc][?]**. Use **[Alt][F10]** to display the "Key Table" on Windows systems.

Note: When pressing **[ESC][?]** while in Inquire/Update/Add (clerk), it will display your licensed User Count.

If your terminal isn't listed in the following topics, it may have been added afterwards and may still be listed in the /appl/ftp/termcap file. If not in the termcap file, you must change /appl/ftp/termcap to include your termcap. See the instructions in "Adding New Terminal Types".

Terminal Features

Some terminals have more features than others. Refer to the following if you encounter problems when using your terminal.

Reverse Characters and Scrolling

On some terminals, pressing the reverse-mode key in "Define Screens" at position 1,1 causes the screen to scroll up by a few lines making it impossible to see the cursor and any reverse characters you may have typed. To correct the problem, re-display the screen by pressing the [redraw screen] key for your terminal.

Graphics

Some terminals do not have complete graphics capabilities. Although the graphics key turns on and off such terminals in "Define Screens", the characters accessed are plus signs (1-9), a dash(0) and a vertical line (the period) instead of the graphic bars and corners.

End Markers

End Markers on IBM terminals are small triangles. On other systems and terminals, the End Markers may be small boxes, periods or even question marks.

Lost Cursors

Some terminals don't have separate on and off sequences but use a toggle instead. On these terminals, it is possible for the cursor to disappear or get "lost". When this occurs, the [Restore Cursor] key turns the cursor back on.

Adding New Terminal Types

Before adding a new terminal type, check the /usr/share/terminfo file to find a similar terminal type or one that is reasonably close to your terminal type. You can then copy the similar terminal type to a new name in the terminfo file so that you don't have to start from scratch when adding a new terminal type. Check your terminal's manual to find the proper sequences for each code.

Note: Case is significant when writing or editing terminfo files.

Refer to the " **Terminfo Function Codes** " and " **Terminfo Graphic Codes** " tables for codes to be set equal to the strings of keys that filePro sends to the terminal for each function. These first sets of codes are standard entries and listed for your convenience.

The " **filePro Plus Codes** " and " **Key Label Codes** " tables list special codes related to filePro. The " **Key Label Codes** " table, starting with " **LO** ", directly correspond to the " **P** " series codes. These codes tell filePro what to display in the screen prompts for each of the corresponding " **P** " series codes.

A typical terminfo definition starts with comment lines and a title. Then the codes are added with the applicable sequences. The following is an example for a LINUX terminfo definition.

```
# Terminfo for LINUX created on 02/05/2000
#
linux|clinux|Linux Console:\
:al=EL:ambbs:cd=ELJ:ce=EK:c=EL2JELHcm=EL%/%d;%dHco#80:\
:dc=ELPd=ELMdn=ELB:ei=ho=ELHtic=EL@im=:li#25:\
:nd=ELCms:pt:so=EL7mse=ELmus=EL4mue=ELmup=ELA:\
:kh=ELHkb=^h:ku=ELA:kd=ELB:k=ELD:kr=ELC:eo:\
:GS=EL12m:GE=EL10m\
:GV=263:GH=304:\
:G1=277:G2=332:G3=300:G4=331:GZ=376:\
:GU=301:GD=302:GC=305:GR=303:GL=264:RT=AJ:\
:L0=F1:L1=F2:L2=F3:L3=F4:L4=ESC ESC:L5=F5:\
:L6=Pg Up:L7=Pg Dn:L8=ESC TAB:L9=TAB:LA=Ctrl-O:\
:LB=F6:LC=Ctrl-L:LD=F7:\
:LE=F8:LG=Ctrl-Z:LH=F9:LY=Ctrl-C:LZ=Enter:\
:PO=EL[A:P1=EL[B:P2=EL[C:P3=EL[D]\
:P4=BE:P5=EL[E:P6=EL5~:P7=EL6~:P8=EL^:P9=EL^\
:PA=^O:PB=EL17~:PC=^L:PD=EL18~:PE=EL19~:PG=^Z:PH=EL20~:\
:PJ=EL21~:LJ=F10:PV:
```

For more information, see the section in your operating system manual or a WebSite that describes preparing or revising terminfo entries.

Termcap Function Codes

The following table reflects some common termcap function codes. This is by no means a complete list but includes many of the codes related to filePro operation. Refer to your terminal manual for a complete list of codes.

Entry	Function
ti	initialize terminal for full-screen mode
tc	entry of similar terminal (not
te	restore terminal to line oriented mode
cd	clear to end of display
al	insert a blank line
dl	delete a blank line
Ce	clear to end of current line
Cl	clear the entire display and home cursor
CF	cursor off
CN	cursor on
Cm	cursor motion
ho	home cursor (move to
up	move cursor up one line
do	move cursor down one line
bc	move cursor left one column
nd	move cursor right one column (non-destructive space)
co	number of columns in a line (numeric; must be at least 80)
li	number of lines on screen
so	Begin reverse ("stand-out") mode
se	end reverse ("stand-out") mode
sq	number of extra characters needed by "so" or "se"; (numeric must be zero)

Termcap Graphic Codes

The following is a list of common graphic codes.

Example: The following graphics codes would be entered for a LINUX termcap definition.

```
:GS=\E[12m:GE=\E[10m\
```

```
:GV=\263:GH=\304:\
```

```
:G1=\277:G2=\332:G3=\300:G4=\331:GZ=\376:\
```

```
:GU=\301:GD=\302:GC=\305:GR=\303:GL=\264:\
```

Graphic Code	Function
GS	start graphics mode
GE	end graphics mode
GH	horizontal line
GV	vertical line
G1	upper right corner
G2	upper left corner
G3	lower left corner
G4	lower right corner
GU	up-facing T
GD	down-facing T
GL	left-facing T
GR	right-facing T
GC	cross
GY	F6 arrow
GZ	end-of-field marker

filePro Plus Codes

This table identifies the codes and filePro function. Apply the key sequence to associate each of the filePro functions for your terminal in the termcap file.
Example: The following lines would be entered for a LINUX termcap definition.

```
:F0=EQ[A:P1=EQ[B:P2=EQ[C:P3=EQ[D\  
:P4=BE:P5=EQ[E:P6=EQ5~:P7=EQ6~:P8=EQ7~:P9=EQ8~\  
:PA=^O:PB=EQ17~:PC=^L:PD=EQ18~:PE=EQ19~:PG=^Z\  
:PH=EQ20~:PJ=EQ21~:PV:
```

Code	Function
F0	insert character label
P1	delete character label
P2	insert line label
P3	delete line
P4	save information
P5	duplicate information
P6	up-tab label
P7	down-tab label
P8	left-tab label
P9	right-tab label
PA	clear-to-end-of-field
PB	display fields
PC	redraw screen
PD	enter print code
PE	display print codes
PF	restore cursor
PG	reverse-video-toggle
PH	graphics-mode toggle
PI	display-key-table for the current terminal (default is [ESC][?] if not specified)
PJ	help
PN	local printing on
PS	local printing off

Key Label Codes

The entries below tell *filePro Plus* what to display in screen prompts. Enter the names of key labels to associate and control what is displayed on your terminal (maximum of eight characters each) in your termcap definition.

Example: The following lines would be entered for a LINUX termcap definition.

```
:L0=F1:L1=F2:L2=F3:L3=F4:L4=ESC ESC:L5=F5:\
:L6=Pg Up:L7=Pg Dn:L8=ESC TAB:L9=TAB:LA=Ctrl-O\
:LB=F6:LC=Ctrl-L:LD=F7:LE=F8:LG=Ctrl-Z:LH=F9\
:LJ=F10:LY=Ctrl-C:LZ=Enter:\
```

Other Code

L0	Insert-character label
L1	delete-character label
L2	insert-line label
L3	delete-line label
L4	save-information label
L5	duplicate-information label
L6	up-tab label
L7	down-tab label
L8	left-tab label
L9	right-tab label
LA	clear-to-end-of-field label
LB	display-fields label
LC	redraw-screen label
LD	enter-print code label
LE	display-print codes label
LF	restore-cursor label
LG	reverse-video toggle label
LH	graphics-mode toggle label
LI	display-key-table label
LJ	help-key label
LY	cancel-key label
LZ	carriage-return-key label

Function

ALTOS 3/5 Key Usage

Key Function	Key(s) to Use	Key in Manual
All filePro Plus programs		
New field, line, carriage return	[Return]	[Return]
Cancel, break	[Del]	[Ctrl][Break]
Save, record	[Esc][Esc]	[Esc]
Home cursor	[Home]	[Home]
Insert, duplicate character	[Ins char]	[F1]
Delete character	[Del char]	[F2]
Insert line	[Ins line]	[F3]
Delete line	[Del line]	[F4]
Cursor up	[Up Arrow]	[Up Arrow]
Cursor down	[Down Arrow]	[Down Arrow]
Cursor left	[Left Arrow]	[Left Arrow]
Cursor right	[Right Arrow]	[Right Arrow]
Up or previous tab	[Ctrl][F]	[PgUp]
Down or next tab	[Ctrl][N]	[PgDn]
Right tab	[Tab]	[Tab]
Left tab	[Esc][Tab]	[Shift][Tab]
Clear to end of line, field	[Ctrl][O]	[Ctrl][End]
Restore cursor	not applicable	not applicable
Redraw screen	[Ctrl][L]	not applicable
Display key table	[Esc][?]	[Esc][?]
Help	[Help]	[F10]
Toggle Insert Mode	[Ctrl][Z]	[Alt][F9]
Define Screens		
Resolve Fields	[Ctrl][R]	[F5]
Display Fields	[Ctrl][F]	[F6]
Box Functions	[Ctrl][E]	[F7]
Extended Functions	[Ctrl][T]	[F8]
Toggle Graphics	[Ctrl][G]	[F9]
Toggle Reverse Video	[Ctrl][Z]	[Alt][F9]
Define Output		
Enter Print Codes	[Ctrl][R]	[F5]
Display Fields	[Ctrl][F]	[F6]
Box Functions	[Ctrl][E]	[F7]
Extended Functions	[Ctrl][T]	[F8]
Toggle Graphics	[Ctrl][G]	[F9]
Define Processing Tables		
Define Lookups	[Ctrl][R]	[F5]
Display Fields	[Ctrl][F]	[F6]
Go to Line/String	[Ctrl][G]	[F9]
Inquire, Update, Add		
Duplicate	[Ctrl][R]	[F5]
Browse Lookup	[Ctrl][F]	[F6]

ANSI Console Key Usage

Key Function	Key(s) to Use	Key in Manual (DOS/Windows)
All filePro Plus programs		
New field, line, carriage return	[Return]	[Return]
Cancel, break	[Ctrl][Del]	[Ctrl][Break]
Save, record	[Esc][Esc]	[Esc]
Home cursor	[Home]	[Home]
Insert, duplicate character	[F1]	[F1]
Delete character	[F2]	[F2]
Insert line	[F3]	[F3]
Delete line	[F4]	[F4]
Cursor up	[Up Arrow]	[Up Arrow]
Cursor down	[Down Arrow]	[Down Arrow]
Cursor left	[Left Arrow]	[Left Arrow]
Cursor right	[Right Arrow]	[Right Arrow]
Up or previous tab	[PgUp]	[PgUp]
Down or next tab	[PgDn]	[PgDn]
Right tab	[Tab]	[Tab]
Left tab	[Shift][Tab]	[Shift][Tab]
Clear to end of line, field	[Ctrl][O]	[Ctrl][End]
Restore cursor	not applicable	not applicable
Redraw screen	[Ctrl][L]	not applicable
Show key table	[Esc][?]	[Alt][F10]
Version No. (when at Menu)	[Shift][Tab]	[Shift][Tab]
Help	[F10]	[F10]
Toggle Insert Mode	[Ctrl][Z]	[Alt][F9]
Define Screens		
Resolve Fields	[F5]	[F5]
Display Fields	[F6]	[F6]
Box Functions	[F7]	[F7]
Extended Functions	[F8]	[F8]
Toggle Graphics	[F9]	[F9]
Toggle Reverse Video	[Ctrl][Z]	[ALT][F9]
Define Output		
Enter Print Codes	[F5]	[F5]
Display Fields	[F6]	[F6]
Box Functions	[F7]	[F7]
Extended Functions	[F8]	[F8]
Toggle Graphics	[F9]	[F9]
Define Processing Tables		
Define Lookups	[F5]	[F5]
Display Fields	[F6]	[F6]
Go to Line/String	[F9]	[F9]
Inquire, Update, Add		
Duplicate	[F5]	[F5]
Browse Lookup	[F6]	[F6]

AT&T 4410/5410 Key Usage

Key Function	Key(s) to Use	Key in Manual
All filePro Plus programs		
New field, line, carriage return	[Return]	[Return]
Cancel, break	[Del]	[Ctrl][Break]
Save, record	[Esc][Esc]	[Esc]
Home cursor	[Ctrl][A]	[Home]
Insert, duplicate character	[F1]	[F1]
Delete character	[F2]	[F2]
Insert line	[F3]	[F3]
Delete line	[F4]	[F4]
Cursor up	[Up Arrow]	[Up Arrow]
Cursor down	[Down Arrow]	[Down Arrow]
Cursor left	[Left Arrow]	[Left Arrow]
Cursor right	[Right Arrow]	[Right Arrow]
Up or previous tab	[Ctrl][P]	[PgUp]
Down or next tab	[Ctrl][N]	[PgDn]
Right tab	[Tab]	[Tab]
Left tab	[Esc][Tab]	[Shift][Tab]
Clear to end of line/field	[Ctrl][O]	[Ctrl][End]
Redraw screen	[Ctrl][L]	not applicable
Display key table	[Esc][?]	[Esc][?]
Help	[F8]	[F10]
Toggle Insert Mode	[Ctrl][Z]	[Alt][F9]
Define Screens		
Resolve Fields	[Ctrl][R]	[F5]
Display Fields	[Ctrl][F]	[F6]
Box Functions	[Ctrl][Y]	[F7]
Extended Functions	[Ctrl][T]	[F8]
Toggle Graphics	[Ctrl][G]	[F9]
Toggle Reverse Video	[Ctrl][X]	[Alt][F9]
Define Output		
Enter Print Codes	[Ctrl][R]	[F5]
Display Fields	[Ctrl][F]	[F6]
Box Functions	[Ctrl][V]	[F7]
Extended Functions	[Ctrl][T]	[F8]
Define Processing Tables		
Display Fields	[Ctrl][F]	[F6]
Toggle Graphics	[Ctrl][G]	[F9]
Go to line/string	[Ctrl][X]	[Alt][F9]
Inquire, Update, Add		
Duplicate	[Ctrl][R]	[F5]
Browse Lookup	[Ctrl][F]	[F6]

DEC VT 100 Key Usage

Key Function	Key(s) to Use	Key in Manual
All filePro Plus programs		
New field, line, carriage return	[Return]	[Return]
Cancel, break	[Del]	[Ctrl][Break]
Insert, duplicate character	[Chr Ins]	[F1]
Delete character	[Chr Del]	[F2]
Insert line	[Line Ins]	[F3]
Delete line	[Line Del]	[F4]
Cursor up	[Up Arrow]	[Up Arrow]
Cursor down	[Down Arrow]	[Down Arrow]
Cursor left	[Left Arrow]	[Left Arrow]
Cursor right	[Right Arrow]	[Right Arrow]
Up or previous tab	[Ctrl][P]	[PgUp]
Down or 1 next tab	[Ctrl][N]	[PgDn]
Right tab	[Tab]	[Tab]
Left tab	[Backtab]	[Shift][Tab]
Clear to end of line, field	[Ln Erase]	[Ctrl][End]
Restore cursor	not applicable	Not applicable
Redraw screen	[Pg Erase]	Not applicable
Display key table	[Esc][?]	[Esc][?]
Help	[Ctrl][X]	[F10]
Toggle Insert Mode	[Ctrl][Z]	[Alt][F9]
Define Screens		
Resolve Fields	[Ctrl][R]	[F5]
Display Fields	[Ctrl][F]	[F6]
Box Functions	[Ctrl][E]	[F7]
Extended Functions	[Ctrl][T]	[F8]
Toggle Graphics	[Ctrl][G]	[F9]
Toggle Reverse Video	[Ctrl][Z]	[Alt][F9]
Define Output		
Enter Print Codes	[Ctrl][R]	[F5]
Display Fields	[Ctrl][F]	[F6]
Box Functions	[Ctrl][E]	[F7]
Extended Functions	[Ctrl][T]	[F8]
Toggle Graphics	[Ctrl][G]	[F9]
Define Processing Tables		
Define Lookups	[Ctrl][R]	[F5]
Display Fields	[Ctrl][E]	[F6]
Go to Line/String	[Ctrl][G]	[F9]
Inquire, Update, Add		
Duplicate	[Ctrl][R]	[F5]
Browse Lookup	[Ctrl][F]	[F6]

IBM-3151 Key Usage

Key Function	Key(s) to Use	Key in Manual
All filePro Plus programs		
New field, line, carriage return	[Return]	[Return]
Cancel, break	[Ctrl][C]	[Ctrl][Break]
Save, record	[Esc][Esc]	[Esc]
Home cursor	[Home]	[Home]
Insert, duplicate character	[F1]	[F1]
Delete character	[F2]	[F2]
Insert line	[F3]	[F3]
Delete line	[F4]	[F4]
Cursor up	[Up Arrow]	[Up Arrow]
Cursor down	[Down Arrow]	[Down Arrow]
Cursor left	[Left Arrow]	[Left Arrow]
Cursor right	[Right Arrow]	[Right Arrow]
Up or previous tab	[Ctrl][P]	[PgUp]
Down or next tab	[Ctrl][N]	[PgDn]
Right tab	[Tab →]	[Tab]
Left tab	[← Tab]	[Shift][Tab]
Clear to end of line	[Erase EOF]	[Ctrl][end]
Redraw screen	[Ctrl][L]	not applicable
Display key table	[Esc][?]	[Esc][?]
Help	[F12]	[F10]
Toggle Insert Mode	[Ctrl][Z]	[Alt][F9]
Define Screens		
Resolve Fields	[F5]	[F5]
Display Fields	[F6]	[F6]
Box Functions	[F7]	[F7]
Extended Functions	[F8]	[F8]
Toggle Graphics	[F9]	[F9]
Toggle Reverse Video	[Ctrl][X]	[Alt][F9]
Define Output		
Enter Print Codes	[F5]	[F5]
Display Fields	[F6]	[F6]
Box Functions	[F7]	[F7]
Extended Functions	[F8]	[F8]
Toggle Graphics	[F9]	[F9]
Define Processing Tables		
Define Lookups	[F5]	[F5]
Display Fields	[F6]	[F6]
Go to Line/String	[F9]	[F9]
Inquire, Update, Add		
Duplicate	[F5]	[F5]
Browse Lookup	[F6]	[F6]

IBM-3164 Key Usage

Key Function	Key(s) to Use	Key in Manual
All filePro Plus programs		
New field, line, carriage return	[Return]	[Return]
Cancel, break	[Del]	[Ctrl][Break]
Save, record	[Esc][Esc]	[Esc]
Home cursor	[Home]	[Home]
Insert, duplicate character	[Ctrl][A]	[F1]
Delete character	[Ctrl][B]	[F2]
Insert line	[Ctrl][D]	[F3]
Delete line	[Ctrl][U]	[F4]
Cursor up	[Up Arrow]	[Up Arrow]
Cursor down	[Down Arrow]	[Down Arrow]
Cursor left	[Left Arrow]	[Left Arrow]
Cursor right	[Right Arrow]	[Right Arrow]
Up or previous tab	[Ctrl][P]	[PgUp]
Down or next tab	[Ctrl][N]	[PgDn]
Right tab	[Tab]	[Tab]
Left tab	[Esc][Tab]	[Shift][Tab]
Clear to end of line	[Ctrl][O]	[Ctrl][end]
Redraw screen	[Ctrl][L]	not applicable
Display key table	[Esc][?]	[Esc][?]
Help	[Ctrl][Z]	[F10]
Toggle Insert Mode	[Ctrl][Z]	[Alt][F9]
Define Screens		
Resolve Fields	[Ctrl][R]	[F5]
Display Fields	[Ctrl][F]	[F6]
Box Functions	[Ctrl][Y]	[F7]
Extended Functions	[Ctrl][T]	[F8]
Toggle Graphics	[Ctrl][G]	[F9]
Toggle Reverse Video	[Ctrl][X]	[Alt][F9]
Define Output		
Enter Print Codes	[Ctrl][R]	[F5]
Display Fields	[Ctrl][F]	[F6]
Box Functions	[Ctrl][Y]	[F7]
Extended Functions	[Ctrl][T]	[F8]
Toggle Graphics	[Ctrl][G]	[F9]
Define Processing Tables		
Define Lookups	[Ctrl][R]	[F5]
Display Fields	[Ctrl][F]	[F6]
Go to Line/String	[Ctrl][G]	[F9]
Inquire, Update, Add		
Duplicate	[Ctrl][R]	[F5]
Browse Lookup	[Ctrl][F]	[F6]

TANDY DT-100 Key Usage

Key Function	Key(s) to Use	Key in Manual
All filePro Plus programs		
New field, line, carriage return	[Return]	[Return]
Cancel, break	[Delete]	[Ctrl][Break]
Save, record	[Esc][Esc]	[Esc]
Home cursor	[Home]	[Home]
Insert, duplicate character	[FF1]	[F1]
Delete character	[FF2]	[F2]
Insert line	[FF3]	[F3]
Delete line	[FF4]	[F4]
Cursor up	[Up Arrow]	[Up Arrow]
Cursor down	[Down Arrow]	[Down Arrow]
Cursor left	[Left Arrow]	[Left Arrow]
Cursor right	[Right Arrow]	[Right Arrow]
Up or previous tab	[Ctrl][P]	[PgUp]
Down or next tab	[Ctrl][N]	[PgDn]
Right tab	[Tab]	[Tab]
Left tab	[Esc][Tab]	[Shift][Tab]
Clear to end of line	[Ctrl][O]	[Ctrl][end]
Redraw screen	[Ctrl][L]	not applicable
Display key table	[Esc][?]	[Esc][?]
Help	[F15]	[F10]
Toggle Insert Mode	[Ctrl][Z]	[Alt][F9]
Define Screens		
Resolve Fields	[Ctrl][R]	[F5]
Display Fields	[Ctrl][F]	[F6]
Box Functions	[Ctrl][E]	[F7]
Extended Functions	[Ctrl][T]	[F8]
Toggle Graphics	[Ctrl][G]	[F9]
Toggle Reverse Video	[Ctrl][Z]	[Alt][F9]
Define Output		
Enter Print Codes	[Ctrl][R]	[F5]
Display Fields	[Ctrl][F]	[F6]
Box Functions	[Ctrl][E]	[F7]
Extended Functions	[Ctrl][T]	[F8]
Toggle Graphics	[Ctrl][G]	[F9]
Define Processing Tables		
Define Lookups	[Ctrl][R]	[F5]
Display Fields	[Ctrl][F]	[F6]
Go to Line/String	[Ctrl][G]	[F9]
Inquire, Update, Add		
Duplicate	[Ctrl][R]	[F5]
Browse Lookup	[Ctrl][F]	[F6]

TANDY DT-110 Key Usage

Key Function	Key(s) to Use	Key in Manual
All filePro Plus programs		
New field, line, carriage return	[Enter]	[Return]
Cancel, break	[Del]	[Ctrl][Break]
Save, record	[Esc][Esc]	[Esc]
Home cursor	[Home]	[Home]
Insert, duplicate character	[F1]	[F1]
Delete character	[F2]	[F2]
Insert line	[F3]	[F3]
Delete line	[F4]	[F4]
Cursor up	[Up Arrow]	[Up Arrow]
Cursor down	[Down Arrow]	[Down Arrow]
Cursor left	[Left Arrow]	[Left Arrow]
Cursor right	[Right Arrow]	[Right Arrow]
Up or previous tab	[PgUp]	[PgUp]
Down or next tab	[PgDn]	[PgDn]
Right tab	[Tab]	[Tab]
Left tab	[Esc][Tab]	[Shift][Tab]
Clear to end of line, field	[Ctrl][O]	[Ctrl][End]
Restore cursor	not applicable	not applicable
Redraw screen	[Ctrl][L]	not applicable
Display key table	[Esc][?]	[Esc][?]
Help	[F10]	[F10]
Toggle Insert Mode	[Ctrl][Z]	[Alt][F9]
Define Screens		
Resolve Fields	[F5]	[F5]
Display Fields	[F6]	[F6]
Box Functions	[F7]	[F7]
Extended Functions	[F8]	[F8]
Toggle Graphics	[F9]	[F9]
Toggle Reverse Video	[Ctrl][Z]	[Alt][F9]
Define Output		
Enter Print Codes	[F5]	[F5]
Display Fields	[F6]	[F6]
Box Functions	[F7]	[F7]
Extended Functions	[F8]	[F8]
Toggle Graphics	[F9]	[F9]
Define Processing Tables		
Define Lookups	[F5]	[F5]
Display Fields	[F6]	[F6]
Go to Line/String	[F9]	[F9]
Inquire, Update, Add		
Duplicate	[F5]	[F5]
Browse Lookup	[F6]	[F6]

UNISYS UVT-1220 Key Usage

Key Function	Key(s) to Use	Key in Manual
All filePro Plus programs		
New field, line, carriage return	[Return]	[Return]
Cancel, break	[Del]	[Ctrl][Break]
Save, record	[Esc][Esc]	[Esc]
Home cursor	[Ctrl][W]	[Home]
Insert, duplicate character	[FF1]	[F1]
Delete character	[FF2]	[F2]
Insert line	[FF3]	[F3]
Delete line	[FF4]	[F4]
Cursor up	[Up Arrow]	[Up Arrow]
Cursor down	[Down Arrow]	[Down Arrow]
Cursor left	[Left Arrow]	[Left Arrow]
Cursor right	[Right Arrow]	[Right Arrow]
Up or previous tab	[Ctrl][P]	[PgUp]
Down or next tab	[Ctrl][N]	[PgDn]
Right tab	[Tab]	[Tab]
Left tab	[Esc][Tab]	[Shift][Tab]
Clear to end of line, field	[Ctrl][O]	[Ctrl][End]
Restore cursor	not applicable	not applicable
Redraw screen	[Ctrl][L]	not applicable
Display key table	[Esc][?]	[F10]
Toggle Insert Mode	[Ctrl][Z]	[Alt][F9]
Define Screens		
Resolve Fields	[Ctrl][R]	[F5]
Display Fields	[Ctrl][F]	[F6]
Box Functions	[Ctrl][E]	[F7]
Extended Functions	[Ctrl][T]	[F8]
Toggle Graphics	[Ctrl][G]	[F9]
Toggle Reverse Video	[Ctrl][Z]	[Alt][F9]
Define Output		
Enter Print Codes	[Ctrl][R]	[F5]
Display Fields	[Ctrl][F]	[F6]
Box Functions	[Ctrl][E]	[F7]
Extended Functions	[Ctrl][T]	[F8]
Toggle Graphics	[Ctrl][G]	[F9]
Define Processing Tables		
Define Lookups	[Ctrl][R]	[F5]
Display Fields	[Ctrl][F]	[F6]
Go to Line/String	[Ctrl][G]	[F9]
Inquire, Update, Add		
Duplicate	[Ctrl][R]	[F5]
Browse Lookup	[Ctrl][F]	[F6]

WYSE50 Key Usage

Key Function	Key(s) to Use	Key in Manual
All filePro Plus programs		
New field, line, carriage return	[Return]	[Return]
Cancel, break	[Del]	[Ctrl][Break]
Save, record	[Esc][Esc]	[Esc]
Home cursor	[Home]	[Home]
Insert, duplicate character	[Ins Char]	[F1]
Delete character	[Del Char]	[F2]
Insert line	[Ins Line]	[F3]
Delete line	[Del Line]	[F4]
Cursor up	[Up Arrow]	[Up Arrow]
Cursor down	[Down Arrow]	[Down Arrow]
Cursor left	[Left Arrow]	[Left Arrow]
Cursor right	[Right Arrow]	[Right Arrow]
Up or previous tab	[Prev Page]	[PgUp]
Down or next tab	[Next Page]	[PgDn]
Right tab	[Tab]	[Tab]
Left tab	[Shift][Tab]	[Shift][Tab]
Clear to end of line, field	[CtrlLine]	[Ctrl][End]
Redraw screen	[CtrlScrn]	not applicable
Display key table	[Esc][?]	[Esc][?]
Help	[F5]	[F10]
Toggle Insert Mode	[Ctrl][Z]	[Alt][F9]
Define Screens		
Resolve Fields	[Ctrl][R]	[F5]
Display Fields	[Ctrl][F]	[F6]
Box Functions	[Ctrl][E]	[F7]
Extended Functions	[Ctrl][T]	[F8]
Toggle Graphics	[Ctrl][G]	[F9]
Toggle Reverse Video	[Ctrl][Z]	[Alt][F9]
Define Output		
Enter Print Codes	[Ctrl][R]	[F5]
Display Fields	[Ctrl][F]	[F6]
Box Functions	[Ctrl][E]	[F7]
Extended Functions	[Ctrl][T]	[F8]
Toggle Graphics	[Ctrl][G]	[F9]
Define Processing Tables		
Define Lookups	[Ctrl][R]	[F5]
Display Fields	[Ctrl][F]	[F6]
Go to line/string	[Ctrl][G]	[F9]
Inquire, Update, Add		
Duplicate	[Ctrl][R]	[F5]
Browse Lookup	[Ctrl][F]	[F6]

WYSE60 Key Usage

Key Function	ASCII Keyboard	AT Keyboard	Key in Manual
All filePro Plus programs			
New field, line, carriage return	[Return]	[Enter]	[Return]
Cancel, break	Break]	[Del]	[Ctrl][Break]
Save, record	Esc][Esc]	[Esc][Esc]	[Esc]
Home cursor	Home]	[Home]	[Home]
Insert, duplicate character	[FF1]	[F1]	[F1]
Delete character	[FF2]	[F2]	[F2]
Insert line	[FF3]	[F3]	[F3]
Delete line	[FF4]	[F4]	[F4]
Cursor up	[Up Arrow]	[Up Arrow]	[Up Arrow]
Cursor down	[Down Arrow]	[Down Arrow]	[Down Arrow]
Cursor left	[Left Arrow]	[Left Arrow]	[Left Arrow]
Cursor right	[Right Arrow]	[Right Arrow]	[Right Arrow]
Up or previous tab	[Ctrl][P]	[PgUp]	[PgUp]
Down or next tab	[Ctrl][N]	[PgDn]	[PgDn]
Right tab	[Tab]	[Tab]	[Tab]
Left tab	[Shift][Tab]	[Shift][Tab]	[Shift][Tab]
Clear to end of line, field	[Ctrl][O]	[Ctrl][O]	[Ctrl][End]
Restore cursor	not appl	not appl	NA
Redraw screen	[F1O]	[Ctrl][W]	NA
Display key table	[Esc][?]	[Esc][?]	[Esc][?]
Help	[F15]	[F1O]	[F1O]
Toggle Insert Mode	[Ctrl][Z]	[Alt][F9]	[Alt][F9]
Define Screens			
Resolve Fields	[Ctrl][R]	[F5]	[F5]
Display Fields	[Ctrl][F]	[F6]	[F6]
Box Functions	[Ctrl][E]	[F7]	[F7]
Extended Functions	[Ctrl][T]	[F8]	[F8]
Toggle Graphics	[Ctrl][G]	[E9]	[F9]
Toggle Reverse Video	[Ctrl][Z]	[Ctrl][Z]	[Alt][F9]
Define Output			
Enter Print Codes	[Ctrl][R]	[F5]	[F5]
Display Fields	[Ctrl][F]	[F6]	[F6]
Box Functions	[Ctrl][E]	[F7]	[F7]
Extended Functions	[Ctrl][T]	[F8]	[F8]
Toggle Graphics	[Ctrl][G]	[F9]	[F9]
Define Processing Tables			
Define Lookups	[Ctrl][R]	[F5]	[F5]
Display Fields	[Ctrl][F]	[F6]	[F6]
Go to Line/String	[Ctrl][G]	[F9]	[F9]
Inquire, Update, Add			
Duplicate	[Ctrl][R]	[F5]	[F5]
Browse Lookup	[Ctrl][F]	[F6]	[F6]

WYSE75 Key Usage

Key Function	Key(s) to Use	Key in Manual
All filePro Plus programs		
New field, line, carriage return	[Return]	[Return]
Cancel, break	[Delete]	[Ctrl][Break]
Save, record	[Esc][Esc]	[Esc]
Home cursor	[Home]	[Home]
Insert, duplicate character	[FF1]	[F1]
Delete character	[FF2]	[F2]
Insert line	[FF3]	[F3]
Delete line	[FF4]	[F4]
Cursor up	[Up Arrow]	[Up Arrow]
Cursor down	[Down Arrow]	[Down Arrow]
Cursor left	[Left Arrow]	[Left Arrow]
Cursor right	[Right Arrow]	[Right Arrow]
Up or previous tab	[Ctrl][F]	[PgUp]
Down or next tab	[Ctrl][N]	[PgDn]
Right tab	[Tab]	[Tab]
Left tab	[Esc][Tab]	[Shift][Tab]
Clear to end of line, field	[Ctrl][O]	[Ctrl][End]
Redraw screen	[Ctrl][L]	not applicable
Display key table	[Esc][?]	[Esc][?]
Help	[F15]	[F10]
Toggle Insert Mode	[Ctrl][Z]	[Alt][F9]
Define Screens		
Resolve Fields	[Ctrl][R]	[F5]
Display Fields	[Ctrl][F]	[F6]
Box Functions	[Ctrl][E]	[F7]
Extended Functions	[Ctrl][T]	[F8]
Toggle Graphics	[Ctrl][G]	[F9]
Toggle Reverse Video	[Ctrl][Z]	[Alt][F9]
Define Output		
Enter Print Codes	[Ctrl][R]	[F5]
Display Fields	[Ctrl][F]	[F6]
Box Functions	[Ctrl][E]	[F7]
Extended Functions	[Ctrl][T]	[F8]
Toggle Graphics	[Ctrl][G]	[F9]
Define Processing Tables		
Define Lookups	[Ctrl][R]	[F5]
Display Fields	[Ctrl][F]	[F6]
Go to Line/String	[Ctrl][G]	[F9]
Inquire, Update, Add		
Duplicate	[Ctrl][R]	[F5]
Browse Lookup	[Ctrl][F]	[F6]

WYSE85 Key Usage

Key Function	Key(s) to Use	Key in Manual
All filePro Plus programs		
New field, line, carriage return	[Return]	[Return]
Cancel, break	[Del]	[Ctrl][Break]
Save record	[Esc][Esc]	[Esc]
Home cursor	[Home]	[Home]
Insert duplicate character	[FF1]	[F1]
Delete character	[FF2]	[F2]
Insert line	[FF3]	[F3]
Delete line	[FF4]	[F4]
Cursor up	[Up Arrow]	[Up Arrow]
Cursor down	[Down Arrow]	[Down Arrow]
Cursor left	[Left Arrow]	[Left Arrow]
Cursor right	[Right Arrow]	[Right Arrow]
Up or previous tab	[Ctrl][F]	[PgUp]
Down or next tab	[Ctrl][N]	[PgDn]
Right tab	[Tab]	[Tab]
Left tab	[Esc][Tab]	[Shift][Tab]
Clear to end of line, field	[Ctrl][O]	[Ctrl][End]
Restore cursor	not applicable	not applicable
Redraw cursor	[Ctrl][L]	not applicable
Display key table	[Esc][?]	[Esc][?]
Help	[Help]	[F10]
Toggle Insert Mode	[Ctrl][Z]	[Alt][F9]
Define Screens		
Resolve Fields	[Ctrl][R]	[F5]
Display Fields	[Ctrl][F]	[F6]
Box Functions	[Ctrl][E]	[F7]
Extended Functions	Ctrl][T]	[F8]
Toggle Graphics	[Ctrl][G]	[F9]
Toggle Reverse Video	[Ctrl][Z]	[Alt][F10]
Define Output		
Enter Print Codes	[Ctrl][R]	[F5]
Display Fields	[Ctrl][F]	[F6]
Box Functions	[Ctrl][E]	[F7]
Extended Functions	[Ctrl][T]	[F8]
Toggle Graphics	[Ctrl][G]	[F9]
Define Processing Tables		
Define Lookups	[Ctrl][R]	[F5]
Display Fields	[Ctrl][F]	[F6]
Go to Line/String	[Ctrl][G]	[F9]
Inquire, Update, Add		
Duplicate	[Ctrl][R]	[F5]
Browse Lookup	[Ctrl][F]	[F6]

YOUR TERMINAL Key Usage

Key Function	Key(s) to Use	Key in Manual
All filePro Plus programs		
New field, line, carriage return	_____	[Return]
Cancel, break	_____	[Ctrl][Break]
Save record	_____	[Esc]
Home cursor	_____	[Home]
Insert, duplicate character	_____	[F1]
Delete character	_____	[F2]
Insert line	_____	[F3]
Delete line	_____	[F4]
Cursor up	_____	[h]
Cursor down	_____	[i]
Cursor left	_____	[f]
Cursor right	_____	[g]
Up or previous tab	_____	[PgUp]
Down or next tab	_____	[PgDn]
Right tab	_____	[Tab]
Left tab	_____	[Shift][Tab]
Clear to end of line, field	_____	[Ctrl][End]
Restore cursor	_____	Not applicable
Redraw screen	_____	Not applicable
Display key table	_____	[Esc][?]
Help	_____	[F10]
Toggle Insert Mode	_____	[Alt][F9]
Define Screens		
Resolve Fields	_____	[F5]
Display Fields	_____	[F6]
Box Functions	_____	[F7]
Extended Functions	_____	[F8]
Toggle Graphics	_____	[F9]
Toggle Reverse Video	_____	[Alt][F9]
Define Output		
Enter Print Codes	_____	[F5]
Display Fields	_____	[F6]
Box Functions	_____	[F7]
Extended Functions	_____	[F8]
Toggle Graphics	_____	[F9]
Define Processing Tables		
Define Lookups	_____	[F5]
Display Fields	_____	[F6]
Go to Line/String	_____	[F9]
Inquire, Update, Add		
Duplicate	_____	[F5]
Browse Lookup	_____	[F6]

Add Qualifier

Addqual allows you to easily add qualifiers to your files either interactively or through the command line.

v6.1 (6.0.03 USP)

Syntax: addqual [fName|-] [flags]

6.1	-q	qualifier to create
6.1	-x	qualifier to copy indexes from
6.1	-s	silent, no graphics
6.1	-h	syntax help

Autoshuf

Description:

"Autoshuf" synchronizes filePro key and data segments to match a revised map file. "Autoshuf" is a valuable tool for adjusting the filePro data to a revised map file that you have created on another system. As a developer, you often have a need to change a file definition to extend a field length, add a new field, or apply another edit in the "TYPE" column. These same changes will invariably be needed on several clients' systems. Prior to this utility, you would be forced to dial in to the clients system or go on-site to make the map change on each system and wait for files to be re-structured. With autoshuf, you can copy the new map file to map.new, install on your customers' systems and run autoshuf to expand or shrink the key and data segments. The beauty of the program is that it will run unattended in a batch or script file, so you can create upgrade routines and forward to customers on floppy (or other removable media) or e-mail your map changes.

Procedure:

Anytime a map change is made, copy the map from the applicable filePro directory on the source system to map.new in the same filePro directory on the target system.

Note: DO NOT COPY the map or map.tmp files from the source system into the target system or your data on the target system will not be usable. Run autoshuf "filename" where "filename" is the filePro directory that map.new has been copied into on the target system. When autoshuf finishes, your map and map.tmp on the target system will be identical to "map.new".

Syntax:

autoshuf filename

where filename is the filePro directory where you copied "map.new".

Troubleshooting:

Check Environment & Path - If autoshuf fails to run or does not restructure your data, you may not have set your environment and path properly. The same environment variables required to run filePro are required for autoshuf to work properly.

Check for map.new - Make sure that you copied "map.new" to the correct filePro directory.

blobfix

IMPORTANT: Make a backup copy of your data, key, and blob files before executing this command.

Blobfix is designed to correct the pointers from records to the blob file pointing to the blob contents for each record.

Syntax: blobfix filename <-mqualifier>

-mqualifier is optional without -mqualifier, blobfix will run on the non-qualified data source.

Configuration Editor

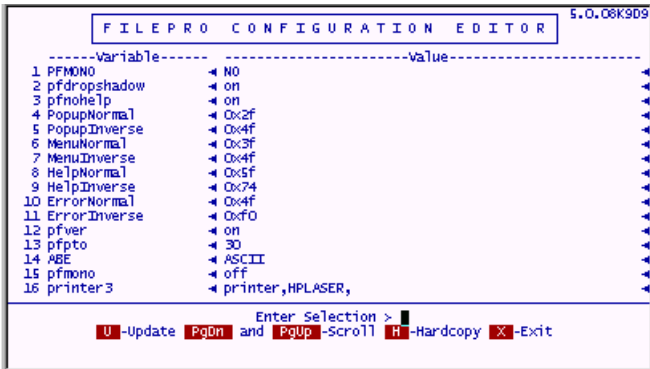
Description

The configuration editor allows you to easily update filePro configuration files to control your filePro environment settings such as screen colors, printers, etc.

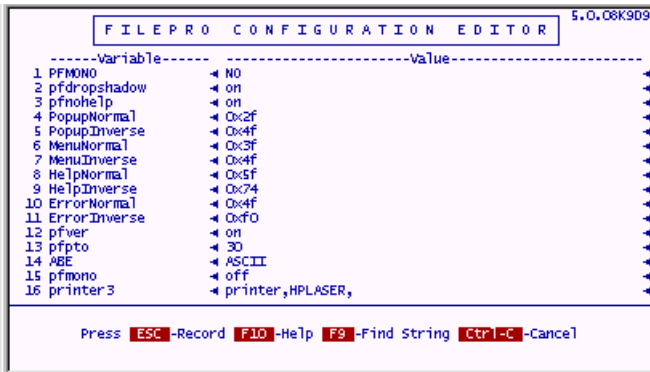
Use the "filePro Utilities" menu option to display the filePro utilities menu.



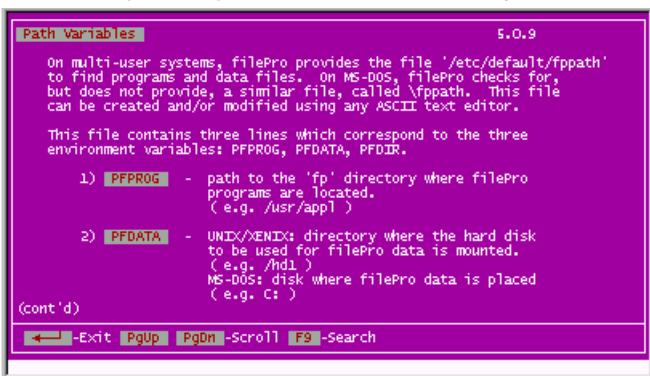
Select option [2] "Configuration Editor" from the filePro Plus Utilities menu to display your configuration settings.



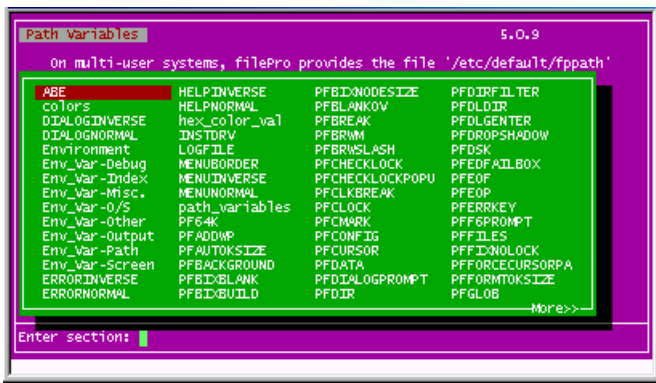
To update your configuration settings, Press [U] to update.



The [F10] help key will display a list of available variables and usage information similar to the following screen.



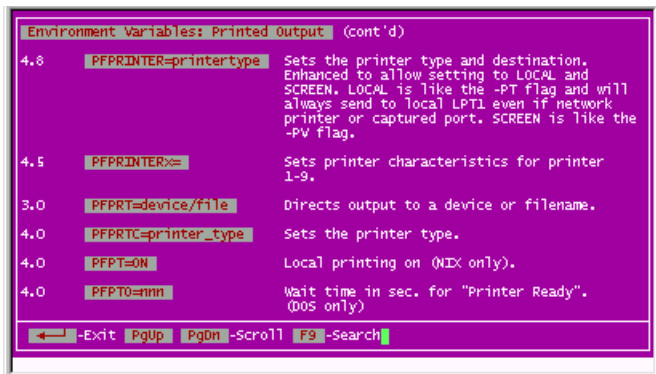
When pressing the [F9] - Search key, an index is displayed for the available filePro variables.



Note that the variables are identified in all uppercase letters while general help categories contain lowercase letters in the index. This is done only to separate items in the index to indicate that the index item is a VARIABLE name rather than a general help category. Environment variables are not case sensitive when using the configuration editor so they can be entered in either upper or lower case.

Classification

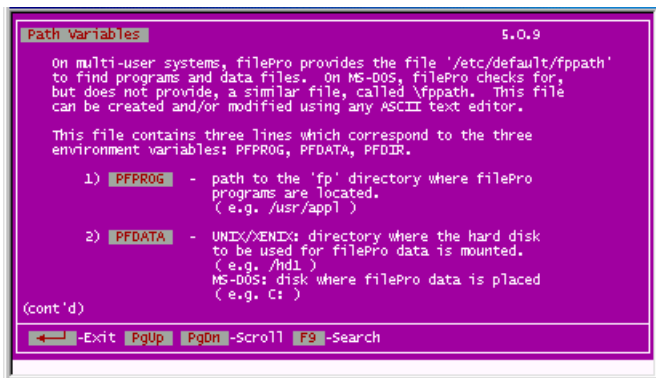
Environment variables are classified based on the utilization e.g. Debug, Screen, Output, etc. Note that these categories are abbreviated in the index as "Env_Var-Debug", "Env_Var-Screen", "Env_Var-Output", etc. The following is an example of the help for environment variables for "Output".



Caution: Be careful when updating your configuration settings to ensure that filePro continues work properly.

Variables that should NOT be added using the filePro Configuration Editor.

Path Variables - Path variables such as PFPROG, PFDATA, PFDSK should not be entered with this editor. These types of variables should be set using a "fppath" file or in the batch file used to start your filePro sessions. Path variables are clearly identified in the help file and in the fPmanual.



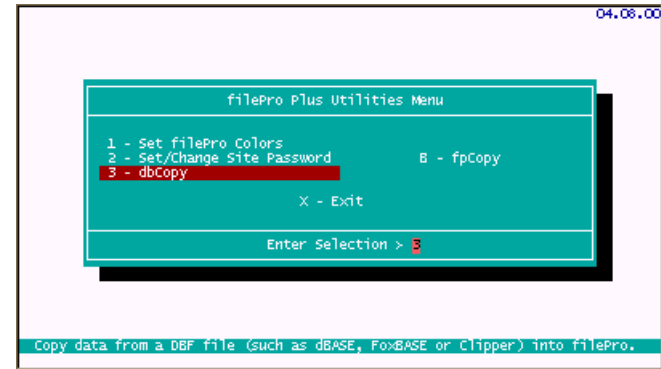
Once you have finished updating your filePro configuration variables, Press [Esc] to record your changes and then exit the configuration editor.

Note: Refer to the topic "References" and sub-topic "Environment Variables" in this manual for more details on usage of environment variables.

dbCopy

Description:

DbCopy provides for copying dBase (xBase) files directory into a filePro data file. This utility is included in the filePro "Utilities Menu" as option "3".



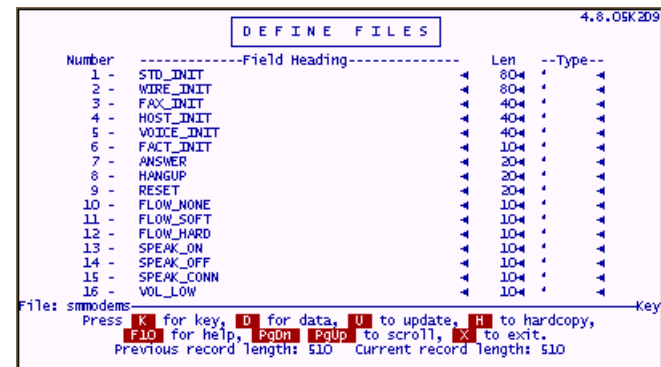
When selecting option "3", you will see the following screen. Enter the path to the dbase(xBase) file and the filePro filename you want to copy to. In this case, we used c:\smmodems and the same name for my filePro filename.



When the Xbase file has been converted, you will see a statistics box as follows;



Your dBase file has been converted. The dbCopy program creates the map file just as if you manually created it using the "Define files" option of the "filePro Plus Main Menu", and has imports the data so that you can use it in filePro. Exit the dBase Conversion program and Utilities Menu and go to the "Define Files" Option "1" of the filePro Plus Main Menu. Select "SMVODEMS" and view the filePro definition created by the dBase conversion program.



Press U to Update then Press ESC to trigger creation of a default screen and press "Y" to create screen 0.

Now we can review the "smmodemf" file in IUA.

Go to "Inquire/Update/Add" Option "B" of the "filePro Plus Main Menu", and select "smmodems".

Select Screen 0 (default screen created by "Define Files") and view a record.

```

SMODEMS

RES_NOANSW: NO ANSWER/MAJ
RES_MISCL1:
RES_MISCL2:
COM_PREFIX: AT
SUPP_ECM: 04
FAX_FLOW: 14
FAX_BAUD: 192004
AUTO_BAUD: 04
COM_LENGTH: 404
SHORT_NAME: E-Tech Bullet

Screen 0          Enter Selection > |          Record: 100
D-Delete, H-Hardcopy, U-Update, X-Exit, F-Print Form, B-Browse

```

Although all the fields in the dBase file are not presented on the Screen 0, you can see that the data has been properly imported while scanning through the records. You can now enhance "srmodesm" by creating additional filePro screens, browse formats etc. to use the data in the filePro format.

The following is a tailored browse format for "srmodesm" built on fields 44, 45 & 46 with the results of the browse definition. Save this format as the "Default browse" format.

```

SHORT NAME      MFR      MODEL

1- STD_INIT      6- FACT_INIT      11- FLOW_SOFT
2- WIRE_INIT     7- ANSWER         12- FLOW_HARD
3- FAX_INIT      8- HANGUP         13- SPEAK_ON
4- HOST_INIT     9- RESET          14- SPEAK_OFF
5- VOICE_INIT    10- FLOW_NONE     15- SPEAK_CONN

Browse Header:
SHORT NAME      MFR      MODEL
Browse Format:
*44             *45             *46

Press ESC To Record Format
Press F5 To Toggle Between Field Headings And Lengths/Edits

```

This will create the default screen browse whenever you enter Inquire/Update/Add and use the "Browse" similar to the following screen.

```

SHORT NAME      MFR      MODEL

Dallas Fax 2496P Dallas Fax      Dallas Fax 2496P
Dallas Fax 2496V Dallas Fax      Dallas Fax 2496V
Dallas Fax 2496VC Dallas Fax      Dallas Fax 2496VC
Dallas Fax 9696  Dallas Fax      Dallas Fax 9696
Designer 9600   Designer        Designer 9600 MNP Modem
Digicom 19.2    Digicom         Digicom Connection Pro 19.2
Digicom PC Classic Digicom        Digicom PC Classic
Digicom PC Classic Digicom        Digicom PC Classic 14.4
Digicom Scout   Digicom         Digicom Scout
Digicom Scout 28.8 Digicom         Digicom Scout 28.8
Digicom Scout Plus Digicom         Digicom Scout Plus
Digicom ScoutPCMCIA Digicom         Digicom ScoutPCMCIA 144
Digitan DF92R   Digitan         Digitan DF92R
Digitan DS144FV Digitan         Digitan DS144FV
Digitan DS92    Digitan         Digitan DS92
Digitan DS92V   Digitan         Digitan DS92V
Digitan DS96F   Digitan         Digitan DS96F
Dynamlink 1428 Dynamlink       Dynamlink 1428 V0(E & H)

ARROWS -Change Record, ← -Select Record, U -Update Record
S -Set Selection, Z -fuzzy search, R -Reset C -Continuous On
F -Format, H -Hardcopy, X -Exit

```

dosetforms

this is a new program located in the fp directory that allows you to bulk select files in your filePro directory to process and set up the various .out files to appear or not to appear when pressing F for forms in a *clerk (UIA) session. The default selection for dosetforms is all filenames, and then .out files that are 1. process only, and 2. all reports. Only .out files that are labels and forms will show in clerk.

doresync Version Ref. 5.8.01

doresync" acts like "autosshuf", except that it doesn't do any restructuring -- it just does the resync and marks the file as "mirroring on" again. You can also pass the filename on the command line, as well as "-H heading". And, unlike autosshuf, it doesn't mark the lockfile as being used by define files, thereby allowing others into the file while it is resyncing

```
/fp/doresync <filename> -h "Heading"
```

Within each filePro directory there is a new file called 'mirror.xml' which contains the mirror status. The format is:

```
<fileProMirrorInfo>  
<Mirror Status='STATUS' When='2015-12-28T23:20:58' User='USERNAME' />  
<Reason>REASON FOR MESSAGE</Reason>  
</fileProMirrorInfo>
```

Where STATUS is either 'off', 'paused', 'on', or 'resync'

Dual Write/Mirror Version 5.8.01

When setting up Dual Write, you need to make sure that you have FFDIR2 and FFDATA2 set in the configuration file.

An example would be:

```
FFDATA2=K:  
FFDIR2=fpmirror  
or  
FFDATA2=\\w.x.y.z\sharename or \\server\sharename  
FFDIR2=fpmirror
```

On NIX platforms you could have this in your config file:

```
FFDATA2=  
FFDIR2=/backup/mirror
```

Then you need to make sure that the path for those variables (including the filepro directory) is set up prior to activating a dual write.

In the first example it would be the path of K:\fpmirror\filepro or for NIX /backup/mirror/filepro. Note that filePro needs read and write permissions.

Then, in Define Files, select the filename that you wish to mirror. Press O for Options.

Press ENTER to get to the last field: Mirror data from this file: Enter Y and then press SAVE to record.

When you SAVE to exit ddefine, it will copy all instances of data, key, index, and blob to the filename located in the mirror path.

This process edits the header of the map to include the trigger for filePro to know this is a dual write or mirrored file.

IMPORTANT NOTE If during normal operations within a mirrored file the path to the mirror becomes unavailable, filePro will detect the error and popup a message warning the user that the mirror has been suspended. Someone will at this point need to reestablish the mirror path and then [resync](#) the files that have been suspended. Within each filePro directory there is a new file called 'mirror.xml' which contains the mirror status. The format is:

```
<fileProMirrorInfo>  
<Mirror Status=STATUS When=2015-12-28T23:20:58 User=USERNAME />  
<Reason>REASON FOR MESSAGE</Reason>  
</fileProMirrorInfo>
```

Where STATUS is either 'off', 'paused', 'on', or 'resync' (See [doresync](#))

DO NOT mirror an alien or ODBC file.

%LOCALAPPDATA%\fptech\fpmirrorwarn contains the flag for warning an end user that there is a problem.

Form filtering version 5.8.02

dosetforms – this is a new program located in the fp directory that allows you to bulk select files in your filePro directory to process and set up the various .out files to appear or not to appear when pressing F for forms in a *clerk (UIA) session. The default selection for dosetforms is all filenames, and then .out files that are 1. process only, and 2. all reports. Only .out files that are labels and forms will show in clerk.

dmoedef enhancement – from the F8 – Options in Define Output, you can select F – Hide or Show Forms. This will allow you to manage the .out files that are hidden from clerk. A list of .out files will appear when you press F. This list will display showing .out file hidden with an asterisk and those not hidden unmarked. You can manually toggle the .out files or you can press F7 to reverse toggle them all. Once you have the list the way you wish it to be, press SAVE to process the new settings for this one directory.

dmoedef also allows you to switch just the .out file that you have selected by selecting F8 – Options. The line “Hide forms from clerk: N” determines rather this form is hidden or not from clerk (UIA).

fpCopy

fpCopy command line flags (Ver 5.8.03.13)

In this update, fpCopy will now accept command line flags so the previous interface does not need to be used.

IMPORTANT: Note below that the flags used for command line use are different for ODBC files. Use Caution when executing this from the command line.

NOTE: If any flags are missing from a command line execution of fpCopy, the interface will trigger for the questions needing answers.

The syntax is:

For regular filePro files: `fpcopy FPname newFPname -FC [1|2|3|4] -RP [1|2|3]`

-FC flags are:

- 1 - Copy File Layout And Formats Only
- 2 - Copy File Layout, Formats, And Data
- 3 - Copy File Layout Only
- 4 - Rename The File

-RP flags are:

- 1 - Assign The Creation Password As The Runtime Password
- 2 - Prompt For The Runtime Password
- 3 - Remove All Runtime Passwords

For ODBC files: `fpcopy FPname newFPname -FC [1|2|3] -RP [1|2|3]`

-FC flags are:

- 1 - Copy File Layout And Formats Only
- 2 - Copy File Layout Only
- 3 - Rename The File

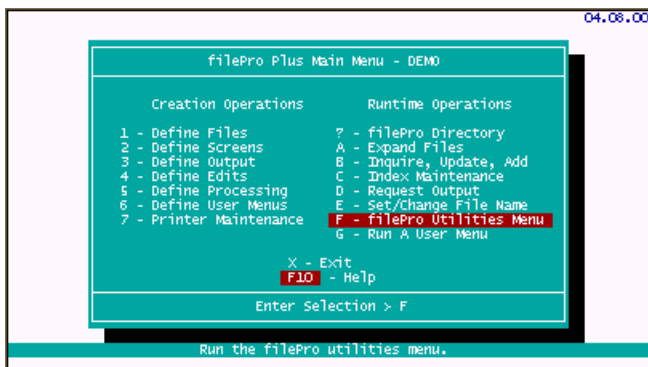
-RP flags are:

- 1 - Assign The Creation Password As The Runtime Password
- 2 - Prompt For The Runtime Password
- 3 - Remove All Runtime Passwords

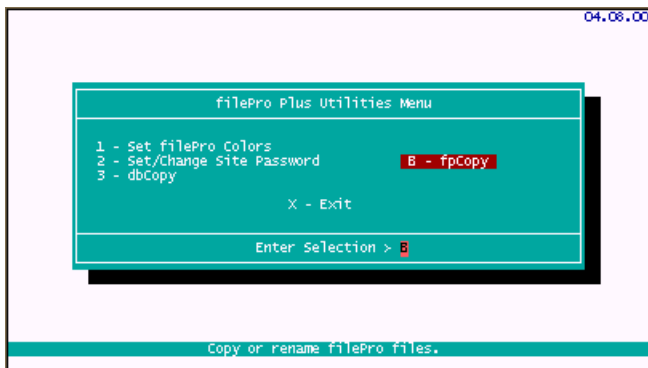
Description:

fpCopy provides for copying one filePro database to another. This utility is included in the filePro "Utilities Menu".

Select Option "F" from the filePro Plus Main Menu to enter the filePro Utilities Menu



Select Option "B"

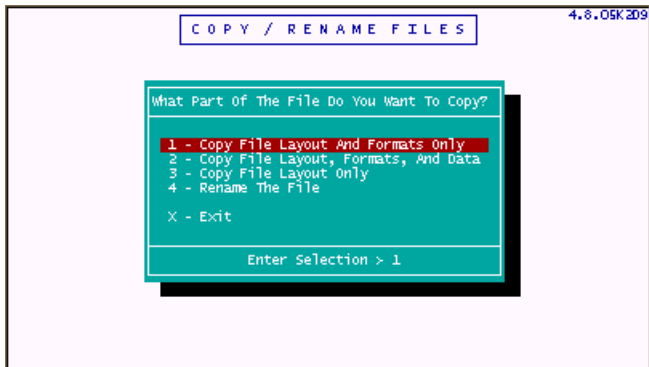


Enter the Copy/Rename from filePro filename and New filename.

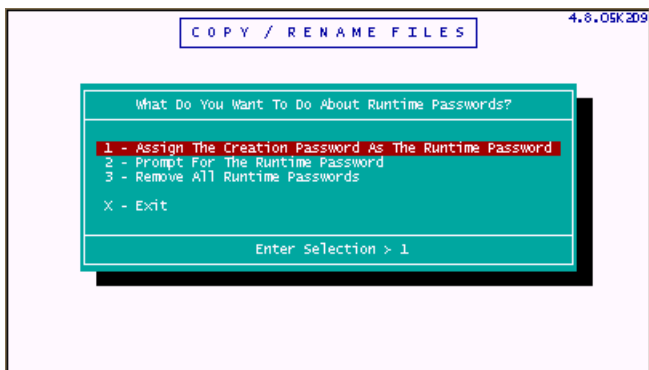


You will then be presented with various options for copying and renaming the file. Select the option that applies for your particular scenario.

Note: Option "2" will copy all layouts, formats, and data. You will probably want to use this option when creating archive files so that you have the screens, reports, edit dictionary, and so forth if you will have a need to periodically retrieve data from the archive.



After selecting the parts of the filePro to copy or rename, you will see the following screen which provides various Runtime password options. Select the option of your choice.



After fpCopy finishes, you will be returned from the filePro "Utilities Menu". You should be able to view the new or renamed filePro file definition, formats, etc. from the "filePro Plus Main Menu".

Pack Files

This program allows a developer to remove deleted records in a file to reclaim space. It can also be used to display statistics and optionally rebuild index files after the packing operation.

v6.1 (6.0.02 USP)

Syntax: fppack [fName|-] [flags]

```
6.1 -H "heading"      Custom title to display in box.
6.1 -E              Don't actually pack the records, just
                   give statistics.
6.1 -R              Rebuild the automatic indexes even if
                   no records were deleted.
6.1 -EX            Skip statistics.
6.1 -C              Skip continue and finished prompts
6.1 -X              Skip rebuilding the auto indexes.
6.1 -M name         Qualifier file name to use.
6.1 -MD            Ask for qualifier with default prompt.
6.1 -MQ "mesg"     Ask for qualifier with "mesg" as the prompt.
6.1 -MA            Use all qualified files & main file.
```

\r Unix/Linux only: \r

```
6.1 -BG            Work in the background.
6.1 -BS            Suppress "completed in background" message.
```

fPtransfer

The fPtransfer utility is an add-on product and used for transferring data between operating systems. However, you can only transfer from the same or older (lower) version to the same or newer (higher) version of filePro. It is highly recommended that you use the same version of fPtransfer for both operating systems. Also, although many versions allow direct transfer of the filePro files using serial communications ports, this method can be problematic due to differences in standards being employed for new hardware. We recommend running fPtransfer from the command line using the "-lf" flag to create and read binary files on the source and target systems respectively. The binary files can then be transferred between systems by any means available e.g. FTP (File Transfer Protocol), tape, CD, etc.

Examples:

```
xfer -t -pn -lf file.dat file1 file2 file3 etc...
```

The above will create "file.dat" for filePro files identified as file1, file2, file3, etc.

```
xfer -r -lf file.dat
```

The above will extract the file(s) to the location identified by the path variables.

Note: Make sure to set the path variables e.g. pfdisk, pfdata, pfdir prior to running fPtransfer and use

You will need to buy copies of fPtransfer for the source and target systems.

To upgrade and/or buy the DOS or UNIX fPtransfer, please contact our sales department at 1-800-847-4740 or email us at sales@fileproplus.com

Freechain

This utility is used to correct record chain problems that affect indexes. Although the freechain is normally maintained through index maintenance and when expanding a file, this utility is also furnished as a separate utility program to repair the freechain pointer. This may be necessary when you receive an error message "Rebuild Freechain" or "Error in freechain".

Syntax:

For NIX

freechain filename [qualifier]

For Windows

freechn.exe filename [qualifier]

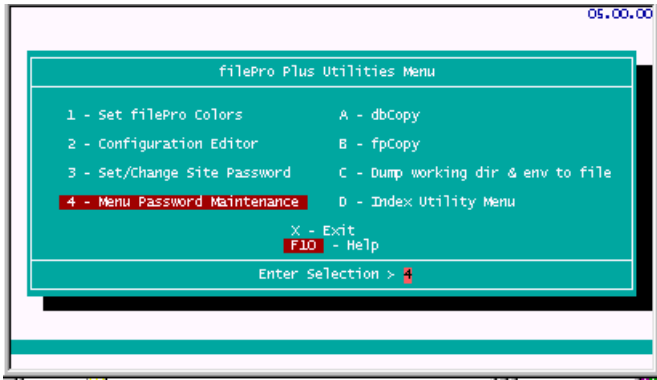
where filename is the filePro file name and [qualifier] is the optional qualified data set.

Licinfo

Provides details with regard to the filePro license. This utility is useful for diagnosing license problems and providing details on the licensed features, user counts, etc.

Menu Password Maintenance

You can assign passwords to each of your user menus with this option. Select the "Utilities" option from the filePro Plus Main menu to show the filePro Utilities menu.



Select a menu name from the available menus.



Select the file that you want to protect with a password and enter your passwords.

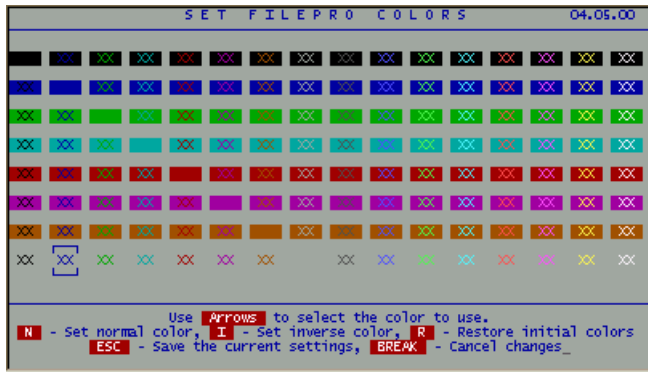


You can enter, change and delete a menu password with this option but must know the password to change a previously "password protected" menu. Menu passwords cannot be removed by simply copying the menu from a system prompt.

Note: The "Define User Menus" option on the filePro Plus main menu can also be used to change menu passwords.

Set filePro's Colors Utility

Under DOS, select F for the filePro Utility Menu, then choose 1 for the filePro Color Utility. This program lets you quickly set the colors for all filePro programs globally. You adjust the normal color and the inverse color and filePro does the rest.



Site Password**Description:**

The site password option is intended to prevent theft of your applications. It prevents copies of your source code from being modified on another system. Keep in mind that if you apply a site password to your own system and copy the processing tables and formats to another system, you will not be able to access these files unless you have the same site password on the other system. This can present problems if you develop or modify systems at various locations.

In the event that you end up with undesired password protection, the passwords can be removed by fP Technologies for a nominal fee. Contact our sales office for details and procedures for removal of the passwords.

SpellEditor

Description:

Allows you to edit personal spell check dictionaries maintained in %pfprog%\fp\spell or as specified by PFSPELLUSERLIST. The utility allows you to add, change & delete words in your personal dictionary.

Swapcpu

This program switches the file formats between little and big endian. (Endian is determined by hardware architecture)

Swapcpu converts a *nix filePro file between little- and big-endian format, without the need to go through xfer. For example, between Intel (little-endian) and AIX (big-endian).

Note that HP-UX comes in two flavors -- Intel (little-endian) and RISC (big-endian).

Usage

sw apcpu [flags] [filename filename filename ...]

Available flags:

-S = skip key/data/files.

-Q = quiet mode (no "already in destination order" messages)

-CN = convert to native format

-CF = convert to foreign format

UID Mapping

Added UID mapping to filePro, ddir/dprodir option F5. This allows for UIDs (User IDs) to be aliased to specific usernames. In the event that a login account is removed from your system, this can be used to maintain the link between the removed login's UID and those stored in filePro, effectively allowing system variables such as @CB and @UB to be maintained.

Windows Only:

This also has the added benefit of allowing @CB and @UB to function on Windows by linking a "pseudo" UID to a given username. These UIDs are automatically generated but can also be manually added. When a user opens filePro and their username does not exist in the UID map file, a UID will be generated for that user. filePro will find the next available UID in the list, starting from 2000, and assign it to that username.

On all platforms, UIDs stored in this program must be unique and in the range 0-65535. Usernames can be duplicated on Unix and Linux platforms, but must be unique on Windows.

Usernames are case-sensitive on Unix and Linux platforms and are case-insensitive on Windows platforms.

Environmental Variables:

PFUIDMAP = /path

Alternate filePro UID map file. (Use full path)

Note: Must be set in the environment.

PFUSEUIDMAP = ON

Allows filePro to do UID mapping. Also expands the maximum username length returned by @CB, @UB, and @ID to 32.

Default: ON

UNPAD

Earlier versions of filePro under HP-UX stored the data on disk in a format that is not compatible with other versions of filePro. There were some additional padding bytes in the binary headers, due to the different CPU architecture. The HP version of filePro has been changed to use a format that is compatible with the other systems that filePro runs on. If you have data from the earlier version of filePro, you need to "unpad" the data to remove the extraneous padding bytes to convert to the compatible format.