TAS
PROFESSIONAL
VERSION 5.1

APPLICATION DEVELOPMENT SYSTEM

4th Generation Language
Relational Database
Btrieve™ Record Manager
Runtime Compiler
Report Writer
Screen Painter
Program Generator

Includes Windows Runtime

www.cassoftware.com
800-358-4222 Fax 909-793-6974 International 909-335-1205

COPYRIGHT

TRADEMARKS

DISCLAIMER

SOFTWARE COPYRIGHT NOTICE

INTENTIONALLY BLANK

SUPPORT POLICIES AND PROCEDURES

CAS. has provided you with complete documentation and a "read me" file to assist you with the operation of the product. However, if you have questions, unexpected problems, update requirements, or a product return, you must initially contact the place of purchase. Please review the following support policies and procedures before requesting assistance.

Before Contacting Product Support

1) Review your manuals thoroughly.

2) Have your product serial number ready.

3) Have your invoice number and date of purchase available.

4) Designate a **single** point of contact for dealing with product support. If that's you, please be located in front of your computer, with the computer on and the software loaded, if possible. You need to be ready to work with the software.

5) Have your manuals within reach.

You must be a registered user to obtain support. If you bought through a dealer, your dealer is your first line of support. You must complete the registration process before support will be available from CAS. For more information on registering please refer to **HOW TO REGISTER YOUR COPY OF TAS PROFESSIONAL 5.1**, which follows the installation instructions in Chapter 1.

Basic Support Program

Source:	Computer.Accounting.Solutions
Fee:	No Charge
Length of Agreement:	30 days (from date of purchase) or a maximum of 1 hour. Each call to CAS support will be counted as 15 minutes.
Purpose:	To assist in the installation of TAS Professional 5.1.
Enrollment:	Registered User (Registration process is complete.)
Support Hotlines:	1:00 pm - 5:00 p.m. PST or PDT Monday through Friday (909) 335-1205
EXCEPTIONS:	Initial no-charge support is **NOT** offered for installations on LANs (local area networks) other than Novell Netware products 2.0 or greater (not including Netware Lite) and Lantastic 6.x or greater, running on DOS version 5.0 or higher. This support is limited to CAS products running on your system and is not support for the LAN software or hardware system itself. Support may be available for a fee. Ask your support representative when you call.

Regular Support Program

If you have purchased this product directly from CAS or wish to get support from CAS the two following options are available.

PER CALL PLAN

Source:	Computer.Accounting.Solutions(CAS)
Fee:	$95.00 per Hour (min of 15 minutes per call). Must have a VISA or M/C credit card available when you contact CAS.

PRE-PAID HOURLY SUPPORT

Source:	Computer Accounting Solutions .
Fee:	$95.00 per hour, pre-paid in 3 hour blocks ($285.00 per block). Each time you contact CAS for a support question you will be charged for the actual time involved and the amount will be deducted from your account. This may also include time not spent on the phone with the support technician. There are no minimum time amounts so you will be charged only for the time used. Also, your account will stay active until the pre-paid amount is fully used or for 3 years, which ever is first.
Availability:	Both the PER CALL and PRE-PAID plans are available from 9:00 am - 5:00 p.m. PST or PDT, Monday through Friday. The support phone number is 909-335-1205.
Other Support:	FAX: (919) 477-1731 WEB: www.business-tools.com

UPDATE PROCEDURES

If you are a registered user and want to obtain the current version of a product, contact:

Product Support Department
9:00 A.M. - 5:00 EST or EDT
Monday through Friday
(800) 358-4222 Email: support@cassoftware.com web: www.cassoftware.com

Before you call, please have the following information about your existing product available:

Product Name
Serial Number
Invoice Number and Date of Purchase

PRODUCT RETURNS

Once the registration process is complete you may no longer return the product. Due to the nature of software and the fact that we don't, for your convenience, copy protect the product, we are unable to offer extended money back guarantees.

If you have purchased this product from a dealer (not directly from Computer Accounting Solutions.) you must make any returns to that dealer. No returns will be accepted more than 30 days after date of purchase.

INTENTIONALLY BLANK

TAS
PROFESSIONAL
VERSION 5.1

Chapter 1

Installation and General Information

INTENTIONALLY BLANK

INTRODUCTION

Welcome to TAS Professional 5.1, a key member of the Business Tools family of computerized business enhancement tools. With TAS Professional 5.1, creating, maintaining, and using computer programs has never been easier. And now, with the introduction of the Windows extensions and runtime, you can easily move to the Windows platform.

TAS Professional 5.1 is a 4GL, a fourth generation language. In general, a 4GL is easier to use than 3GLs such as BASIC or COBOL, but might be more restrictive in what it can do. We feel that TAS Professional 5.1 has combined the best of both worlds. So often a programmer has had to give up the flexibility, size, and speed of a 3GL to get the ease of programming and maintenance of a 4GL. With TAS Professional 5.1, you don't have to! In fact, Pro 5.1 combines the 4GL with a relational database manager and several powerful utilities to offer a complete application development system.

Completely structured programming is available for those who want it, and yet the GOTO/GOSUB commands are there for those who would feel lost without them (even though they aren't necessary). The commands range from the very simple to the very sophisticated. Many complex processes can be handled in a single line. We even allow you, through the use of the translation file, to change the command/option names, if desired. It can't get much more flexible than that! If you are serious about creating useful and easy to use application programs quickly and easily, TAS Professional 5.1 is the tool for you.

If you've upgraded from an earlier version of TAS Professional, you'll be very pleased with some of the enhancements and features we've added. Those of you partial to TAS Pro 3 will be able to convert and compile your code into Pro 5.1. All the utilities and commands you're familiar with are available to you, and you'll have the added benefits of integrated memory management, graphics, use of the serial port, and mouse support that are part of Pro 5.1.

If you made the move to TAS Pro 4.0 and the subsequent releases, you'll be able to recompile your code easily to get the benefits of Pro 5.1 mentioned above. The same commands and functions you used in Pro 4.0 are still there.

Application Development and Runtime Licenses

If you plan to develop applications using TAS Professional 5.1, there are two approaches. If you develop an application in TAS Pro 5.1 completely from scratch (without using source code to any of our products), you may buy an unlimited runtime license and then distribute your application (executable only) without buying runtimes. If you modify, enhance, or otherwise use the source code from any of our products in your application, you must buy a runtime copy of the product(s) whose source code you used for every copy you sell or distribute. Please call us for more details.

NOTE: The Windows Unlimited Runtime is a separate product from the DOS Unlimited Runtime. If you're going to be distributing programs for both DOS and Windows you will need both unlimited runtimes. If you have already purchased the TAS Professional 5.0 (DOS) Unlimited Runtime License you can still use it with version 5.1. Just copy the new TPC50.EXE file that gets installed with version 5.1 onto your existing unlimited runtime disk.

Other Products

Source code for CAS' business management products is written in TAS Professional and is available to developers who would like to build applications based on our products. For example, a developer could obtain runtimes and source code for our Advanced Accounting and Point of Sale Management System products and then modify the code to fit a particular retail operation (e.g., dry cleaners). The developer could then offer this specialized product to customers in the particular industry or market niche. As mentioned above, the developer would need to purchase one runtime copy of each product modified for

each sale of the specialized application in order to obtain the necessary licenses. All of our management products have the necessary Windows modifications including buttons, mouse control, etc.

Brief descriptions of some of Business Tools' other products follow.

Advanced Accounting

The easy-to-use, feature-rich accounting system built to meet the most challenging business requirements. This fully integrated system includes General Ledger, Accounts Receivable, Sales Order, Accounts Payable, Purchase Order, Inventory Control, and Payroll modules. Menus and options are laid out clearly and logically, and every operation runs extremely fast. Take advantage of dozens of features found only in packages costing thousands of dollars.

Job Cost Management System

A fully featured JCMS that will allow you to easily keep track of individual jobs. Now you can really tell whether you're making a profit or not on each job you take. Easy estimating, and simple conversion of estimates to jobs. If you have employees that work at many different tasks, and get paid at many different rates (including different worker's liability rates) this product's for you. Fully integrated with Advanced Accounting.

Bill of Materials

If you build 'kits' or do simple manufacturing and want to be able to keep track of how many units you could build, or easily create build orders from sales orders entered, then BOM is for you. A very easy to use system, it is fully integrated with Advanced Accounting. Allows up to 9 levels of assemblies, and you can easily include both labor and materials in a product. Can instantly show you the actual cost of an item updated for new costs of its parts.

Point of Sale Management System

A POS system designed for any business that has sales that occur at a cash register. With this fast, efficient replacement for your registers, you can easily control how your clerks log in as well as how much information needs to be entered for each sale. You can audit sales activity by clerk, by register, by payment type, by time of day--you name it! Fully integrated with Advanced Accounting, the system offers such capabilities as layaways, split sales, and promotional items.

Please note that a number of third-party systems have been developed in TAS Professional for a variety of applications, ranging from property management to Medicare claims processing. For more information on these third-party TAS Pro applications, please contact your dealer or Business Tools.

The Rest of This Chapter

Please turn to the next page for instructions on how to install TAS Professional 5.1. Following the installation instructions is an overview of the key features of the system, including specifications, file information, special commands for compiling and executing, and so on.

INSTALLATION

What This Package Contains

Please take a few minutes to make sure you received everything necessary to successfully run TAS Professional 5.1. This package should contain the following items:

TAS Professional 5.1 Manual
TAS Professional 5.1 Disk Packet

If either of these items is missing, please contact your dealer immediately for a new copy. If you purchased TAS Professional 5.1 directly from Business Tools, Inc., then contact the Product Support Department.

The multi-user (LAN) version of TAS Professional 5.1 is included and is automatically installed.

What You Need

To install and run TAS Professional 5.1 you will need:

1. An IBM-PC compatible 386 or better (486 or better preferred).
2. A minimum of 4MB of RAM installed on the machine. TAS Pro 5.1 can access any available memory.
3. One floppy drive (3.5" high density) and one fixed hard disk drive with at least 10MB of storage available.
4. DOS 3.3 or higher or Windows 3.1x/Windows for Workgroups (running in 386 Enhanced Mode) or Windows 95.

The TAS Professional 5.1 Disk Packet

The Disk Packet contains your license to the product and four 3.5" high density disks.

NOTE: If you previously purchased TAS Professional 5.0 for DOS and have registered that product, you will only have 3 disks in your TAS Professional 5.1 Disk Packet. You copy of TAS Professional 5.1 has been pre-registered for you. You do not need a REGISTRATION DISK, and will not need to run the registration procedure.

NOTE: The disks are labeled RUN DISK 1, RUN DISK 2, WINDOWS RUNTIME DISK, and REGISTRATION DISK. You will need RUN DISK 1, RUN DISK 2, and the WINDOWS RUNTIME DISK during your initial installation. You will also be able to use the WINDOWS RUNTIME DISK later on to help in configuring individual application subdirectories (see section **Creating a New Subdirectory for Development Purposes** later in this chapter). The WINDOWS RUNTIME is for use with this copy of TAS Professional 5.1 only and is not for general distribution. If you want to distribute Windows programs that you have written, you must purchase a Windows Unlimited Runtime. Please contact Business Tools for more information.

NOTE: You will need the REGISTRATION DISK when you contact Business Tools to unlock the product. Until your product is registered you are limited to 100 records in any one data file. Please be sure not to lose this disk. You will not be able to register your product without it!

Installing TAS Professional 5.1

NOTE: If you will also be installing Advanced Accounting 5.1, please complete the TAS installation first, then see the section **Setting Up Multiple Directories for TAS and Advanced Accounting** later in this chapter before installing Advanced Accounting. That section also contains tips for working with the Advanced Accounting source code.

Installation consists of 2 steps: installing the DOS Development Tools (RUN DISKS 1and 2), then installing the WINDOWS RUNTIME DISK. You will end up with TAS Pro 5.1 for DOS *and* TAS Pro 5.1 for Windows installed on your system.

Installing the DOS Development Tools

These tools consist of the program editor, screen and report format generator, compiler, and other utilities which make up TAS Pro 5.1's DOS-based Application Development Environment, from which you will be able to create both DOS and Windows programs.

If TAS Pro 5.0 for DOS is currently installed on your computer: The default installation path for the version 5.1 Development Tools is C:\TAS50. This is the same directory in which you may have originally installed TAS Pro 5.0 for DOS. When you install the version 5.1 Development Tools, they will overwrite existing files in the TAS50 directory and its SOURCE and SAMPLE subdirectories (including the data dictionary files). If you do not wish this to happen, you should specify an alternative installation path such as \TAS51.

The version 5.1 Development Tools can be installed either through Windows or from a DOS prompt. Please follow the instruction set that applies in your situation:

Windows installation:

1. Insert RUN DISK 1 into your A: or B: drive. If you are running Windows 3.1, from Windows' Program Manager, click on File, then Run. If you are running Windows 95, click on the Start button and choose Run from the menu displayed. Enter A:SETUP (or B:SETUP) as the command line. Click on Install from the installation menu. On the next screen, you may change the installation path if desired. Installation will commence; when prompted, insert RUN DISK 2.

2. If your CONFIG.SYS file requires modification (the only requirement is that FILES=150), the installation routine will offer to make this change for you. If you choose not to have this done, please remember to edit your CONFIG.SYS file later.

3. The installation routine will then offer to change the PATH statement in your AUTOEXEC.BAT file (by adding C:\TAS50 or whatever location you specified as the installation path). Again, if you choose not to have this done for you, please remember to edit your AUTOEXEC.BAT file later.

4. Finally, click on Yes when asked if you wish to have a Program Group and Icons installed. This will create a TAS Pro 5.1 Program Group window and place an icon for TAS Pro 5.1 [DOS] on it.

Once these steps are complete, you should skip ahead to the section titled **'Installing the Windows Runtime'.**

DOS Installation:

1. Insert RUN DISK 1 into your A: or B: drive. From any DOS prompt, type and enter A:INSTALL (or B:INSTALL). On the first installation screen, enter an I to install. The next two screens will allow you to change the drive and directory of the installation path. Installation will commence; when prompted, insert RUN DISK 2.

2. If your CONFIG.SYS file requires modification (the only requirement is that FILES=150), the installation routine will offer to make this change for you. If you choose not to have this done, please remember to edit your CONFIG.SYS file later.

3. The installation routine will then offer to change the PATH statement in your AUTOEXEC.BAT file (by adding C:\TAS50 or whatever location you specified as the installation path). Again, if you choose not to have this done for you, please remember to edit your AUTOEXEC.BAT file later.

4. The installation program must make a temporary modification to your WIN.INI file so that a Program Group and Icons can be added to your Windows setup. Answer 'Y' when asked to proceed with the modification of WIN.INI.

5. Installation is complete. The next time you run Windows, you will be given the option to automatically create a TAS Pro 5.1 Program Group window, to which a TAS Pro 5.1 [DOS] icon will be added.

Installing the Windows Runtime

The WINDOWS RUNTIME DISK can only be installed through Windows. If you are only interested in using TAS Pro 5.1 for DOS, you do not need to install this disk (you can always install it at some later date if you do decide to do some Windows development). If you do not wish to install the Windows Runtime at this time, skip ahead to the section titled **'Final Installation Steps'**.

To install the Windows Runtime:

1. Insert the WINDOWS RUNTIME DISK into your A: or B: drive. If you are running Windows 3.1, from Windows' Program Manager, click on File, then Run. If you are running Windows 95, click on the Start button and choose Run from the menu displayed. Enter A:SETUP (or B:SETUP) as the command line. Click on Install from the installation menu. On the next screen, you may change the installation path if desired (for this initial installation you should specify the same drive and path as you did for the DOS Development Tools). Installation will begin.

2. The installation routine will try to modify the Btrieve section of your WIN.INI file. You should answer 'Y' when asked to confirm this modification.

3. If your CONFIG.SYS file requires modification (the only requirement is that FILES=150), the installation routine will offer to make this change for you. If you choose not to have this done, please remember to edit your CONFIG.SYS file later.

4. The installation routine will then offer to change the PATH statement in your AUTOEXEC.BAT file (by adding C:\TAS50 or whatever location you specified as the installation path). Again, if you choose not to have this done for you, please remember to edit your AUTOEXEC.BAT file later.

5. Finally, click on Yes when asked if you wish to have a Program Group and Icons installed. This will create a TAS Pro 5.1 Program Group window (if it does not already exist) and place an icon for TAS Pro 5.1 [Windows] on it.

Final Installation Steps

If you chose not to have your CONFIG.SYS and AUTOEXEC.BAT files modified for you, you should go ahead and make those changes now.

Next you need to run a program called ADDTPRO, which will add a SET statement to your AUTOEXEC.BAT file (SET TAS50=x:\xxxxxx, where x:\xxxxxx is the installation path you specified above; for example, SET TAS50=C:\TAS50). The ADDTPRO program will also update your TAS50.OVL file with information it needs. ADDTPRO can be run from either DOS or Windows.

To run ADDTPRO from Windows: If you are running Windows 3.1, from Windows' Program Manager, click on File, then Run. If you are running Windows 95, click on the Start button and choose Run from the menu displayed. Enter C:\TAS50\R50 ADDTPRO as the command line (substitute for C:\TAS50 the installation path you specified above if different).

To run ADDTPRO from DOS: Go to the C:\TAS50\> DOS prompt (substitute for C:\TAS50 the installation path you specified above if different). Then type and enter R50 ADDTPRO.

Be sure to reboot your computer now so that the changes made by ADDTPRO and the changes to the CONFIG.SYS and AUTOEXEC.BAT files will be made current on your system.

Uninstalling TAS Professional 5.1

TAS Professional 5.1 can easily be uninstalled if necessary. Just insert RUN DISK 1 into your floppy drive. If you installed through Windows originally, run A:SETUP.EXE as you did before. If you installed through DOS, run A:INSTALL.EXE as before. Be sure to click on Uninstall rather than Install this time, though.

NOTE: The uninstall routine will delete all files in your main TAS50 directory (or wherever you originally installed TAS Professional 5.1), as well as all subdirectories, so be careful when choosing this option..

REGISTRATION

Until your copy of TAS Professional 5.1 is registered, you will be restricted to no more than 100 records per data file (you may have noticed the 'demo mode' registration screen that appeared when you ran ADDTPRO).

NOTE: Your registration information and product unlocking code will be contained in the file START_UP.RUN, located in your main TAS50 directory. You should be sure to keep a copy of your START_UP.RUN file in a safe location in case you ever need to reinstall your copy of TAS Professional 5.1. If you lose your START_UP.RUN file you will have to get a new copy of this file from Business Tools which will delay you in your use of the product.

If you previously purchased TAS Professional 5.0 for DOS and have registered that product, you copy of TAS Professional 5.1 has already been pre-registered for you. When you run TAS Professional 5.1, the initial registration screen should display your company name and address. If it does not, please contact Business Tools. Your TAS Professional 5.1 Disk Packet does not contain a REGISTRATION DISK and you do not need to run this registration procedure.

HOW TO REGISTER YOUR COPY OF TAS PROFESSIONAL 5.1

1. Insert the floppy disk with REGISTER on the label into your A or B drive.

2. From your TAS Pro 5.1 sub-directory type R50 x:REGISTER (where x is the appropriate drive code, A or B).

3. The standard demo screen will be displayed. Press the ENTER key and an entry block will come up for your name, address, etc. At the bottom of the screen is displayed a message about a different screen. Press the enter key to continue.

4. The program should momentarily change the screen and then will return to the entry screen first displayed. In this block enter the licensee's name, company name, if applicable, and address. YOU MUST ENTER AT LEAST A USER NAME OR COMPANY NAME, STREET AND CITY/STATE/ZIP, PROVINCE, ETC. IF YOU DO NOT CAS WILL NOT GIVE YOU AN UNLOCKING CODE.

5. After you have entered all applicable information the program will display the final registration screen as it will appear. You will be asked if you wish to print a copy to fax or for your permanent record. You should answer Y to this question. It will print-out a single page with the information you just entered and other pertinent data we need to register your product. Please fax this report to CAS if possible. If you do not have a fax machine or easy access to one then this is not necessary. However, if we have this information it will make registration much easier.

6. After that the original entry screen will be re displayed again along with a box underneath it that has your serial number and the registration code for the data you entered. If you are in current contact with CAS then we will ask for that information. If you aren't, it was printed on the report you received in step 5. If you are not in contact with CAS then press ESC now. The program will ask if you wish to keep a record of the entries made, answer Y. When you run this program the next time it will tell you there was a registration in process and do you wish to continue with it, you will answer Y to that also.

7. If you are in current contact with CAS we will give you an unlocking code that you will enter in the field that says Unlock Code from CAS. NOTE: It will only work for the exact set of characters you entered the first time and only one time. If you exit the process before completion, and do not save the information first, you will need to contact CAS for another unlocking code, even if you put in the exact same entries!

8 Once the product is unlocked there are no longer any record number restrictions. Please be aware; if you attempt to change anything on the registration screen you will not be able to run TAS Professional 5.1.

IMPORTANT: Please put your REGISTRATION DISK and License Agreement envelope in a secure place. You may need to locate them in the future and this disk, along with the license agreement envelope it was in, is your way of showing that you are legally licensed to use this software package.

CONFIGURATION

Listed below are some additional instructions for configuring your system to run TAS Pro 5.1 or applications you have written using TAS Pro 5.1.

For Workstations running TAS Pro 5.1 applications in DOS mode

Memory Managers. Occasionally DOS memory managers will conflict with TAS Pro 5.1. While earlier versions of TAS Pro did require use of a DOS memory manager, that requirement has been eliminated by TAS Pro 5.1's use of virtual memory while running in DOS mode (see below). To reduce the chance of conflicts:

If you are loading EMM386 in your CONFIG.SYS, load it with the NOEMS parameter set (device=C:\DOS\EMM386.EXE NOEMS).

TAS Pro 5.1 does not work well with QEMM versions 7.0 or higher. We recommend that you do not use QEMM with TAS Pro 5.1.

If you are experiencing unresolved conflicts of some sort, try loading Btrieve into low memory (by default it loads into high memory). To do this add a /E to the end of your Btrieve load string, either in the Utilities-Configuration menu selection or the batch file where you load Btrieve.

TPC50.INI and Virtual Memory. TAS Pro 5.1 in DOS mode uses your hard disk to create 'virtual memory' when RAM memory space is exhausted. This use of virtual memory effectively eliminates the possibility of running out of memory while running a TAS Pro 5.1 application. Virtual memory is not used when running a TAS Pro 5.1 Windows application (TP5WIN.EXE) or if the TAS Pro 5.1 DOS application is being run through Windows or from a Windows DOS prompt. Rather, Windows' own virtual memory management takes precedence.

When TPC50.EXE loads from a DOS prompt, it checks for the presence of a TPC50.INI file. If the .INI file is not present, virtual memory will not be used and you are limited to physical RAM. If ON=TRUE in the TPC50.INI file, then virtual memory will be used (ON=FALSE to turn it off). The SWAPMIN=4M and SWAPMAX=4M settings in the .INI file will result in creation of a 4MB 'swap file' (extension .SWP) on the hard disk when the application is first run. You can change the size of the swap file here if desired but we recommend that the SWAPMIN and SWAPMAX settings always be identical. The SWAPFILENAME setting in the .INI file indicates where on the hard drive to create the swap files. This is a *systemwide* setting, so it must be recognizable by all machines on a network. If all workstations do not have their own C: drive, you will have to specify a network path. You should never specify an actual filename as part of the SWAPFILENAME setting as this will result in messy conflicts over a network. Upon normal termination of your TAS Professional 5.1 application, the swap file should be erased from the hard drive. The swap files are temporary, so you can delete them if some have accumulated on your disk.

Free Extended Memory. In order to load a TAS Pro 5.1 application from a DOS prompt, you'll need about 1.7 to 2.0 MB of free Extended Memory available.

Large Hard Drives. When running in DOS mode, some TAS Pro 5.1 applications have trouble if installed on macines where Logical Block Addressing (LBA) is set ON in the BIOS. LBA is generally used to allow use of hard drives larger than 500MB with motherboards that only support drives up to 500MB. The only workaround if this creates a problem is to turn off LBA or upgrade the motherboard or BIOS.

Configuring a Lantastic Network

These instructions apply to both DOS and Windows TAS Pro 5.1 applications.

On the file server, you should increase the FILES= to 199 from 150 in the CONFIG.SYS file. BUFFERS, FCBS, STACKS don't matter.

You will probably have to make the .EXE files read-only (for Win this is TP5WIN.EXE, for DOS it's TPC50.EXE) so they can be shared over the network.

It is very important that you only run SHARE (either DOS's SHARE.EXE or Lantastic's version of SHARE) ONLY on the file server where your application's data files are installed. You should disable either version of SHARE on ALL other machines that will be using your application. Apparently multiple copies of SHARE and Btrieve data files do not coexist well over a Lantastic network. If other machines that are also running SHARE use your application, there is a chance that data files could become damaged.

On the server, the SHARE settings should be /L:1000 (for locks) and /F:14200 (for files). These settings would be appropriate for a fairly large multiuser application, your optimal settings may vary.

We have not had the best experiences using Lantastic's LANCACHE or LANPUP, particularly on the older Lantastic networks. You may want to consider disabling these and using SMARTDRV instead.

If you experience network printer slowdowns on a Lantastic network when printing from within your application:

1) Run Lantastic's NET_MGR
2) Select Shared Resources
3) Select the printer (@PRINTER or whatever it is called)
4) Set the characters per second to 10000
5) Select System Setup
6) Set it so that printers only do 1 task at a time
7) Make sure that immediate de-spooling is ENABLED

Novell Network Issues

These instructions apply to both DOS and Windows TAS Pro 5.1 applications.

On some installations you may need to have a FILE HANDLES=80 setting in each workstation's SHELL.CFG file. This may be the case if you are experiencing lockups at places where there is heacy demand for file handle resources (many files open).

GENERAL INFORMATION

SYSTEM OVERVIEW

TAS Professional 5.1 is a complete application development environment and consists of several modules or parts. Brief descriptions of the major parts of the development environment are given below; these parts correspond to the Main Menu **CHOOSE** options.

PROGRAM: The main tools for developing, compiling, running, and editing programs are found in this section. Included here are the Screen Painter (used for laying out input screens and menus), the Report Writer (for building reports), the Program Editor, and access to the Program Generator, Compiler, and the Runtime. For a brief overview of how these tools are used in developing applications, see below.

DATABASE: This section contains programs for maintaining the Data Dictionary, the centralized repository of information about the structure of a database. The Data Dictionary holds the specifications for all the fields (data holders) used in your programs and includes field information such as size, type, description, and so on. There are also tools for finding, editing, and deleting records, as well as utilities for simple reporting, and importing/exporting data from/to files that use a variety of formats.

RUN: In this part of the development environment, you can run TAS Professional 5.1 programs, other programs not written in TAS Pro, and DOS commands. This facility allows you to perform a variety of tasks without leaving your TAS Pro session.

UTILITIES: TAS Professional 5.1 provides a number of utility programs to perform certain housekeeping and maintenance chores. For example, there are programs for maintaining your printer drivers; for updating existing data dictionaries with new fields; and setting certain configuration parameters such as the default path for your data dictionary, default printer, and even how you want dates to appear!

All the development environment parts and the programs they contain are covered in detail in **Chapter 3, Main Menu Programs**.

THE APPLICATION DEVELOPMENT PROCESS

Before providing details on TAS Professional 5.1 system specifications, let's take a brief look at the typical application development process. The tutorial in Chapter 2 will walk you through most of these steps and give you an excellent introduction to the power and flexibility of TAS Professional 5.1. After finishing this chapter, you'll want to be sure to run through the tutorial. First, here's an overview of the application development process using TAS Professional 5.1:

1. **Design:** The first and most important step in building any database application with ANY product or tool is to design your database and processing flow before writing a line of code. (If you're laughing, you can skip to the section on system specifications.) Your design will lay out the file structures, fields, and relationships in your database as well as the contents and arrangements of the menus, input screens, and reports that are part of your application.

2. **Create the Database:** Based on your design, you can proceed with defining the database fields in your data dictionary, specifying the names, types, sizes, descriptions, and so on. During this step you would also indicate which fields will serve as keys or indices for sorting and retrieving records.

3. **Populate the Database:** In this stage, you would use **Maintain Database** to put information into the fields that make up your database. Facilities are included for finding, editing, and deleting record data.

4. **Create Screens and Reports:** Use the **Screen Painter** to design menus and input screens that will allow the user to interact with your application. The **Screen Painter** in TAS Professional 5.1 allows you to define the screen size, colors, titles and description lines, boxes, and input fields, including defaults and checks on

data input validity. Use the **Report Writer** to design reports that provide different views into the data in your database. For example, you might want to show all your customers, and for each one, their items purchased and amounts so far this year. You can design headers and total lines and put fields in the exact locations where you'd like them to appear on the report.

5. **Create program code:** TAS Professional 5.1 includes a **Code Generator** that takes the specifications you've laid out in the data dictionary as well as your screen and report formats and generates the actual code that implements the database, screens, and reports in your application. In fact, TAS Professional 5.1 is powerful enough that you can actually develop entire applications without ever writing a line of code! But if you do want to write code, that capability is available to you (see below).

6. **Compile the code and run:** Once you've generated the code for your application, you can compile it using the TAS Professional 5.1 **Compiler** and then run the program using the **Runtime**.

> **NOTE:** Before a program can be run, it must be compiled. All programs use the **Runtime** for execution since they cannot be run by themselves. Using a runtime means the compiled programs are much smaller in size, and take much less time to compile (prepare for execution) and load than an application consisting of full .EXE programs. Also, the completed program uses only the absolute minimum amount of memory required. This allows you to **CHAIN** to multiple programs and have them all active in memory at the same time. Even though only one program can be used at a time, data and files from previous programs (that have been chained from) can be used in those subsequent programs. For more information on how the compiler works please refer to **Chapter 4, Compiler Information**.

7. **Debug the code and recompile. Repeat as necessary.**

One of the unique features of TAS Professional 5.1 is that you can also develop programs without using the Screen Painter or the Report Writer or the Code Generator. Drawing from over 180 commands and almost 200 functions, you can use a text editor to construct programs just as you would with any programming language. (See **Chapter 5, Command Reference** and **Chapter 6, Function Reference** for full details on all the commands and functions TAS Professional 5.1 offers.) If the command or function you need doesn't exist, you can even write it yourself! (See the **CMD** command for information on User Defined Commands, and the **FUNC** command for information on User Defined Functions.)

Many developers use the Screen Painter, Report Writer, etc. to produce the first version of the source code and then make edits and additions line by line using a text editor. This way of working can reduce dramatically the amount of time it takes to develop an application. You can even switch back and forth. All the changes you make with a text editor to the program can still be used in the program editor.

Before working through the tutorial in Chapter 2, please take a few moments to familiarize yourself with the system specifications and general information presented below.

SYSTEM SPECIFICATIONS

NUMBERS

Maximums (all sizes in bytes/characters unless otherwise given):

Single field size (string, A type):	Available memory (Windows 64k)
Number of array elements in any one array: (all arrays are single dimension)	Available memory (Windows 64k)
Record size:	65,535
Number of keys per record:	24
File size:	4,278,190,080
Number of fields per record: (an array field counts as one field)	255
Number of files opened per program:	100
Number of fields per program: (an array field counts as one field)	2,000
Size of **DEFINE**d data per program:	Available memory (Windows 256k)
Size of single source code module:	Available memory
Size of compiled constant section:	Available memory
Size of compiled code section:	Available memory
Length of a single source code line:	1024
Number of nested structures (each type separately):	20
Maximum number of nested **CHAIN**ed programs:	Limited by Memory

Minimum memory required:Four megabytes (Two mb available memory minimum)
(includes memory required for Btrieve^{tm})

NOTE: Btrieve can be loaded into expanded memory and will require only 55k (approximately) in 'lower' memory. This happens automatically if you have an expanded memory manager in operation. This is only a concern in DOS. The process is completely different in Windows. For more information please contact CAS customer support.

MATHEMATICAL OPERATORS

The symbolic math operators allowed in order of precedence are:

^	Exponentiation
*	Multiplication
/	Division
+	Addition
-	Subtraction

You can alter the order of evaluation by nesting expressions in parentheses.

EXAMPLES ('?' is shorthand for **PMSG** - the print message command)

```
? 3 ^ 2
9
? 3 * 2
6
? 4 / 2
2
? 3 + 2
5
? 3 - 2
1
? 3 + 2 * 1 + 4
9
? (3 + 2) * (1 + 4)
25
```

Mathematical operators for strings (A type fields):

+ Concatenate two strings together.
* Will trim the trailing spaces of the first field before concatenating the strings together.

EXAMPLES

```
? 'ABCD    ' + 'EFG'
ABCD    EFG
? 'ABCD    ' * 'EFG'
ABCDEFG
```

BOOLEAN (LOGICAL) OPERATORS

These are used in logical expressions and can only affect the logical values, .T. or .F. All Boolean values start and end with a period. There must always be a space before and after the operator in any expression. Here are the Boolean operators in order of precedence:

.N. - not
.A. - and
.O. - or

The .N. operator basically inverts the truth value. For example, .N. .T. says not true, which is equivalent to false (or .F.). The .A. operator only returns true if **both** arguments are true (.T.). Thus .T. .A. .T. would return .T. The .O. operator returns true if **either** argument is true. More examples are given below. Please note that you can alter the order of evaluation using parentheses; nest the operators in parentheses if you want to be completely certain.

EXAMPLES

```
? .F. .O. .F.          ? .F. .A. .F.          ? .N. .F.
.F.                    .F.                    .T.
? .F. .O. .T.          ? .F. .A. .T.          ? .N. .T.
.T.                    .F.                    .F.
? .T. .O. .F.          ? .T. .A. .F.
.T.                    .F.
? .T. .O. .T.          ? .T. .A. .T.
.T.                    .T.
```

COMPARISON OPERATORS

These are used for comparing one value to another. The result from any comparison will be .T. if the comparison is correct, or .F. if it isn't. As with math operators and logical operators, you can alter the order of evaluation using parentheses.

The operators are:

=	equal to (don't confuse with the equals value command)
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
<>	not equal to
$	contained within (only works with strings - A type fields)

EXAMPLES

```
? 1 = 2
.F.
? 1 < 2
.T.
? 1 <= 2
.T.
? 1 >= 2
.F.
? 1 <> 2
.T.

? 'ABC' = 'DEF'
.F.
? 'ABC' < 'DEF'
.T.
? 'ABC' <= 'DEF'
.T.
? 'ABC' >= 'DEF'
.F.
? 'ABC' <> 'DEF'
.T.
? 'ABC' $ 'DEF'
.F.
? 'ABC' $ 'XXXABCXXX'
.T.
```

NOTE: You need to be careful in setting up your comparisons to make sure you will get the results desired. For example, the expression given below combines comparison operators and a logical operator and looks perfectly harmless until you realize that the expression ALWAYS evaluates to .T. (true).

```
(x <> 3) .O. (x <> 4)
```

If you used this expression as part of an **IF**/**ENDIF** structure, the commands within the structure would ALWAYS be executed.

FIELD NAMES

A field name may consist of up to 15 characters. The first character must be alpha (A-Z). You cannot include spaces, or other math or comparison operators within the name. All other characters are acceptable, including numbers as long as they aren't the first character. The compiler sees no difference between upper and lower case characters. This applies to fields within files/records or those created within the program using the **DEFINE** command.

EXAMPLES

The following are legal field names:

abc
Customer_code
INVOICE.NUM

The following are not legal:

2abc	first character must be A-Z
+customer_code	first character must be A-Z and can't use math operator
invoice-num	can't use math operator.

FIELD TYPES

TAS Professional 5.1 has a very rich group of field types, more than any other 4GL today. These types give you much flexibility and power. Since you can call external programs and pass values to them, some of these types, especially the p-pointer (type P), are even more important. (All sizes are in bytes/characters.)

Type	Name	Int Size	Disp sizes	Dec	Acceptable characters
A	Alpha	1-4gb	1-4gb	n/a	All displayable chrs
	(Windows)	1-64k	1-64k		
N	Numeric	8	1-20	1-8	0-9 and . (decimal)
B	Byte	1	1-3	n/a	0-9
L	Logical	1	3	n/a	.T., .Y., .F., .N.
I	Integer	2	1-5	n/a	0-9
R	Record	4	1-10	n/a	0-9
D	Date	4	5,7,8,10	n/a	0-9
T	Time	4	5,7,8, 10,11,13	n/a	0-9
F	F-Pointer	5	7	n/a	no entry allowed
P	P-Pointer	14	26	n/a	no entry allowed
O	3.0 BCD	1-10	1-20	2,4,6,8	Old form Binary Coded Decimal (BCD)
V	Overlay	1-4gb	n/a	n/a	Treated like an A type field. It is recommended that this type field be used only for combining fields as part of a key in a regular TAS Pro 5.1 file.

NOTE: Even though the maximum size of an A or V type field is 4gb (4,294,967,295 bytes) the maximum size of any record in a Btrieve file is 64k (65,535 bytes). Also, the largest size of an A or V type field in Windows is 64k.

INTERNAL SPECIFICATIONS

The alphanumeric type (type A) is straightforward: for each character the program maintains a byte of storage. There are no extra bytes at the beginning or end. The size of the field is determined by the program from a **DEFINE** command or from the data dictionary.

A numeric field (type N) is in the standard IEEE floating point format and is contained in 8 bytes of storage independent of the display size of the field.

A byte field (type B) can have a value from 0 through 255.

A logical field (type L) can have only two values, .T. (true) or .F. (false). Internally .T. is 1 and .F. is 0 and uses 1 byte of memory.

An integer field (type I) can have a value from 0 through 65535. It is maintained in standard Intel low byte low format and uses 2 bytes of memory.

A record field (type R) can have a value from 0 through 4,294,967,295. It is maintained in standard Intel format and uses 4 bytes of memory. This field can also be thought of as being a long or double integer and can be used in that manner.

A date field (type D) is kept in a special format that corresponds to that used by Btrieve. This format is low byte day value, next byte month value and high word year value, total 4 bytes memory used. All values are kept in binary format.

A time field (type T) is kept in a special format that corresponds to that used by Btrieve. This format is low byte hundredths of seconds, next byte seconds, next byte minutes, and high byte hours, total 4 bytes memory used. All values are kept in binary format.

An F-Pointer field (type F) is a compound field. The low byte contains the type of the field being pointed to; F - standard field, C - constant, or X - expression. The high double-word is the location of the field specification within the field list segment within the program being run, for a total of 5 bytes memory used. This field type is not usable outside of TAS Professional 5.1. However, it has great use within a program and is the smallest pointer value.

A P-Pointer field (type P) is a compound field. This field is closer to the standard far pointers in C. The low double-word is the location value, the next byte is the field type (a single alpha character as listed above), next byte is the number of decimal characters, the next double-word is the internal size in bytes, and the final double-word is the display size in characters, for a total of 14 bytes memory used. The reason for the complexity of this field is so that you can pass this value to an external program (non-TAS) and know everything necessary about the field. The location value points to the actual field value in memory, not at the field list. You can add and subtract values to/from a P-Pointer field location using integers. However, all other parts of the field cannot be changed. NOTE: In Windows the location value is in standard SEG:LOC format.

An old form BCD field (type O) uses a BCD format to provide a more efficient use of memory, occupying only a little over half the space occupied by the standard ASCII format. The actual size is known as the calculated size (CSIZE or INTSIZE) and is calculated as (Field Size/2) + remainder. Please note that only fields with an odd number of digits will hold negative values. If the field has an even number of digits, the negative will be ignored, and the absolute value taken. This 'old form' BCD type was used in TAS Professional 3.0. NOTE: In TAS Professional 5.1 programs the defined 3.0 BCD fields are converted, automatically, to an appropriate new field type. This, however, does not apply to fields in files. The conversion program from 3.0 to 5.1 can automatically convert your 3.0 BCD fields that are part of your files to the appropriate new form when you convert your data dictionary. You should take this opportunity to do so. Through the use of the DDF files (that you can create in TAS Pro 5.1) you can access your Btrieve data with many other programs, including Access, Crystal Reports, etc. If you continue to use the 3.0 BCD fields those programs will not know what type of field

you're using. However, the TAS Pro 5.1 programs running in DOS will still handle those fields even if you don't convert. The Windows programs will not.

CONSTANTS

String or alphanumeric constants (type A) are surrounded by quote marks; it doesn't matter whether you use single (‘’) or double (“”) quotes as long as the same type is used at the beginning and end. For example:

‘ABC’	
‘123’	
‘01/01/90’	
“should’ve done it”	When using a single quote as an apostrophe surround the constant in double quotes.
‘abc123”	This is not legal since it starts with a single quote and ends with a double.

Numeric (type N) constants must have a decimal chr (period or comma, as applicable) followed by at least 1 numeric character. For example:

100.0	
234.5678	
0.25	
1,950.25	This is not legal since commas cannot be included (unless it takes the place of the decimal character.)

Byte (type B) constants are followed by a B or b. No decimal characters are allowed. For example:

10b	
1b	
99,999B	This is not legal since commas are not allowed and the maximum Byte value is 255.

Integer (type I) constants can be followed by a ! for clarity. These constants have a numeric value less than 65,536 and no decimal characters or commas are allowed. For example:

10	
1!	
99,999!	This is not legal since commas are not allowed and the maximum Integer value is 65535.

Record (type R) constants are followed by an R. No decimal characters or commas are allowed. For example:

100r	
192543R	
192,543r	This is not legal since commas are not allowed.

Logical (type L) constants are T or F surrounded by periods, i.e., .T. and .F. You can also use .Y. for .T. and .N. for .F. These are the only options available for logical constants. You must leave at least one space before and after the constant.

There are no special Date (type D) or Time (type T) constants. You can set a date or time field equal to an alpha field directly.

time_field = ‘10:10:00’	Can use seconds or not as desired.

time_field = '10:10'

date_field = '10/10/90'	Can use any type of separator between dates and
date_field = '10 10 90'	can use the full year or just 2 digits.
date_field = '10 10 1990'	Don't forget, month, day, and year must be in the proper order as you have them specified in the setup.

There are no P-Pointer (type P) or F-Pointer (type F) constants.

ARRAY SPECIFICATIONS

Any field may be defined as being an array. This means that each field is actually a multiple field. Each array element is of the same type and size, and may be accessed using the same field name and just changing the element number. The array element specifier is a numeric field, constant, or expression surrounded by squared brackets [] and immediately following (without any space separating) the field name. TAS Professional 5.1 allows only single dimensioned arrays; however, since the element number may be an expression you can create one that will treat the array as though it was multi-dimensioned. For example, see the AE UDF in the TASRTNS library. The maximum number of elements in any one array is 4.3 billion (4g); the first element in any array is 1. NOTE: In Windows programs the maximum number of elements in any one array is 64k.

EXAMPLE

SALES[12]

SALES[MONTH[CNTR]]

COST_OF_GOODS[month(date())]
(The function month(date()) will return the month number from the current system date.)

ARRAY[(cntr2-1)*24+cntr1]
(The example above shows how a multi-dimensioned array might be handled. The example assumes that cntr1 is the first dimension value and cntr2 is the second dimension counter and that there are 24 elements in each first dimension. The limit of the second dimension is related to the total number of elements in the array only.)

EXPRESSIONS

An expression might almost be considered a compound value. All of the parts of the expression are resolved (each math operator or function is executed) into a single value. Internally each expression is broken down into a series of expressions each involving only two elements. For example:

1+2+3

In the above case the program calculates 1+2 first and then the result (or 3) +3 to get the answer 6. However, in the case of:

1+2*3

The program calculates 2*3 first (since multiplication has a higher precedence than addition) and then adds that to 1 for a total of 7. If you have any doubts as to the order of calculation you should surround

values that you want to be evaluated in a certain order with parentheses. In the above example, if you wanted 1 to be added to 2 first, and then multiplied by 3 your expression would look like:

```
(1+2)*3
```

The result would then be 9, or 1+2 to get 3 and then 3*3 to get 9.

The different elements (field or value) of the expression need not be of the same type. However, the program will automatically convert the second part of the expression to the same type as the first. For example:

```
"Today's date is: "+date()
```

The result of the expression above would be the current date (derived from the date() function) converted to an alpha field since the first part of the expression is an alpha field and the date value was automatically converted. Again, if you have any doubts over whether or not a value is being converted properly you can also use the full range of conversion functions provided in TAS Pro 5.1.

An expression might be very simple. For example:

```
1+1
```

Or it might get very complex:

```
size_hldr + iif(dict_array_ele <> 0,dict_array_ele*int_size,int_size)
```

Expressions may resolve to a value or may be a logical expression and use comparison operators also. These type of expressions resolve to .T. or .F. For example:

```
1 = 1
```

This resolves to .T. Another example might be:

```
x = 100 .o. (y = 50! .a. z = 'ABC')
```

And they might be made up of both:

```
x = (size*num_array_ele) .o. field_name=('TEST'+date())
```

The only limit to the length of an expression is the maximum 1024 character limit per line. Any legal function, including user defined functions (UDF), may be a part of an expression.

NOTE: Do not put parentheses around single fields or values in an expression. For example:

```
x=(y)*3 is illegal
```

```
x=(y*3) is fine, but unnecessary.
```

NOTE: If the **#PRO3** or **#OLD_MATH** compiler directive is set in a program then all math operations are evaluated on a left to right basis regardless of precedence

INDIRECT FIELD REFERENCES

A very powerful feature of TAS Professional 5.1 not found in other 4GLs is the ability to refer to a field indirectly through the use of pointers. A pointer is a value that literally 'points to' the field. In TAS Pro 5.1 there are two different types of pointers. One points to the field specification within the field list in a program; the other points to the actual location of the field value. This second pointer is more typical of that used in languages such as C. In fact, we have set up this type of pointer ('P' pointer) so that you can pass it to non-TAS programs and access your TAS data directly!

To set the value of a pointer (either F or P) use the pointer command:

```
pointer_fld -> field_name
```

The command automatically determines which type of pointer you are using and returns the correct value. To refer to a field through the use of the pointer precede the pointer name with the redirector symbol ('&'). In all cases where you need a field name in a command you can use this method of referring to it. For example, your code may look like:

```
fptr_fld -> test
enter &fptr_fld
```

This is the same as:

```
enter test
```

Through the use of this facility in TAS Pro 5.1 you have the power to access data without ever knowing what the data is you're going to work with when you write the program. The Maintain Database and Export/Import programs use this capability.

You can set a pointer to an element in an array by specifying the array field and the element, i.e.:

```
pptr_fld -> test[cntr]  (where cntr is the array element number)
```

The array element number can be either a constant or a variable. When pointing to an array field the pointer value must be of P type.

The pointer field can also be an array, i.e:

```
fptr_fld[x] -> test[cntr]
enter &fptr_fld[x]
```

In this example the xth element of the P type pointer array pptr_fld now points to the element in test as determined by the value of cntr. The field names used in this example are arbitrary. You don't have to name P type pointer pptr or F type pointer fptr; however, in the documentation we refer to them in that manner. The array specifier [x] in this case means the xth element in fptr_fld.

You can also refer to an array element in a field being pointed to through the use of the array pointer redirector. For example:

```
fptr_fld -> test
enter @fptr_fld[1]
enter @fptr_fld[2]
...
enter @fptr_fld[10]
```

where test is a 10 element array. TAS Professional 5.1 uses the array element specifier to determine the correct element in the array field being pointed to and not the pointer field. In this situation you may use either the F or P type pointer. In the example above, each **enter** will apply to the appropriate element in the field **test**. You need not use a constant as an array specifier; it is used here only as an example.

LINE LABELS

Certain commands allow transfer of control to specific line labels. These include **GOTO**, **GOSUB** and others. A legal line label may be any set of characters except the math and comparison operators. Only the first 14 characters of any label are used and the name must be terminated with a colon (:). It should also be the only thing on the line other than a remark.

EXAMPLES

```
START:
fini:
enter_customer:
```

LOADING BTRIEVE[tm]

TAS Professional 5.1 uses Btrieve from Btrieve Technologies as its file manager. This allows you to access TAS Pro[tm] files in other programs that use the Btrieve file manager. And, TAS Professional 5.1 can be used to access data from other programs, such as DAC Easy[tm], Great Plains[tm] software, etc.

To be able to use Btrieve it must be loaded first. TAS Professional 5.1 does this automatically. Both the runtime and the compiler check to see if it is loaded first before loading it from the disk. It should be in the same path as the runtime and compiler programs (in Windows the DLL is installed in the \WINDOWS subdirectory). Also, when the runtime or compiler programs exit Btrieve is removed from memory so that it doesn't take up any extra space during the operation of other programs. If Btrieve was loaded previously to running TAS Professional 5.1 it will remain loaded when you exit from TAS Pro.

Special loading instructions are provided to Btrieve from the TAS50.OVL file. This is maintained using the TASCINFO.RUN program (Setup) and is covered in the Getting Started section of this manual.

In Windows the Btrieve file loaded is WBTRTAS5.DLL. This is loaded automatically for you the first time you load the Windows program. The Btrieve parameters are kept in the file WIN.INI which is in your \WINDOWS subdirectory. For more information please refer to **Chapter 11 - Windows Programming**.

DATA DICTIONARY

TAS Professional 5.1 uses a Data Dictionary that keeps track of every field in all of the TAS files, where the files are located, and the keys for each file. The program TASDMGR.RUN maintains the dictionary files.

Four files make up the dictionary. They are: FILEDICT.B - The actual field list for each file, also known as the File Descriptor (FD); FILELOC.B - The location and File Descriptor name for each file; FILEKEY.B - The key specifications; and FILEKNUM.B - The internal key numbers used during compilation. You may create as many different sets of dictionaries as desired. Each set must be located in the same path. New sets of dictionaries are created using the TASDMGR.RUN program.

The records in FILELOC.B can tell the program where the file is and what the file descriptor is when the file is opened. Multiple files may use the same File Descriptor so there may be more files in this list than groups of fields in FILEDICT.B. However, you can open a file that does not have an entry in this list as long as you specify the PATH and FD option when opening the file. (Please see the documentation on the **OPENV** (Open Variable) command.)

There is also a Data Dictionary for defined fields, i.e., those used in a program but not part of a file. The file for these fields is FILEDFLD.B.

The file FDHLDR.B is used to temporarily store a FD while it is being changed.

LOCATION OF TAS[tm]

If the programmer uses the DOS SET command and the location of TAS Professional 5.1 is in the PATH command then TAS programs can be executed from any drive or path. This will allow you to maintain one set of executable programs while keeping different development projects in their own paths.

At the DOS prompt (or during the initial computer startup period when AUTOEXEC.BAT is executed) you should execute the following command.

```
SET TAS50=path
```

where path is the location of TPC50.EXE, the default data dictionary set, the printer control files, the TAS*.RUN programs, and ERRMSG.B. For example:

```
SET TAS50=C:\TAS
```

This assumes that the files above are in a path called TAS that is located on drive C.

The path value should also be added to your PATH= command in AUTOEXEC.BAT. The installation process will do both of these automatically, if you desire.

For more information on using multiple directories, please refer to the section at the end of this chapter.

EXECUTING TPC50

Included in the distribution set are two .BAT files, R50.BAT and C50.BAT. You can use these as shorthand when executing TAS from the DOS prompt. They both execute TPC50.EXE, however, R50.BAT is for running a program that has already been compiled. C50.BAT is for compiling a program. You or the user can include command line extensions just as though you had executed the program directly.

NOTE: The program START_UP.RUN **must** be in the TAS50 sub-directory to run TAS Professional 5.1.

SPECIAL FILES: TAS50.OVL & TASCOLOR.OVL

These two files are used by TAS Professional 5.1 to keep track of certain system information (TAS50.OVL) and colors (TASCOLOR.OVL). You can change the values in TAS50.OVL through the use of the program TASCINFO.RUN (**Main menu, Utilities - Set Configuration**). You can change values in TASCOLOR.OVL through the use of the program SETCOLOR.RUN.

When you run TPC50.EXE it looks for both of these files in the user's current path. If they exist there they are used. If they don't exist there the program uses the files in the TAS50 path. This will only work if the SET TAS50= has been properly executed and the TAS50 path is stored in the computer's environment space.

REFERENCES TO PROGRAM EDITOR

The TAS Professional 5.1 Program Editor uses a series of menus to display all the commands possible and allow you to choose the proper one. Throughout this manual we use a way of referring to this as:

CHOOSE_MENU_OPTION -> 2nd_MENU_OPTION -> 3rd_MENU_OPTION -> 4th MENU_OPTION (if necessary).

An example of this is below. The illustration shows the menu options to choose the **DEFINE** field command. If we were referring to that command in this documentation it would be:

FIELD -> CREATE/CHG -> DEFINE

TAS PROFESSIONAL 5.1 SOURCE CODE

TAS Professional 5.1 uses an 'English-Like' type of source code. For those of you who have been using any of the xBASE variants, this will be very familiar to you. This is a very flexible type of source code since the options can be in any order and if you don't use an option you just don't include it. For example:

```
? 'THIS IS A TEST'  AT 1,1 ABS
```

Note the use of the optional '?' instead of PMSG. The following is treated identically by the compiler:

```
? 'THIS IS A TEST'  ABS  AT 1,1
```

This code may be edited directly using a standard text editor, or using the Program Editor included with TAS Professional 5.1.

MOVEMENT ON THE TAS PROFESSIONAL 5.1 SCREEN

In general the Up and Down arrow keys will move you across and down the screen. The path the cursor moves is dependent on the order of commands within a program. As the user presses the Down arrow key the next line or lines in the program are executed until the next **ENTER** statement is reached. At that point the program checks for a **pre_enter_expression** for that command. If one exists it will be evaluated. If it returns .T. the **ENTER** occurs; if .F. is returned, the program continues on looking for the next **ENTER**, executing all statements along the way. Depending on what the programmer desires it may never reach another **ENTER** or it may come back to one previous.

As the user presses the Up arrow key a similar process occurs, except in this case no commands are executed except the **ENTER**. Again, the **pre_enter_expression** will be evaluated to make sure the programmer wants this field to be entered at this time. If the programmer places an **UPAR** (Up Arrow) command in the program and it is encountered during an upward progression it will temporarily halt progress. If an expression is part of the command it will be evaluated. If it returns .T. it will stop all further upward process. If there is a **goto** label as part of the command the program will transfer control to that line. If there isn't a **goto** label then control will be passed to the line immediately after the **UPAR** command. If there is no expression as part of the command the program treats it as though it returned .T.. If there is an expression and it returns .F. the program will continue processing with the line immediately above the command.

NOTE: If you have a mouse attached to your computer you can use this also in navigating the TAS Professional 5.1 screen. For more information about this please see **Chapter 3, Main Menu Programs, Mouse Control**.

Through the use of these options and commands you can absolutely control what is happening on the screen and where you will allow your user to go. This is how you make 'bulletproof' programs. This process gives you the control that you just can't get with xBASE programs using the standard @SAY...GET and READ commands.

There are other default key stroke actions that are very important to the operation of a TAS Professional 5.1 program. In general pressing the F1 key at an entry will display a help message appropriate to that field. If a file is open and the search keys are not trapped the F5 key will find the first record in the file dependent on the **SEARCH FILE** value or, if the current entry field is a key, whatever that key value is. F6 finds the last record, F7 previous, F8 next, and F9 will find a generic record based on data you have entered into the field. If you are searching through a file using the F5 - F9 keys and reach the beginning or end of the file, an appropriate message will be displayed on the screen. (These messages are found in the file called ERRMSG.B, and are numbered 268 - 271.)

Pressing F4 will delete the record, if it is active, and F10 will save it. If you just want to clear the record from the screen press F3. Very often, but not always, you will find that by pressing the F2 key there will be a look-up window for those records. In general the information concerning the function keys is displayed along the bottom line of the screen.

CREATING A NEW SUBDIRECTORY FOR DEVELOPMENT PURPOSES

Each time you start a new project you should create a new subdirectory for it. This will allow you to keep all the elements of that project localized, including its own data dictionary. We have provided a special program that will create this subdirectory for you and will copy all the appropriate files into it. This program must be run from the DOS version of TAS Professional 5.1.

1. Choose the RUN submenu from the main menu. Choose the Run option and enter the program name: TASDEVSD.

2. The program will ask you for the new subdirectory name. BE SURE TO ENTER A CORRECT NAME SINCE ALL FILES IN THAT SUBDIRECTORY WILL BE DELETED!

3. The program will copy the following files: TAS50.OVL, TASCOLOR.OVL, FILELOC.B, FILEDICT.B, FILEKEY.B, FILEKNUM.B, FILEDES.B, FILEDFLD.B, FILEREL.B, FILECHSP.B, FDHLDR.B and CMPDFLT.B. It will also put the new subdirectory name in TAS50.OVL as the default data dictionary location.

4. If you have your TAS Professional 5.1 subdirectory in your path and have the SET TAS50= properly setup you should now be able to change subdirectories to the new one and run TPC50 from the new location. The main menu should appear and your current location and default dictionary path should both have the new subdirectory name.

5. You can now install the Windows runtime into this same subdirectory if desired. You can use the WINDOWS RUNTIME DISK to accomplish this. Just install it following the instructions from the **Installation** section earlier in this chapter. The target directory should be the local application subdirectory. Doing this will put TP5WIN.EXE and TP5WIN.INI into the application subdirectory, create an icon for the application in your TAS Professional 5.1 Group window, and modify the WIN.INI file if necessary.

6. You are ready to develop a new application in this new subdirectory.

SETTING UP MULTIPLE DIRECTORIES FOR TAS PROFESSIONAL AND ADVANCED ACCOUNTING

The following will explain how to set up Advanced Accounting (Run & Source) and TAS Professional in separate sub-directories. The examples assume installation on drive C and using default directory names. This isn't required and you may change these values at will.

1. Install TAS Professional. The default directory is C:\TAS50. During installation the program will ask if you wish to update your AUTOEXEC.BAT file (append the TAS50 sub-directory to the PATH statement, answer yes - **Y**).

2. After installation type TPC50 ADDTPRO at the DOS prompt in the TAS50 sub-directory. The program will ask if you want to put a SET statement in your AUTOEXEC.BAT file; answer **Y**. It will then ask the location of your AUTOEXEC.BAT file; default is C. If you have done this before the program will warn you that a statement already exists and do you wish to change it; answer **Y**. TAS Professional is now set up properly. Reboot your computer so that the changes take effect.

3. Install Advanced Accounting (both run and source if applicable). The default directory for Advanced Accounting run is C:\ADV50; the source will default to C:\ADV50\ADV50SRC.

4. After you have installed Advanced Accounting in the ADV50 directory type TPC50 ADDTPRO at the ADV50 prompt. This version runs a little differently than the TAS Pro one. The first question it will ask you is the location of your AUTOEXEC.BAT file; again this defaults to C. The next question is whether you have already installed TAS Pro; enter **Y**. The program will display a message and when you press the ENTER key will return you to the DOS prompt.

5. If you have installed Advanced Accounting 5.1 source code, please do the following. While you're in the ADV50 sub-directory type the following lines at the DOS prompt. Each line will copy one or more files.

```
COPY ADV50SRC\*.OBJ
COPY ADV50SRC\*.LIB
COPY ADV50SRC\CMPDFLT.B
```

6. For a final check to make sure that everything is set up properly do the following:

   a. Make sure you're on the C drive - Type C: then press the ENTER key.
   b. Log into the TAS50 sub-directory - Type CD \TAS50 and then press the ENTER key.
   c. Run the TAS menu - Type R50 and press the ENTER key.
   d. At the bottom of the TAS menu you will see 3 paths. The first is Your Current Path, the next is TAS Pro 5.1 Path, and the last is the Default Dictionary Path. All three should have the same value: C:\TAS50.
   e. Exit to DOS, log into ADV50 - Type CD \ADV50 and then press the ENTER key.
   f. Run the TAS menu - Type R50 and press the ENTER key. The three paths should be:

```
Current - C:\ADV50\
TAS Pro - C:\TAS50\
Default Dict - C:\ADV50\
```

   g. If you get the same results as the above (with appropriate changes for drive and path names) then your system is successfully set up.

COMPILING ADVANCED ACCOUNTING 5.1x PROGRAMS

You must compile your Advanced Accounting 5.1 programs using the program provide in the Advanced Accounting Main Menu.

1. From the Advanced Accounting 5.1 Main Menu Choose menu enter **U** for **Utilities**.

2. Choose **J** for **Compile Adv 5.1 Prgs**. NOTE: If the option is not available on the menu then the SET TAS50= value has not be setup properly.

3. You will be able to enter the location of the source files and where you want the run files to go. For example, using the paths setup above you would enter ADV50SRC for source path (the program will automatically append the \). Press the ENTER key and the program will default to the same path for the Run programs. If you want to put those programs in the ADV50 sub directory then just clear the path value. NOTE: A copy of the .RUN file and the .ERR (error) file will still be put in the ADV50SRC path. If you get errors when compiling the program look for the error file in the ADV50SRC path (or wherever you specified the source location).

4. Next you can enter a complete file name or use wild cards. For example, if you wanted to compile all A/R programs you would enter BKAR* (no extension, just the main file name).

5. The program will also allow you to check for programs where the source code is newer than the .RUN program. This will allow you to recompile all programs that have been changed since their last compile.

6. Finally, the program will ask whether to use the OBJ file. If you are compiling Advanced Accounting 5.1 programs always answer Y here. If you don't you will get multiple errors during compilation.

You may also use this program to compile programs you write. Just answer N to the Use OBJ file question.

UPDATING RELATED FILES

If you should make a change to either the LISTGEN.LIB (or other library) files you must make sure you have a copy of that revised file in the ADV50 sub-directory.

NOTE: Through the use of the SET TASLIB= DOS command you can put all your libraries in one location on the disk. Then you only have to worry about updating one set of libraries instead of several. For more information about this please refer to **Chapter 4, Compiler Information**.

If you need to change values in the CMPDFLT.B file (using the Program Comp Info program) the system will automatically make sure that the proper file is updated. This is always the one located with the data dictionary.

INTENTIONALLY BLANK

TAS
PROFESSIONAL
VERSION 5.1

Chapter 2

Tutorial

INTENTIONALLY BLANK

INTRODUCTION

TAS Professional 5.1

TAS Professional 5.1 is designed to be a complete application development environment. It provides application building tools for the programmer new to applications development as well as the seasoned developer demanding professional quality development tools designed to work together from the beginning. These powerful utilities include:

Data Dictionary Manager
Screen Painter
Program Generator
Source Code Editor
Report Writer
Runtime Compiler

The TAS Professional 5.1 programming language is a powerful structured 4th Generation Language. It includes over 150 commands, each with many options, and almost 160 functions that can be used in expressions. Many of the TAS Professional 5.1 commands can be compared to macro functions found in other high level languages. In other words, a single TAS Professional 5.1 command can often replace several entire lines of Basic code. Add to this the ability to create User Defined Commands (UDCs) and User Defined Functions (UDFs), and you now have a very powerful development tool. You can also directly access non-TAS programs and assembly language routines, thus greatly expanding what can be accomplished within a TAS Professional 5.1 program.

One of the reasons TAS Professional 5.1 is so flexible and powerful is that it is built around a Data Dictionary. The Data Dictionary is a special TAS Professional 5.1 data file that is used to keep track of every specification of every field in every record and file you create.

This means that once you define a field you never have to worry about defining it again. Each time you want to use that field in a program, TAS Professional 5.1 automatically looks up the specifications in the Data Dictionary. Storing the specifications in a Data Dictionary greatly increases the ease and speed with which you can create multi-file applications. By using similar fields with the same characteristics in different files, you can retrieve related records, such as finding all invoice records based on a given customer code.

TAS Professional 5.1 uses Btrieve (Copyright Novell, Inc.) to perform all data file I/O and record and file locking in a multiuser system. A significant benefit of using Btrieve is speed. With TAS Professional 5.1 you can typically find any record in a data file in one second or less. With smaller databases, finding a record can be almost instantaneous. Saving or updating records with Btrieve is equally fast. In addition, the speed is consistent regardless of the file size. It takes virtually the same amount of time to save a record in a file with 100,000 records as it does in a file with 1000 records.

TAS Professional 5.1 allows up to 24 different sorts to be maintained on-line for each datafile. With TAS Professional 5.1, you never have to re-sort a datafile just to find a record

As you can see, TAS Professional 5.1 offers features and power beyond those of development systems costing thousands more!

GETTING READY TO RUN THE TUTORIAL

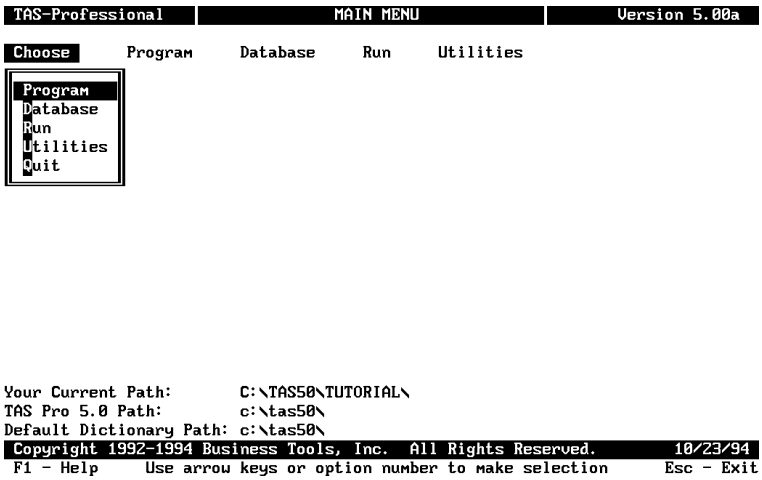
Since this is your first time using TAS Professional 5.0, let's start at the very beginning.

A. Make sure you are in the correct directory where TAS Professional 5.0 was installed. See **Chapter 1, Installation and General Information** for more information on installing TAS Professional 5.0. We will assume this is the \TAS50 directory.

B. At the DOS prompt type the following (You are going to create this subdirectory while logged in the TAS50 subdirectory):

```
MD TUTORIAL
CD TUTORIAL
COPY \TAS50\F*.B
COPY \TAS50\TAS*.OVL
COPY \TAS50\DOPRT2.SRC
COPY \TAS50\SAMPLE\SORPT2.SRP
```

C. Type R50 at the DOS prompt to 'run' TAS Professional 5.0. The Main Menu should be displayed.



D. Press the U key to choose the Utilities sub-menu.

E. Press the S key for Set Configuration.

F. Press the ENTER key only until you reach the entry field with the title Default Dictionary Path (if you are using a mouse you can click on that field directly). If the field was blank before you got there the path will be filled in automatically. If it already had an entry then press the F2 key and the following (or something similar) should appear:

```
C:\TAS50\TUTORIAL
```

G. The program will automatically put the final backslash at the end of the name. The above assumes that your files are on disk C. If not (for example, D, E, etc.) substitute the correct disk drive letter for the C above.

H. Press the F10 key. Enter Y to save the information just entered and press the ESC key 3 times to return to DOS.

I. Type R50 at the DOS prompt and press the ENTER key. The Main Menu should be redisplayed. If you look at the bottom of the screen the values for Your Current Path and Default Dictionary Path should be the same. If they aren't go back to step A and repeat this process.

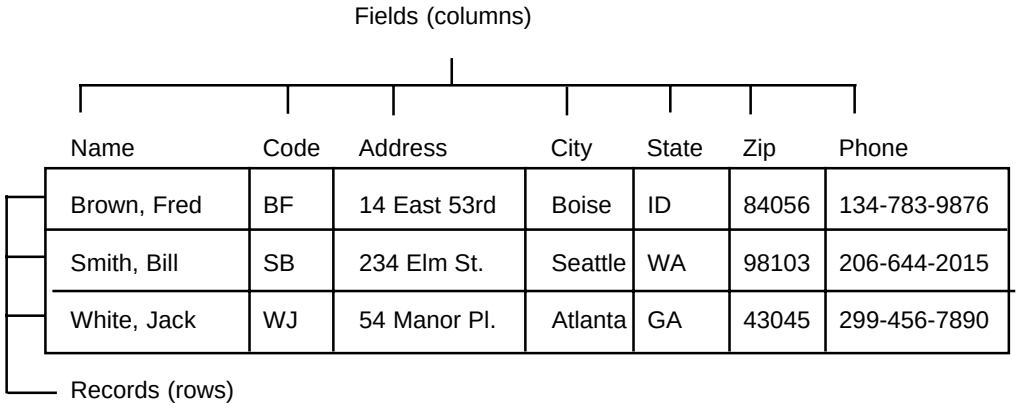
You are now ready to begin the tutorial. Don't forget, if you exit this process that you need to log back into this sub-directory before you can begin again. All the other steps need only be run once.

A BRIEF INTRODUCTION TO DATABASES

Before we begin the tutorial, let's spend just a moment talking about the structure of files in a database and the tools we might use to manage the information contained in those files.

Database Fields, Records, and Files

One way to visualize a database is to think of it as a table of rows and columns. In fact, this is the classic manner in which databases are described in computer literature. In the example shown below, each horizontal row of the table is considered a record and contains all the data about that person. Each vertical column is considered a field.



Name	Code	Address	City	State	Zip	Phone
Brown, Fred	BF	14 East 53rd	Boise	ID	84056	134-783-9876
Smith, Bill	SB	234 Elm St.	Seattle	WA	98103	206-644-2015
White, Jack	WJ	54 Manor Pl.	Atlanta	GA	43045	299-456-7890

Sample Customer Database

Let's say you wanted to find the address of Bill Smith. First you would go vertically down the rows until you found the Smith record. Then you would go horizontally across the columns until you found the Address field.

In a sense, a similar type of thing happens when TAS Professional 5.0 searches for the Smith address. The difference is TAS Professional 5.0 can search thousands of records in the time it takes you to search only a few. This is one of the real strengths of a computerized database like TAS Professional 5.0.

In TAS Professional 5.0, the term database is used to describe a related group of information in an application. Data file is used to refer to a specific file in the database. In a database with one file the data file is the database.

File Managers, DBMSs and ADEs

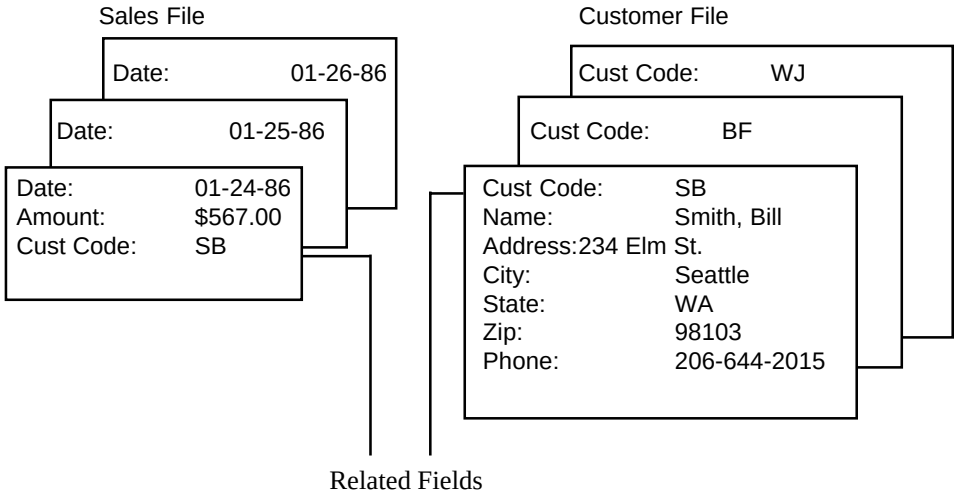
There are basically three general categories of database programs. Each is designed to do different things. The simplest form of database program is called the **File Manager**. As the name implies, this type of program is designed to manage a single file at a time.

The advantage of a File Manager is that it is fairly easy to learn and use. The disadvantage is it doesn't do very much. File managers allow you to add, delete, and search for records in only one file at a time. Because of this, they typically aren't used for serious business applications.

You could use a file manager to computerize the index card file in the previous example, since that database contained only one file. Let's say you then wanted to add a second file such as a Sales Journal to your database. At that point a simple File Manager would no longer work. You would need a **DBMS** (Data Base Management System) to manage two or more files.

A DBMS is more sophisticated than a File Manager and because of this it is usually more difficult to master. However, a DBMS gives you several advantages over a simple file manager since you can have multiple files active at the same time.

Even more important, a DBMS lets you relate files to one another. This is done by selecting certain fields that appear in more than one file. For example, you might have a Customer File which contained a field called **CUSTCODE** (short for customer code). If a related field (perhaps named **SO.CUST.CODE**) also appeared in our Sales File, it would be possible to relate the two files. (See the figure below.)



Relating files can be very important in creating real world database applications. Let's say you want to send a "valued customer" letter to each person in your Customer File who has made a single purchase of over $500. To do this you will need to search both the files shown above. First you will search the Sales File for records in which the Amount field is over $500. When you find one, you will need to find out who the customer code belongs to. You accomplish this by **relating** the record via the **SO.CUST.CODE** field to a record in the Customer file with a matching **CUSTCODE** field.

The next step up from the DBMS is the **ADE** or Application Development Environment. ADE's are systems used to create other programs (applications). Typically, ADE's include a powerful DBMS plus utilites to create custom menus, data input screens and sophisticated reports.

TAS Professional 5.0 is a very special program since it can be used as an ADE, as a DBMS or as a File Manager. It combines a powerful DBMS with a programming language and an easy-to-use "interface."

THE TAS PROFESSIONAL 5.0 TUTORIAL

Now we're ready to start working through the tutorial. You'll learn to create, link, and maintain data files; add, change, delete, and search for records; and build and then enhance a simple application with a customized data input screen and error-checking. Then you'll create reports to summarize and print your data, and develop a menu for using your application. Finally you'll learn how to restructure data files and modify an existing application program. In short, you'll see how easy it is to use TAS Professional 5.0.

To get ready for the tutorial, change directories to the TUTORIAL subdirectory, type R50 at the DOS prompt and press the ENTER key. (When we say "enter," we mean type in the appropriate command or

data and press the ENTER key. When you're just supposed to press the ENTER key by itself, we'll say that.) Now: let's get started!

PART 1 - CREATING A DATABASE

In this section we will create a Customer data file using the **Maintain Dictionary** option from the TAS Professional 5.0 Main Menu.

All TAS Professional 5.0 files must be 'defined' in the **Data Dictionary** before it can be accessed by any program. This will tell the program how many fields there are, the type and size for each field, and what fields, or group of fields, are to be used as keys (or indices).

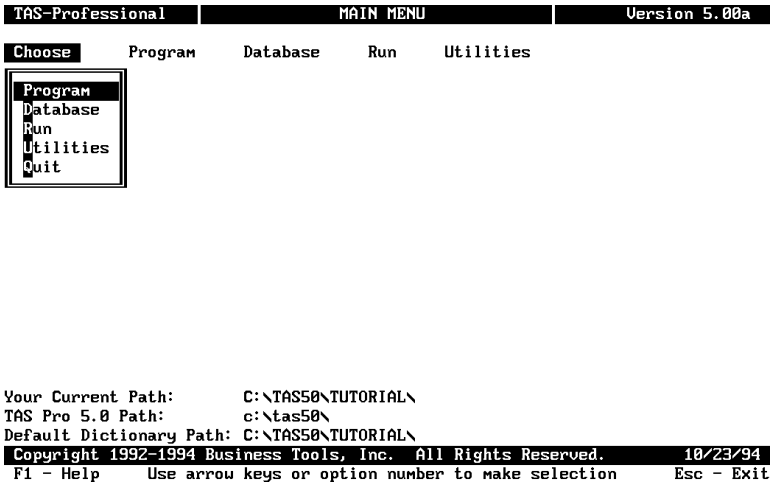
Using the Maintain Dictionary Option

In this example we will create a database to manage the following information about your customers:

1. Code
2. Name
3. Company
4. Address
5. City
6. State
7. Zip
8. Area Code
9. Phone Number

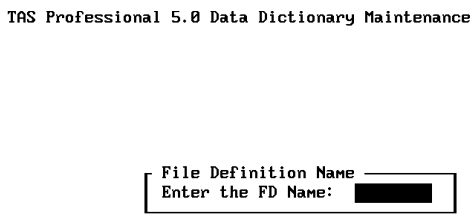
You will use the **Maintain Dictionary** option to create a database with these 9 data fields. When you are finished you will have a working database. You can then add records to it. You can also change, delete or search for records and use these fields in programs or reports you create.

In the TAS50\TUTORIAL subdirectory, type R50 at the DOS prompt. On the main TAS menu (shown next page) use the right arrow key to move to the Database topic menu. Press ENTER to select **Maintain Dictionary**.



TAS Professional Main Menu

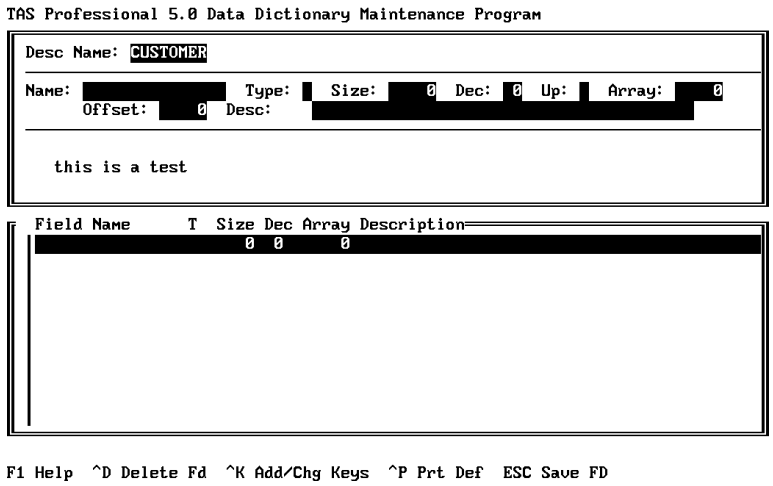
The screen below asks you to "Enter the FD name."



Maintain Dictionary Screen

This is the name of the structure for the file you are creating. Through the use of the Data Dictionary you can have many files that use the same structure, or what we call the File Descriptor (FD). In most cases, however, you will always have a file that has the same name as the FD. This isn't required but is easiest when trying to remember the FD and/or file name. In this case we will use the same name for both. Type in **CUSTOMER**, then press ENTER.

If this is the first time you have entered this FD the program will tell you that it doesn't exist and ask if you want to create it. Press the ENTER or Y key.



Data Dictionary Maintenance Screen

The screen shown is divided into two sections. The first is the field entry window. This window is where you enter the specifications for the field being added. The second is the list of fields already entered for this FD. Since this is a new FD, there are no fields defined for this FD yet; this list is empty.

To move the cursor to the field entry window press the ENTER key. Since the choice line in the list window is blank the program will enter a new item. The first entry is the Field Name. Type in **CUSTCODE**, then press ENTER.

The next entry is the field type. To get a menu of the choices available press the F1 key. The available types are A - Alphanumeric, N - Numeric, D - Date, T - Time, L - Logical, B - Byte, I - Integer, R - Record, V - Overlay. Since this field may contain any printable character including numbers, letters, and symbols, enter A for alphanumeric.

Next you will enter the field size. Type **3** and press ENTER. This will make the field 3 characters long.

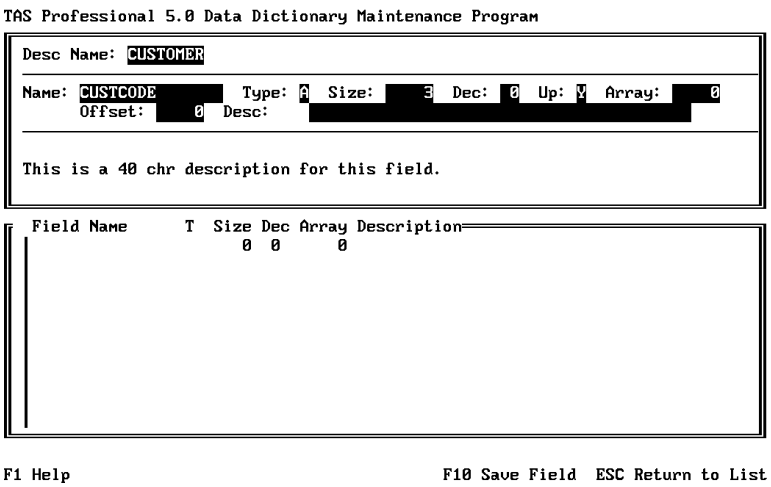
Next is the number of decimal characters. Because this field is type A (alphanumeric), it is bypassed automatically.

The next entry is Upper Case. Since this is an alphanumeric field you can force all entry to upper case characters only. Since this is a code that we will be using to search for customers we will want to make sure that all characters are entered in upper case. The codes: abc and ABC are not the same to a computer. You might not check all the possibilities when searching for the customer and the code 'abc' would come after any ZZZ codes so you might not even see it in a list. The default entry is **Y**, so just press ENTER to accept it.

The next entry is the number of array elements for this field. For this field it is 0 since we're keeping just one customer code for the record.

> A word about arrays. An array allows you to keep multiple values of the same type using the same field name and changing just the element number. This is useful when keeping track of certain types of information, for example, sales by month. You might have an array called MONTHLY_SALES with 12 elements. Element 1 (corresponding to the sales for month number 1) would be MONTHLY_SALES[1], element 2 - MONTHLY_SALES[2], etc. to element 12, MONTHLY_SALES[12]. Since you can use any legal TAS Professional 5.0 expression as the element specifier (the value within the square brackets), this becomes a very flexible way of accessing field values. For example, using the same field as above the amount of a sales order might be saved in the monthly sales field as follows:
>
> MONTHLY_SALES[month(date())] = TOTAL_ORDER_AMT
>
> This would save the value of TOTAL_ORDER_AMT into the element of MONTHLY_SALES that corresponds to the month of the current date. (date() returns the current system date and month() returns the month number (1-12) of a date value. In this case TAS Professional 5.0 first gets the current system date and then the month number of that date. This is then used for determining the element number in MONTHLY_SALES. If the date is August 10, 1992 then the element number is 8).

You can then enter a 40 character description for this field. This description will be displayed whenever the field is listed in any TAS Professional 5.0 utility program.



Entering a Description for a Field

After you press the ENTER key at the Description field the program will ask if the entry is correct. If it is, press Y or the ENTER key to accept the default. If the entry is not correct, press N and the program will return to the entry screen.

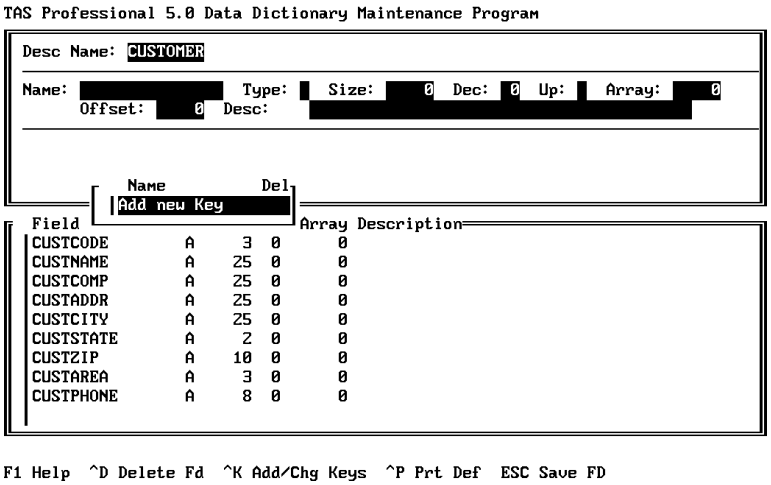
If you made an error or if you want to change something, you can go back to that line by pressing the UP ARROW key, the press the ENTER key and that field line will be back in the entry window. You can make any changes to an FD as desired without concern since you will be allowed to decide whether to keep any changes made at the end before the program updates the data dictionary.

The program automatically moves to the next line. In this case, since we are entering a new FD, the cursor is again at the enter new item line (a blank line) at the end of the list. To enter the next field just press the ENTER key. Use the field specifications in Table T-1 on the following page for the rest of the fields in the Customer data file. (In the case of the following fields all Array sizes are 0.)

Field Name	Type	Size	Dec Chrs	UpCase	
CUSTCODE	A	3	0	Y	;Already Entered
CUSTNAME	A	25	0	N	
CUSTCOMP	A	25	0	N	
CUSTADDR	A	25	0	N	
CUSTCITY	A	25	0	N	
CUSTSTATE	A	2	0	Y	
CUSTZIP	A	10	0	Y	
CUSTAREA	A	3	0	N	
CUSTPHONE	A	8	0	N	

Table T-1
Customer Data File Field Specifications

After you have entered the rest of the fields you are now ready to specify the key fields. Key fields are used to determine the way a database or file is searched or sorted. To enter the keys press the ^K key (CTRL and K at the same time).



Adding a Key

The program will display the screen above. Press the ENTER key to add a new key.

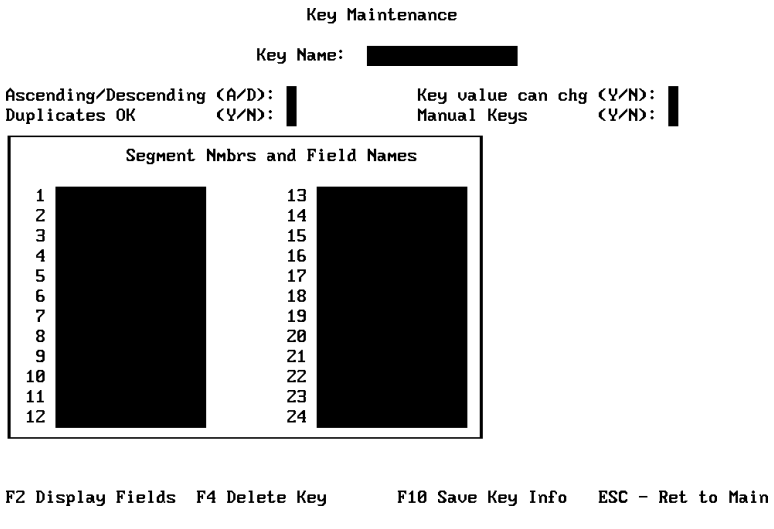
The first step is to enter the key name. For this first one enter **CUSTCODE**.

> This does not necessarily have to be the same as a field name; however, if there is only one field as a part of this key you should use the field name. You can press the F2 key and get a listing of fields for this FD.

The next entry is whether the key will be sorted in Ascending (increasing) or Descending (decreasing) order. Normally this would be A or ascending. In this case enter A or press the ENTER key to accept the default.

This next field, Duplicates, can restrict whether or not duplicate keys can be saved. Since this is the customer code we don't want to allow duplicates so enter N here.

If you don't care whether or not a key value changes after the initial entry enter Y at the next field. However, in this case, where there will be other files that will contain the same key value, you don't want the key to change after the initial entry. If you have sales records that use the same customer code and then



Key Maintenance Screen

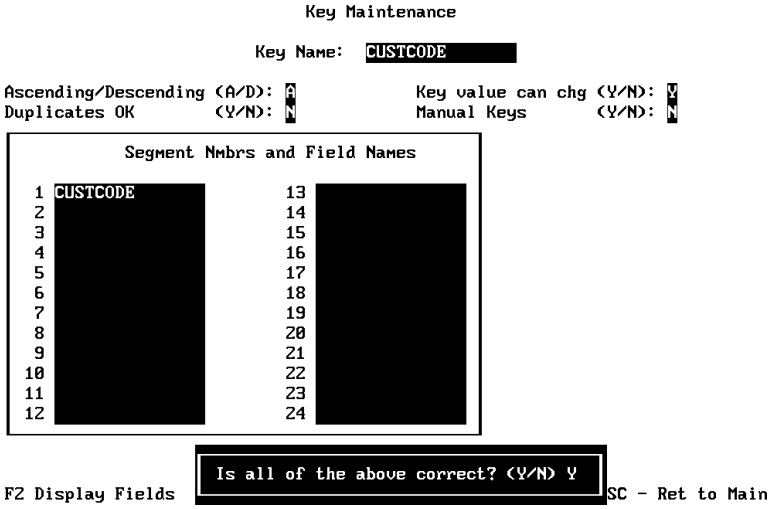
someone changes the value in the main customer record you may not be able to match up records from the sales file with the customer file because the code has changed. To prevent this enter N here; we don't want the key value to change.

The manual keys option is beyond the scope of this tutorial. Just accept the default value by pressing the ENTER key.

Any TAS Professional 5.0 file may have up to 24 different key segments. That may be 24 keys each with only 1 segment or 1 key with 24 segments. To search a file using a different method (or field) requires that you have different keys. In this case we will end up with 4 different keys, each with only 1 segment. Determining how many keys to have and how many segments will make up each key is up to you. However, we recommend that you use a few keys as possible and keep the size of each key as small as possible. This will keep the file access fast.

> If you have more than one segment in a key it sorts in the following manner: The first field is used as the primary sort and then the file is sorted further by the following segments. For example, you might have a key with two segments; the first is the customer code and the second is an invoice date. If you were to search within a file by that key you would see all records for a particular customer code in date order. Then when the search reached the end of that customer it would start with the next customer in order again, beginning with the first sale date for that particular customer. If you wanted the file to be accessed in date/customer code order also you would create a second key where the first segment would be the sale date and the next would be the customer code. Then when you accessed the file using that key you would see all sales for that date in customer order, then the next date, etc.

Enter the field name **CUSTCODE** now. Since there is only one segment in this key (and all others in this example) when the cursor moves to the next line just press the ENTER key. The screen at the top of the next page will appear.



Verifying Your Key Entry Data

The program will ask you to verify your entries and if you enter Y it will return to the main screen. Now, when you press the ^K key again, it will list the first option, Add new Key, and the name of the key you just entered. To change a key previously entered move the cursor to that line and press the ENTER key.

> If you change or add a key to a FD you must either Initialize the file (if there are no records already entered, or at least none you want to keep), Reindex the file, or Restruc-

ture the file. If you have not changed the FD structure you need not Restructure the file; Reindex is the preferred method in this case.

Now enter the other three keys using the data in Table T-2. The entries are as follows (all are Ascending/ Descending='A' and Manual='N'; also the key name and the only segment name are the same):

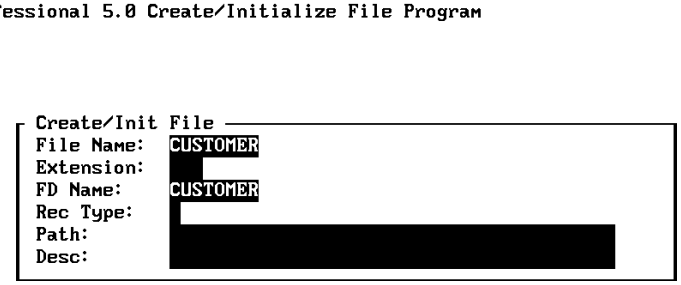
Key/Segment Name	Duplicates	Key Can Chg	
CUSTCODE	N	N	;Already Entered
CUSTNAME	Y	Y	
CUSTSTATE	Y	Y	
CUSTZIP	Y	Y	

Table T-2
Customer Data File Key Specifications

After you have entered the last key press the ESC key. The program will ask if all is correct: enter Y or press the ENTER key to accept the default.

Since this is a new FD the program will automatically initialize the file. The screen at the top of the next page is displayed. Press the ENTER key to accept the file name. Each file can have a different extension; however, the default is B. Press the ENTER key to accept that.

The next entry is the record (file) type. All TAS Professional 5.0 (Btrieve) files are type T. Press the ENTER key to accept the default.



File Initialization Screen

Next is the Path and Description. The Path is the location of the file. Since it is going to be located in our default directory you need not enter anything, just press the ENTER key. The Description is a 40 character field you can use to say anything about this file you desire. When files are listed in any TAS Professional 5.0 program this description will be displayed also.

After you press the ENTER key at the Description field the program will automatically create (initialize) the file. This must be done before any data can be entered.

After the initialization process is complete the program will ask if you want to add records to the file now. At this point enter N and the program will return to the initial screen. Press the ESC key to return to the Main Menu.

This completes Part 1 of the tutorial.

PART 2 - MAINTAINING A DATABASE

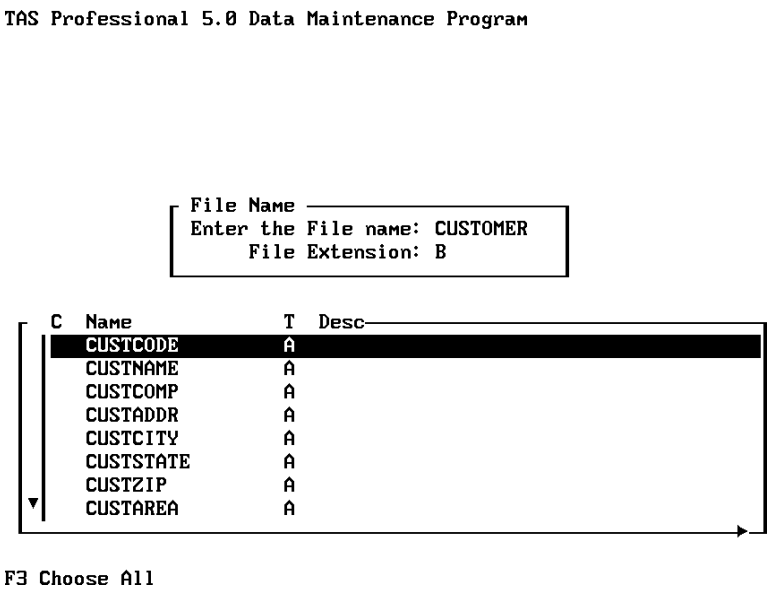
Now that you have created a CUSTOMER database with TAS Professional 5.0, you can start using it right away. You can add, edit, and delete records and generally keep things up to date. This type of activity is known as "maintaining a database."

From the TAS Professional 5.0 Main Menu, use your left/right arrow keys to move to the Database topic menu. Press D to execute **Maintain Database**.

The program then asks you to enter the File Name. At this point you can type in the name of the file you want or you can press F2 (Function key 2) and TAS Professional 5.0 will give you a list of all the files available. Enter the file name **CUSTOMER**.

> If you choose to enter records directly after creating a new FD the program will bypass this first part and go directly to choosing the fields.
>
> You can access non-TAS files using this program also as long as they are defined in the Data Dictionary.

The program will also ask you for the file extension. The default is B. Press the ENTER key to accept the default value.



Selecting Fields for Adding Records to a File

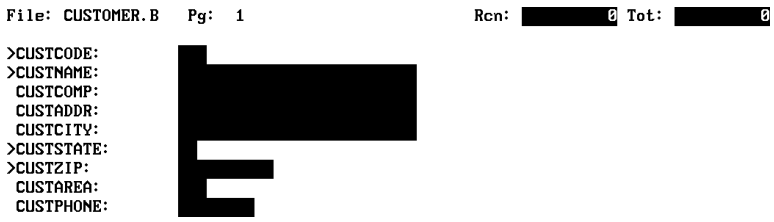
After you enter the file name and extension the program will display a list of the fields available for that file. You can choose a field by pressing the ENTER key when the cursor line is on it or all fields by pressing the F3 key. You can also un-choose a field by pressing the ENTER key while on that field if it has been chosen previously. For this tutorial we will enter data into all fields in the record so press the F3 key. To exit the list press the ESC key.

The program will then ask for an owner name. This is used in protecting the file from outside access. Since this file hasn't been protected you can just press the ENTER key to continue.

The program will list the keys available for this file. We don't need to choose one here so just press the ESC key to bypass this menu.

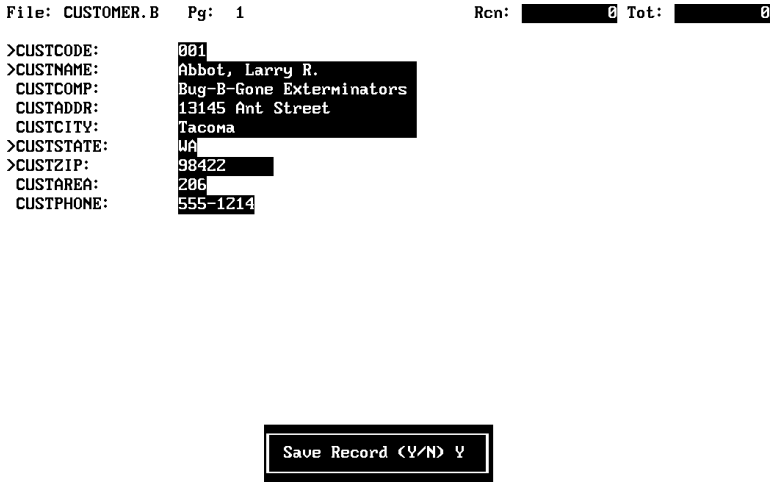
Add a Record to a File

After the file name is entered and fields selected the above screen is displayed. TAS Professional 5.0 displays a complete list of all the fields down the left side of the screen.



Data Input Screen

To the right of the field names is a set of reversed video areas known as data input blocks. This is where you enter the data for each field. At the top of the screen are two fields that will show the current record number and total active records in the file. The record number displayed is not a 'true' record number but the location value of the physical record in the file.



Enter the information shown in the screen above for the first record. Notice that alphanumeric fields always start entry at the left-most position and numeric always start at the decimal point or the right-most position. Date fields are displayed with slashes in the proper position; you only need to enter the numeric portion of the date. Likewise, time fields are displayed with colons and you only need to enter the numeric portion.

After entering the data the program asks if you want to save the record. If all entry was correct enter **Y**. If not enter **N** and the data will remain on the screen and the cursor will return to the first field. You may then edit the data as necessary.

Enter the next four records using the data in Table T-3. You may also choose your own data to enter if desired.

CUSTCODE	**002**	CUSTCODE	**003**
CUSTNAME	**William, Ralph V.**	CUSTNAME	**Larson, David**
CUSTCOMP	**Pro Musica**	CUSTCOMP	**Little Greenhouse**
CUSTADDR	**202 Maple Lane**	CUSTADDR	**702 Monroe Ave.**
CUSTCITY	**Bridgeport**	CUSTCITY	**Monroe**
CUSTSTATE	**CT**	CUSTSTATE	**WA**
CUSTZIP	**01255**	CUSTZIP	**98720**
CUSTAREA	**217**	CUSTAREA	**206**
CUSTPHONE	**444-4321**	CUSTPHONE	**234-8765**

CUSTCODE	**004**	CUSTCODE	**005**
CUSTNAME	**Manrique, Rick**	CUSTNAME	**Levins, Brenda**
CUSTCOMP	**La Villa Restaurant**	CUSTCOMP	**Bedroom Boutique**
CUSTADDR	**1774 East State Street**	CUSTADDR	**15543 Webster St.**
CUSTCITY	**Santa Monica**	CUSTCITY	**Chicago**
CUSTSTATE	**CA**	CUSTSTATE	**IL**
CUSTZIP	**91543**	CUSTZIP	**45903**
CUSTAREA	**213**	CUSTAREA	**519**
CUSTPHONE	**765-2576**	CUSTPHONE	**567-4313**

Table T-3
Customer Data

Finding Records in a File

Now that you have created a **CUSTOMER** database and saved some records in it, you can begin experimenting with some of the features of TAS Professional 5.0.

As mentioned earlier in this manual, TAS Professional 5.0 can typically find any one record in one second or less. TAS Professional 5.0 does this by searching for records according to key fields. Recall that when you designed the **CUSTOMER** database, you designated four fields to be keys. They were:

CUSTCODE
CUSTNAME
CUSTSTATE
CUSTZIP

This means you can search for records in the **CUSTOMER** database by customer code, by customer name, by state or by zip code. You could also start at the beginning or the end of the file and step through the records one by one. Of course that would take a lot more time than searching with a key. In this program keys are designated by the '>' to the left of the field name, so, the customer code key is:

>CUSTCODE

To find a record in the data file use the Record Search function keys, **F5** - Find Beg, **F6** - Find End, **F7** - Find Previous, **F8** - Find Next, and **F9** - Find Generic.

Use your up/down arrow keys to move the cursor to the CUSTNAME data input block. Type **Lev** and press the **F9** (find generic). The record for Brenda Levins appears.

What you have just done is known as a "generic field" search. In other words, you typed in a field of characters which you wanted to search for, then TAS Professional 5.0 went into the database and hunted for an match closest to the data entered. In this case, an exact match was made. With just part of the name having been entered, the record with a name equal to or greater than the portion of the name entered was

retrieved. You can do this for any of the keys by moving the cursor to that field and then pressing one of the appropriate file search keys.

Another way to find records in the data file is to use the F5 (find beginning) or F6 (find end) function keys. Place the cursor in a key field and press F5. The first record in the search path is displayed. Pressing F6 will retrieve the last record in the search path. F3 will clear a record from the input block.

Once a record is found, thereby establishing a search path, you may also use the F7 (find previous) and F8 (find next) function keys to "walk" through the file; however, the cursor must remain on the key field for this to work.

Change (Edit) a Record

What happens when one of your customers moves? How do you change his (or her) address? It's really very easy. TAS Professional 5.0 lets you change (edit) or add to data that already exists in your database. In this section we'll show you how.

Now let's say you want to update the address for Brenda Levins. Press F3 (Clear Record) to clear any record currently showing on the screen. Use the F9 (generic search) to find Brenda Levins' record (review the preceding section on finding records if necessary).

Position the cursor on the CUSTADDR field to change the record. Press ^U key (CTRL and U key at same time) to clear the existing address. Enter **456 Bondman Ave. W.** as the new address. Press F10 (Save Record) to save the record and the new address. When the program asks "Save Record," if all the data is correct press **Y** and the data on the screen will be cleared. If you want to make additional changes press **N**. The record remains on the screen allowing additional editing.

Delete a Record from a File

Deleting a record from a file works much the same way as changing a record. That is, first you find the record with the F9 - Find a Record key, then you delete it with the F4 - Delete a Record key.

Let's say you just learned that Brenda Levins sold her business and you want to delete her record from the database. Press F3 (clear record) to clear any existing record from the screen. (Note: This only clears the screen; the data contained in the file is not changed.) Find the record for Brenda Levins using the F9 (generic search) function key (review the preceding section on finding records if necessary).

Press F4 (delete record) to indicate you wish to delete the record. A message appears at the bottom of the screen asking if you want to "Delete record ? Y/N Y." If you are sure this is the correct record to delete just press the ENTER key to accept the default Y for yes or press **Y**.

When you are finished press the ESC key. This will return you to the initial entry screen. Press ESC again and you will return to the main menu. This completes Part 2 of the tutorial.

PART 3 - CREATING A NEW SALES PROGRAM

In this section you will learn how to use the Create Program function to build an application called **SALES**. This **SALES** application is really a simple sales journal. You could use it to record daily sales activity including Order Number, Customer Code, Sales Amount and other pertinent information. In later parts of the tutorial, we'll learn how to add a menu and report to this application.

TAS Professional 5.0 lets you create complete applications, not just simple databases. Let's take a moment to talk about what an application is. In its most basic form an application is a program combined with at least one customized data input screen and one database file.

The real significance of this is the customized data input screen. TAS Professional 5.0 lets you "paint" your computer screen to look like an actual paper form that you use in the office. Because the computer screen will look like an actual form, it makes it a lot easier for non-computer people to use the applications you create.

There are two ways of creating a new database application with TAS Professional 5.0. One way is to choose the Screen Formats->Edit screen format function which is part of the Program topic menu on the TAS Professional 5.0 Main Menu. This program will allow you to create a screen format and then a program from that format. It is the fastest and easiest way to create a single-file application.

The second way to build an application is to choose the Edit Program function which is also part of the Program option on the TAS Professional 5.0 Main Menu. This allows you to go directly to the built-in Source Code Editor/Checker. You would use this method to build sophisticated applications, such as accounting programs or to add certain features to your application that can't be done through the screen painter. However, the screen painter in TAS Professional 5.0 is so complete that you can enter help messages, control entry values, setup relationships among multiple files, and even attach source code directly to the screen format so that it is appended automatically to the program when it is created! For now, we'll stay with the Screen format/Make Program function.

> This section assumes that you have already worked through the **CREATING A CUSTOMER DATABASE** and **MAINTAINING A DATABASE** sections of this tutorial and are familiar with the conventions presented there. If not, you should go back and work through those sections first.

Design your Database on Paper First

The first and most important step in building a database application is to design it on paper before even turning on the computer. A little time spent thinking things through now can greatly reduce the time (and frustration) spent fixing things later on.

Your design doesn't have to be fancy. In fact, a good place to start is to list the types of data (information) that will make up an application. Here's a list of what we want to keep track of in our **SALES** example.

1 - Order Number
2 - Date of Transaction
3 - Sales Amount
4 - G/L Account Number
5 - Customer Name
6 - Customer Code
7 - Company Name
8 - Customer Company
9 - Customer Address
10 - Customer City
11 - Customer State
12 - Customer Zip

The next thing to determine is how the data is going to be stored. In this example, we will be entering sales order records based on the customers' purchases. We already have the customer records from the previous sections, so we do not need to keep the customer specific information in the sales file. We do, however, need to keep the Customer Code in the sales file so we can link (relate) the sales records to a specific customer.

This does two things for us. First, it reduces the data entry required to enter a sales order. When the Customer Code is entered, the program will find the corresponding record in the customer file and display the appropriate information. We will be adding the code to do this later.

Secondly, it reduces the amount of disk space required to store our application. Each file stores only the data required and the fields necessary to link to other files in the application.

The following table shows how the data we are keeping track of will be stored.

File:	Sales Order	File:	Customer
Fields:	Customer Code	Fields:	Customer Code
	Order Number		Customer Name
	Order Date		Customer Company
	Sales Amount		Customer Address
	G/L Account		Customer City
			Customer State
			Customer Zip
			Customer Area Code
			Customer Phone

The Customer file already exists so we will only be entering the Sales file. The first thing we need to determine is the type and size of each field in the Sales file. The size of the field is determined by the largest entry you expect to put in that field. For example, you wouldn't make the Sales Amount field only four characters long, since that would limit the sales amount to a maximum of 9.99.

Here are the data types, lengths and key types for the **SALES** file:

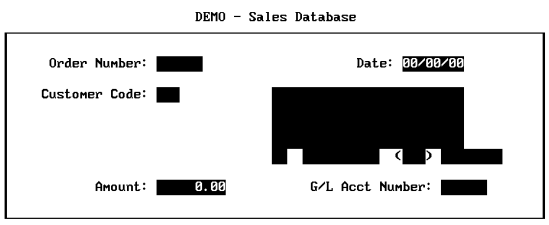
Data	Field Name	Type	Size	Dec	Up
Order Number	SO.NUMBER	A	6		Y
Date of Transaction	SO.DATE	D	8		
Sales Amount	SO.SALES.AMT	N	9	2	
G/L Account Number	SO.GL.ACCTNUM	A	6		Y
Customer Code	SO.CUSTCODE	A	3		Y

Key/Segment Name	Asc/Desc	Duplicates	Key Can Chg
SO.NUMBER	A	N	N
SO.DATE	A	Y	N
SO.CUSTCODE	A	Y	N

The keys specified allow searching and reporting based on the Order Number, the Date of Transaction, or the Customer Code. Note that we now allow duplicates of the Customer Code. Since you can have multiple sales for a single customer you need to allow duplicates here; however, we still don't allow the code to change for the same reasons we had when setting the Customer Code key.

The final thing to design on paper is what you want the computer screen to look like. Often you can get good ideas by looking at the paper forms around your office. For example, if you were building an invoice tracking system, you might design the computer screen so it looked like your paper invoices. With TAS Professional 5.0 you have the ability to make the screen look any way you want.

For our **SALES** example, we'll keep things fairly simple. The data input form will look something like this:



Data Input Screen for SALES Application

You will learn how to build this screen in the next section.

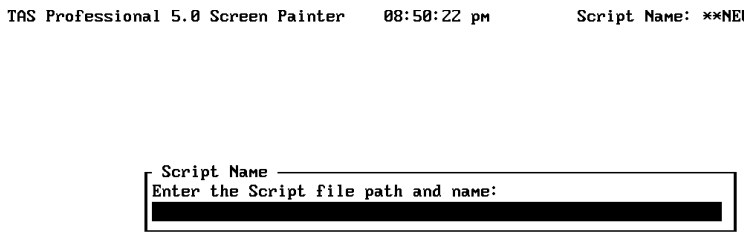
The Data Dictionary

The first step in creating your application program is to add the **SALES** file to the data dictionary. Review what you learned in **PART 1 - CREATING A DATABASE.** Using the **Maintain Dictionary Program** and the specifications above, enter the FD (file descriptor) for the **SALES** file.

Create a Program

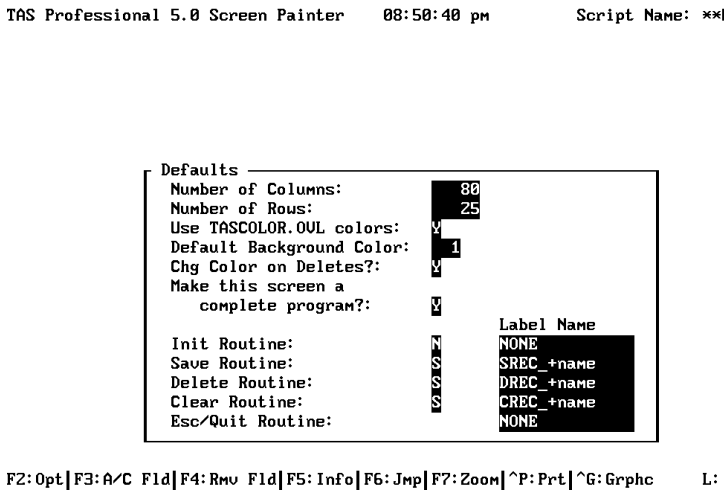
Now that you know what you are going to keep track of in your database, you are ready to use the Screen Painter to “paint” a data entry form on the screen.

From the Main Menu select the Program topic menu. Press **S** to select the Screen formats option, then press the ENTER key to select the Edit Screen Format option.



Screen Painter Opening Screen

The first entry is the screen format name. In this case we are creating a new format (or "script") so just press the ENTER key.



Screen Painter Defaults

The screen above is displayed; it allows you to enter general information about the screen format you are creating. For the purposes of this tutorial we aren’t going to make any changes so press the ESC key to accept the default values.

Screen Painter Entry Screen

The screen displayed above is basically a blank canvas. You can put anything you want on it. Across the top of the screen is a ruler marked off in 10 character increments. The 1 specifies the 10th character (the far left hand character is number 1). Across the bottom of the screen are the function keys available. They are:

F2 Display an option menu (the option menu is described on the next page).

F3 Add a field to the format or change an existing field. The cursor must be at the appropriate location to add a field or must be on the field to change it.

F4 Remove a field from a format. The cursor must be on the field to remove it.

F5 Display the field information about a previously added field. The cursor must be on the field to get this information.

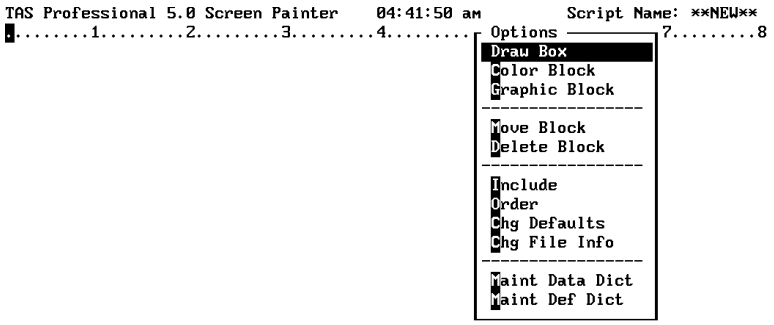
F6 Jump to the next field. The cursor will go to the next field on the format, from left to right, top to bottom.

F7 Only 22 lines can be displayed on the screen at one time. By pressing the F7 key you will see the entire format displayed just as it would if you were running this program. Press any key and the program will return to the entry screen.

^P This will print the screen format and any appropriate field information to the screen, printer or disk.

^G This will allow you to put a graphic character on the screen at the current cursor location. A list of characters will be displayed that you can choose from.

If you press the F2 key, the Option Menu below will appear.



Option list

Draw Box - This option will allow you to draw a box on the screen. It will start at the current cursor location and you'll move the cursor to the lower right hand corner. Press the ENTER key and you will then be able to choose the box type and color.

Color Block - This option will allow you to paint a color block on the screen. It will start at the current cursor location and you'll move the cursor to the lower right hand corner. Press the ENTER key and you will then be able to choose the color.

Graphic Block - This option will allow you to paint a block of graphic characters on the screen. It will start at the current cursor location and you'll move the cursor to the lower right hand corner. Press the ENTER key and you will then be able to choose the appropriate graphic character from a list.

Move Block - This option will allow you specify a block of characters to be moved. It will start at the current cursor location and you'll move the cursor to the lower right hand corner. Press the ENTER key and you will then be able to move the cursor to the new upper left corner for the block.

Include - You can specify up to 18 source code files that will be attached to the code created for this screen format at the time it is compiled.

Order - The Make Program option will automatically create **ENTER** commands for the fields on the screen format. These are in the same order as they appear on the screen from top to bottom, left to right. This means the field closest to the upper left corner will be entered first, and the field closest to the lower right corner last. If you wish to change that order, i.e., two columns of fields to be entered and you want to enter 1 column and then the other, you would use this option. You will be able to specify which field comes first, second, etc. In this tutorial we will accept the default order.

Chg Defaults - Use this option to change the values entered in the Default Screen that appears when this format was first created.

Chg File Info - If you have multiple files you can use this option to specify which will be primary and the relationships of the others. You can also specify the primary search key to be used or allow the program to ask at runtime. We will use this option in this section to setup the relationship between the **CUSTOMER** and **SALES** files.

Maint Data Dict - Use this option to access the **Maintain Data Dictionary** program directly, without having to return to the **Main Menu**.

Maint Def Dict - TAS Professional 5.0 requires that all fields in a program be 'defined' before they are used. This can happen in three ways: The field can be part of a file that is in the data dictionary; it can be defined in the program through the use of the **DEFINE** command; or, it can be part of the defined data dictionary. You can access the maintenance program for that file directly through this option or using the **Maint Defined Fields** option in the **Main Menu**.

Moving Around the Screen

To move around the screen you can use the following keys:

Up, Down, Left & Right Arrow keys - These will move the cursor one character up, down, left, or right on the screen respectively. The ruler bar at the top of the screen and the line counter in the lower right hand corner will follow your movement so that you will know exactly where you are at all times. As you move to the bottom of the screen if there are lines below they will be moved up, and likewise, if you move to the top and there are lines above, they will be moved down.

Tab keys - Tab and Back Tab (shift) will move the cursor forward or backward 5 characters at a time.

Home - Will move the cursor to the beginning of the current line.

End - Will move the cursor to the end of the current line.

Pg Up - Will move the cursor to the upper left hand corner of the current screen.

Pg Dn - Will move the cursor to the lower right hand corner of the current screen.

^Pg Up - Will move the cursor to the far upper left hand corner of screen format, column1, row 1.

^Pg Dn - Will move the cursor to the far lower right hand corner of the screen format, column 80, row 25 (or whatever maximums you have entered in the Defaults screen).

Backspace - Will delete the character to the left of the cursor and move all characters from the current cursor column to the end of the line 1 character to the left. You will not be able to backspace over a field specifier.

Delete - Will delete the character at the cursor and move all characters from the current cursor column+1 to the end of the line 1 character to the left. You will not be able to delete a field specifier character.

Insert - If you wish to insert characters on the current line then press this key. It will remain on until you press it again. You will be able to tell you're in insert mode since the cursor will be a small block instead of the normal underline.

^A - Will insert (add) a line at the cursor row. All lines at the cursor and below will be moved down 1. The line at the end of the format is deleted (line 25). If there are fields on the last line you will not be able to insert a line.

^D - Will delete current line. If there are any fields on that line the program will ask if you are sure you wish to delete those also.

The keys we are concerned with now are the cursor movement keys, up/down/left/right arrow keys, the functions keys for adding (F3 - A/C Fld) and removing (F4 - Rmv Fld) fields, and the F2 (Opt) function key. Practice moving the cursor around the screen to get a feel of how the program reacts.

You can 'paint', or create, the input screen shown previously using the column and lines positions in the following table.

FIELD	COLUMN	LINE
SO.NUMBER	25	7
SO.DATE	57	7
SO.SALES.AMT	25	15
SO.GL.ACCTNUM	62	15
CUSTCODE	25	9
CUSTNAME	40	9
CUSTCOMP	40	10
CUSTADDR	40	11
CUSTCITY	40	12
CUSTSTATE	40	13
CUSTZIP	44	13
CUSTAREA	57	13
CUSTPHONE	62	13

LITERAL TEXT	COLUMN	LINE
DEMO - Sales Database	30	4
Order Number:	11	7
Date:	51	7
Customer Code:	10	9
(	56	13
)	60	13
Amount:	17	15
G/L Acct Number:	45	15

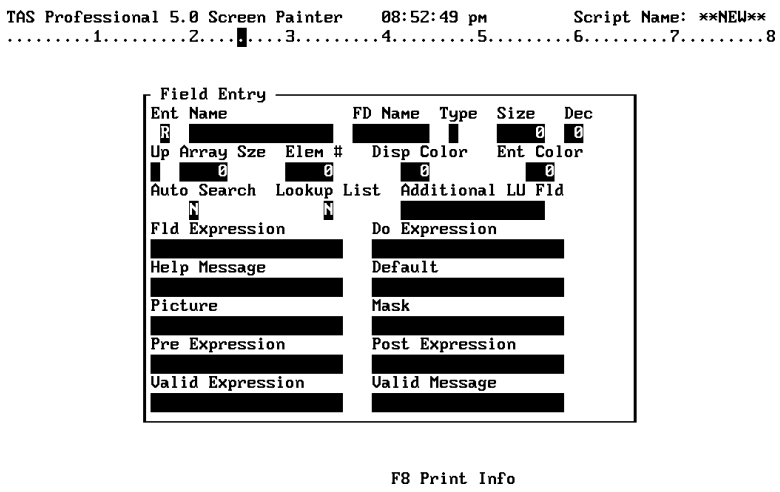
Let's enter the first field. Press the ^PG UP key to move back to the beginning of the screen format. Move the cursor using the RT ARROW key until the block on the ruler at the top of the screen is midway between the 2 and the 3. It will be on the 5th dot after the 2; this is column 25. Then, using the DN ARROW key move the cursor until the line counter at the lower right hand corner of the screen shows 7. The screen should look like this:

```
TAS Professional 5.0 Screen Painter    08:52:31 pm         Script Name: **NEW**
.........1.........2....▪....3.........4.........5.........6.........7.........8
```

```
F2:Opt|F3:A/C Fld|F4:Rmv Fld|F5:Info|F6:Jmp|F7:Zoom|^P:Prt|^G:Grphc     L:  7
```

Ready to place field on screen

Now press the F3 key. The following window will appear.



Enter field information

The following are the different entries you can make for this field:

Ent - What type of entry field is this. The options are:

R - Regular, create an ENTER command for this field. All of the fields in this tutorial will be R.

D - Display only, put it on the screen but don't allow entry.

O - Do something. You must specify a Do Expression. This must be a User Defined Function (UDF) that will be executed when the user is at this field and presses the F2 key.

A - This is an array field. When the user is at this field the different elements will be displayed by pressing the PG DOWN (increasing element number) and PG UP (decreasing element number) keys.

N - This is an enumerated field. Enter the options available in the Field Expression separated with commas. The program will display the first as the default and the user can see the others by pressing the PG UP or PG DOWN keys, or by pressing the space bar will scroll through them.

X - This entry field type allows you to enter an expression instead of a field. The expression will be evaluated and the value returned will be shown on the screen.

Name - This is the name of the field to be placed on the screen. It can be part of the file data dictionary, the defined data dictionary, or you can stipulate that it is a 'MEMORY' field and it will be defined automatically within the program created from this screen format. To get a list of fields in the file data dictionary press the F2 key. Press the F3 key to get a list of fields in the defined data dictionary.

FD Name - If this field is part of the file data dictionary the FD name will be inserted here automatically. If it is part of the defined data dictionary or you wish to define the field within the program this will be MEMORY. No entry is required or allowed.

Type - If this field is to be defined within the program you must specify the field type. If this field is part of the file or defined data dictionary this information will be automatically inserted.

Size - If this field is to be defined within the program you must specify the field size.

Dec - If this field is to be defined within the program and the type is N you may specify the number of decimal characters here.

Up - If this field is to be defined within the program and the type is A you may specify whether (Y) or not (N) all entry to this field should be in upper case.

Array Sze - If this field is to be defined within the program you may specify how many array elements (if any) there are.

NOTE: If the field is part of the file data dictionary, all of the above will be automatically filled and you will not be able to make any changes. If it is part of the defined data dictionary, the appropriate items will be filled; however, you can make changes if desired.

Elem # - If this field has array elements you must specify the element number here. You can use the same array element only once per screen format.

Disp Color - The color to be used when the field is displayed.

Ent Color - The color to be used when the field is entered.

Auto Search - If this field is a key then you will have the chance to choose this option. If you do the program will search on this field automatically after the user presses the ENTER key if a record is not currently active for the file.

Lookup List - If this field is a key then you will have the chance to choose this option also. If you do so the user will be able to see a lookup list for this field if they press the F2 key when the cursor is at this field.

Additional LU Fld - If you enter Y for Lookup List the program will allow you to enter a field to be part of the list in addition to the field being entered. Many times the actual key (index) value may not be enough information to determine whether or not it is the correct record. This option allows you to add a single field in the same file to the list so that you can get more information.

Field Expression - If you have chosen an enter type of D, O or N you will be able to enter the appropriate expression here. This is a slider type field (see description below).

Do Expression - If you have chosen an enter type of O you will be able to enter the UDF that should be executed.

Help Message - A help message can be entered here that will be displayed if the user presses the F1 key while at this field. To edit this option press the F2 key and the TASEdit routine will be automatically loaded. If you are entering an actual message it must be surrounded by quote marks.

Default - You can enter a value that will be used as the default entry value for this field. This means that if the field is blank or 0 (depending on field type), this value will be displayed to the user instead of the blank or 0. This is a slider field.

Picture - If this is a type A (alphanumeric) field you may put in a slider spec here. A slider is a field that shows fewer characters on the screen than actually exist in the field. The user can see the other characters by pressing the LEFT and RIGHT ARROW keys and the characters will appear to ‘slide’ back and forth across the available space. This is very useful when you have

several large fields and want them all to fit on the same screen. All of the larger fields in this window are sliders except Help and Valid Message. A slider field is specified with an alpha constant of "Sxx", where xx is the number of columns to display.

Mask - These are the allowable entry characters for this field.

Pre Expression - This is a UDF that will be checked before the entry is made. If the UDF returns .F. the program will go to the next line.

Post Expression - This is a UDF that will be executed after the entry is made. It is usually used for cleaning up after an entry and before the next line is executed.

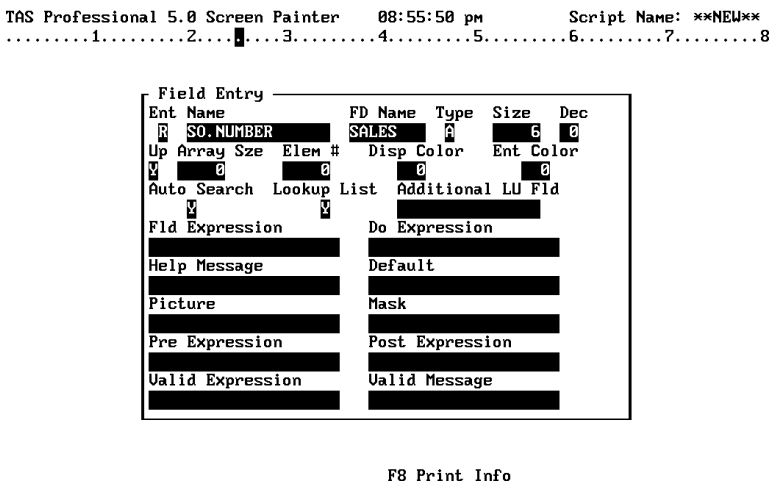
The use of both the Pre and Post Expressions are beyond the scope of this tutorial.

Valid Expression - This can be used to make sure that the entry the user made is within acceptable bounds. If this expression fails the program will display the Valid Message, or if one doesn't exist, a general message alerting the user that the entry was incorrect. This can be any expression that resolves to a true or false.

Valid Message - This is the message that will be displayed if the Valid Expression returns a false result. This uses the TASEdit program to enter the message.

Now to enter the field information. Press the ENTER key to accept the Ent type as R. Now enter the field name, **SO.NUMBER**. If you have already created the FD for the SALES file (which you should have done) the program will fill in the appropriate fields. Press the ENTER key only for Disp Color and Ent Color. By leaving these both 0 the program will use the appropriate default values as set elsewhere. Enter Y for both Auto Search and Lookup List. Now press the F10 key and then the ENTER key to let the program know that we don't have any other entries to make. We will come back later and add other information but for right now, let's just enter the basics.

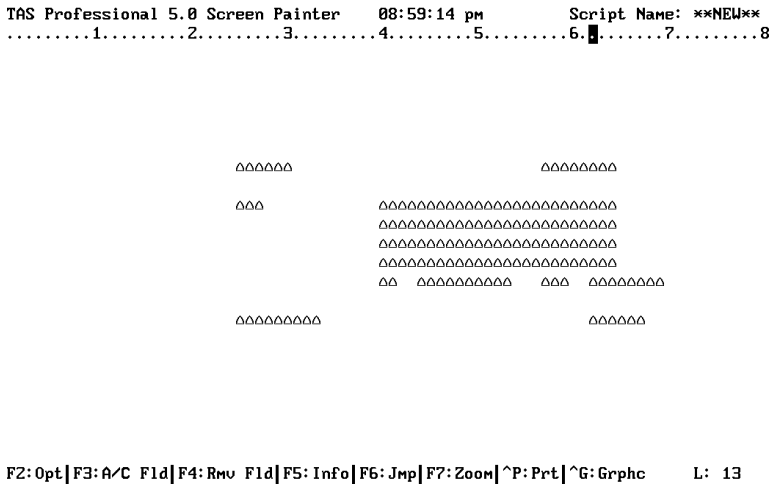
You can see what you have entered by moving the cursor to any character of the field and pressing the F3 key. This time the window will be redisplayed with the information you have just entered (see below). To return to the main screen press the ESC key and answer Y to the Return to Full Screen message.



Field Information Redisplayed

Now place the balance of the fields on the screen (using the information on page 22) by moving to the appropriate location, pressing the F3 key, and putting in the field name. For right now just enter the field names and the press the ENTER key only until you get to the Help Message entry. Then press the F10 key, answer Y and continue to the next field. For the SO.DATE field make sure that the Auto Search value is N.

If it isn't when you enter the date for a new order the program will find the first order with that date value. For CUSTCODE make sure Auto Search and Lookup List are set to Y.

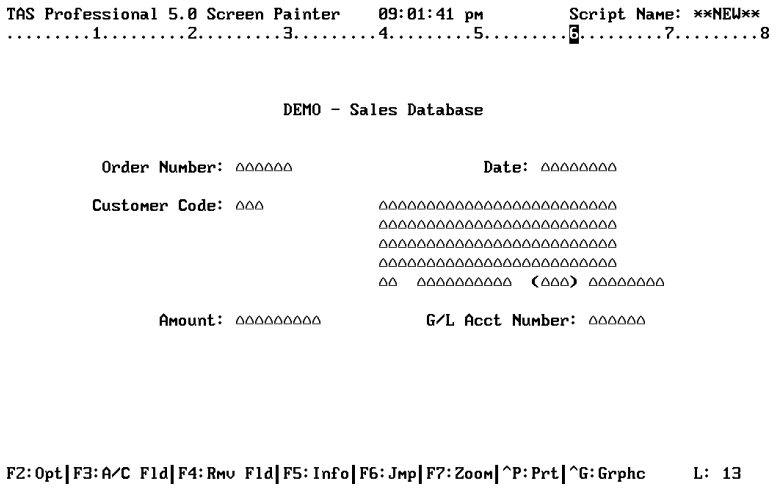


Screen with all fields but no text

Once you have entered all the fields you are ready to start entering the text so that your user will know what to enter. In normal practice you would be entering text along with fields. Move the cursor to column 30 (number 3 on the ruler) and line 4. Then just enter the text string:

DEMO - Sales Database

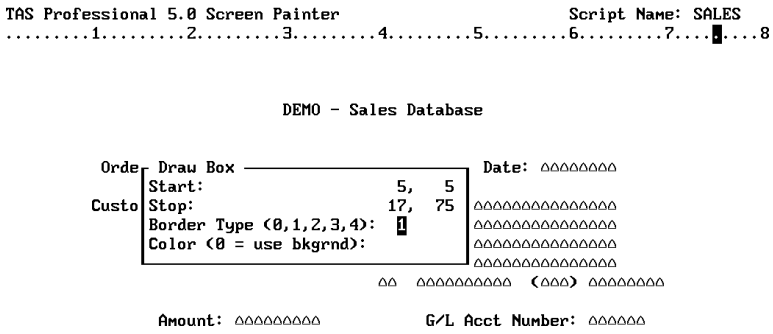
There's nothing to it. If you make a mistake you can press the BACKSPACE key or use the LEFT ARROW key and retype it. Don't worry about mistakes; you can always edit your entries. Now enter the rest of the text data found under the "Literal Text" column on page 22.



Screen with all field and text but no box

Now let's put a box around the entry fields. Move the cursor to column 5, line 5. Press the F2 key. The option menu as shown previously will be displayed. The Draw Box option is the first so to choose it, just press the ENTER key. The program will tell you to move the cursor to the lower right hand corner of the box (when you choose the option, the cursor is at the upper left hand corner of the box). A line at the

bottom of the screen will give you the current location and will be updated as you move the cursor. In this case you want to go to column 75, line 17. Press the ENTER key when the cursor is at the proper location.



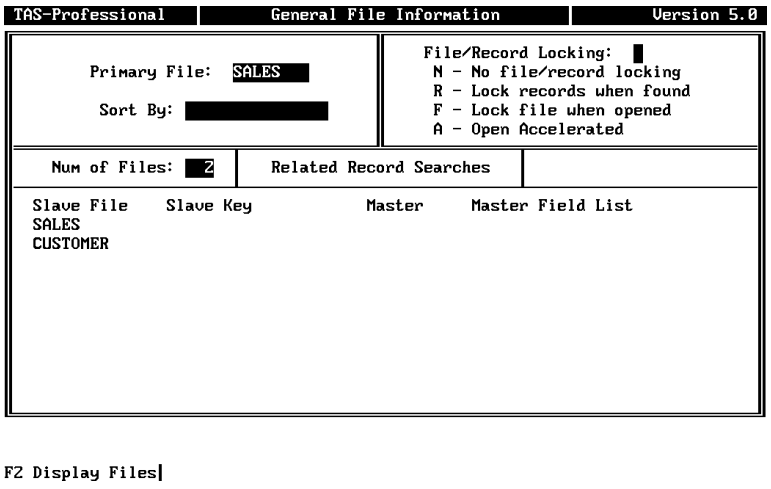
Screen with box entry window

You should now see the screen above. The Border Type options are: 0 - Clear any box currently there, 1 - Single line border, 2 - Double line border, 3 - Double line top and bottom and single line sides, and 4 - Single line top and bottom and double line sides. Enter your choice; the default is single line box. You can also specify a color but for right now let's stick with the current one and just press the ENTER key. The program will now put the box on the screen and we're almost done.

Setting File Information

The final step is to give the program certain file information. Press the ESC key to exit the screen painting process. The program will ask if you want to save the script, press the **Y** key or just the ENTER key to accept the default.

The program will then ask for the name of the script file. Enter **SALES** and press the ENTER key.



File Info screen

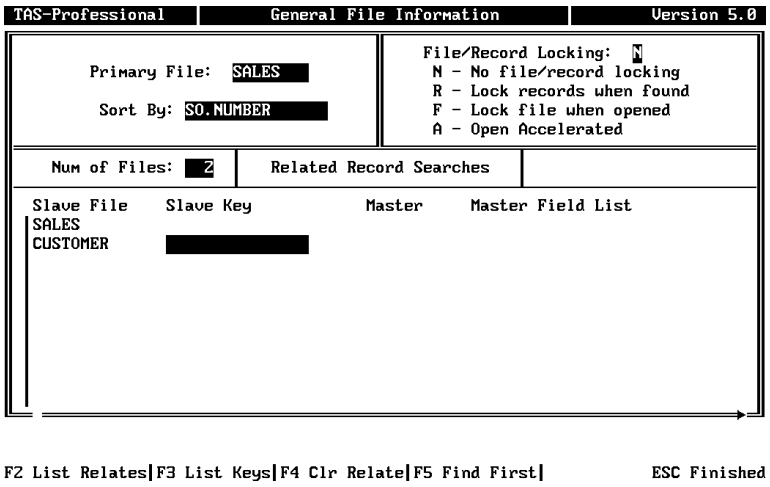
Next the above screen will be displayed. We use this screen to tell the program which file will be primary and what key to use. You can also use this screen to setup relations among the files. Since the first field we entered was in the **SALES** file the program has that file as the default primary file. Press the F2 key and you will see all the files that are part of the screen format listed. If you have pressed the F2 key and are now in a list with the two files then press the ESC key to return to the Primary File entry. Press the ENTER key to accept **SALES** since we want that to be the main file in the program. The **CUSTOMER** file will become the related (or slave) file.

The next entry is Sort By. By putting in a index name here you can set which index is to be used at runtime when searching for a record. If you leave this field blank then you will be able to choose an index when the program is run. Again, you can get a list of available indexes by pressing the F2 key. For right now let's enter an index name, put **SO.NUMBER** into the field so that all searches will be by order number.

Next the program asks about File/Record locking. Just enter N for this program.

Now the cursor moves to the bottom half of the screen. There are two files listed, **SALES** and **CUSTOMER**. If we had fields from other files on this screen format those file names would be listed here also. What we're going to do now is set the relationship between the **SALES** and **CUSTOMER** file.

Since the **SALES** file is primary we don't have to do anything with it. Move the cursor to the **CUSTOMER** file by pressing the DN ARROW key. Then press the ENTER key.



Screen with entry ready into Slave Key

You now need to enter the index (or key) that is part of the **CUSTOMER** file that has matching data in the **SALES** file. This is of course, the key **SO.CUSTCODE** which matches the field **CUSTCODE** in the **CUSTOMER** file. When a record is found in the **SALES** file the program will use the data in the **SO.CUSTCODE** field to search for the appropriate record in the **CUSTOMER** file.

Enter the key name **CUSTCODE** for the Slave Key value. We know that the master file will be the **SALES** file so enter that for the Master value. And finally, enter the field **SO.CUSTCODE** as the field in the Master Field List. If there were multiple fields (segments) in this key then you would enter each field in order separated by commas.

After you enter the Master Field List value the program will ask if you wish to update the master or slave files related fields automatically during a save. Enter **M** to update Master Fields from Slave data..

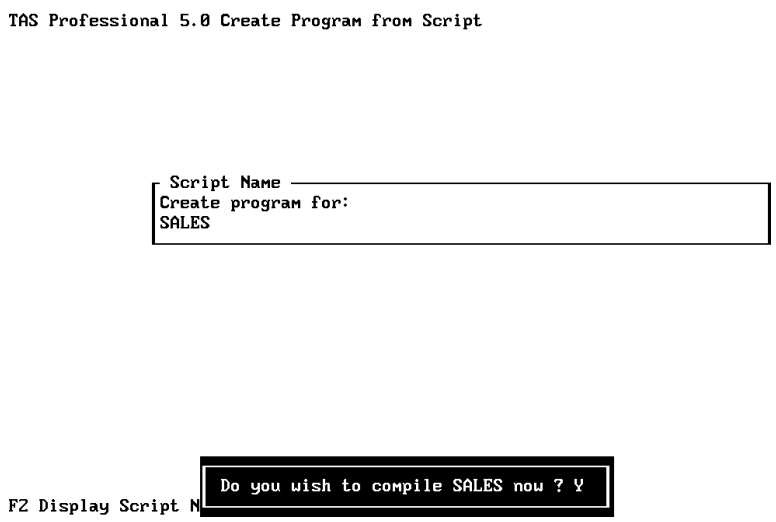
> When we painted the fields on the screen we used the **CUSTCODE** and not the **SO.CUSTCODE** field. The program will create a lookup list automatically for the customer code and we will be able to use this when searching for customers. However, the down side to this is that the **SO.CUSTCODE** field is never updated. Without that we will not know who this sale belongs to. By automatically updating this value when the record is saved you will eliminate the need for any programming to make sure that all appropriate data has been saved in the **SALES** record. With the customer code saved properly you will be able to find the appropriate customer record when you search for previous sales records.

The program will return to the entire list. You can make more changes to the line by pressing the ENTER key again, or if there were more files, move the cursor to the next file and set up that relationship.

> In this example we only had two files and so the Master had to be the primary file. However, this is not necessarily always the case. You can have a situation where you have multiple files and the master for one may be the slave of another. By setting up the relationships here you will be able to make sure that all records will be found properly.

Now press the ESC key to finish with this operation.

The program will now ask whether you want to create a program from this script. Enter **Y** or press the ENTER key to accept the default; we want to create a full program.



Make Program screen

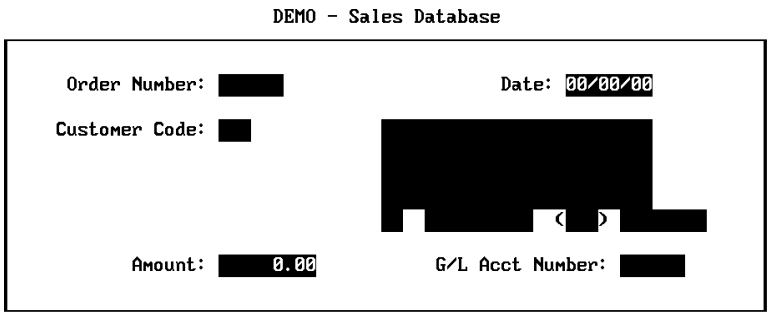
> After you create the source file the first time when you use this program again it will alert you that a file exists with that name. You will then have to enter Y to continue. This process is to protect you from accidentally overwritting a source file that has the same name as the screen format.

After the program has been created you will be asked if you want to compile it now. All TAS Professional 5.0 programs must be compiled before they can be run. Enter Y or just press the ENTER key.

After the program is compiled (just a few moments), the program will return and ask if you wish to run it now. At this time enter **N** and the program will return to the Main Menu. This completes Part 3 of the tutorial.

PART 4 - RUNNING & MODIFYING THE SALES PROGRAM

Now we're ready to run the program you created in Part 3. At the **Main Menu** move to the **Run** sub-menu. Press the ENTER key to **Run a TAS Program**. Enter the name of the program, **SALES**, and press the ENTER key. The following screen should appear:



Sales Program screen

Now enter the first record. At the Order Number field enter **1005**. Enter today's date in the DATE field. At the Customer Code field press the F2 key. A list of customer codes will be displayed. Enter **004**. The cursor should now be on the appropriate line. Press the ENTER key and the data for customer 004 is displayed. Since the program will update both records (**SALES** and **CUSTOMER**) you can make any changes here for the customer and it will be saved properly.

Enter **451.25** for the Amount and **101** for the G/L Acct field. After entering data in the G/L Account field the program asks 'Save the record above Y'. Press **Y** to save the record. Both records are saved, the screen is cleared, and the cursor is returned to the Order Number field.

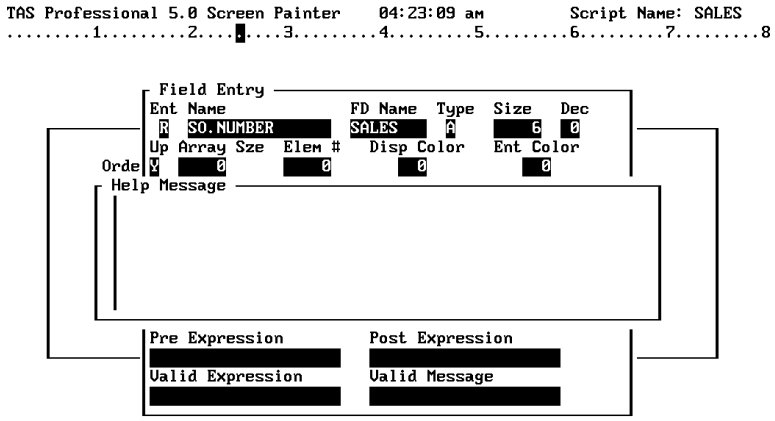
If you now press the F2 key a list with one record will be displayed. This is the sale you just entered. If you press the ENTER key it will be redisplayed, along with the appropriate customer record. To return to the main screen press the ESC key. If the record is on the screen and you wish to clear it press the F3 key.

You can enter some more records, if you wish. When you're done press the ESC key and you will return to the Main Menu. You're now ready to modify your **SALES** program.

Modifying the Sales Program

There are some things we can do to make the program easier to use and more foolproof. These include making sure certain entries are checked, including help messages, etc. These are very easy to add using the TAS Professional 5.0 Screen Painter.

Run the screen painter from the **Main Menu** (you should be an old hand at this by now). At the script name entry screen enter the file name **SALES**. The familiar screen will be displayed and the cursor will be in the upper left corner. Press the F6 key and the cursor will move to the first field, Order Number. Press the F3 key and the field information will be displayed. Now press the ENTER key 6 times until the cursor is in the Help Message field. Now press the F2 key. The TASEdit™ screen will be displayed.



Edit Help Message in TASEdit

Within this screen enter the following (include quote marks). Don't press the ENTER key at the end of the line; the program will automatically wrap your line within the window:

```
'This is the number that will be used when referring to this order.'
```

At the end press the ENTER key and the program will return to the main field entry screen. Part of the message will be displayed in the entry field. If you wish to make further changes move the cursor back to the Help Message field and press the F2 key again.

You now have a help message that will appear when the user is at that field and presses the F1 key. It's just that simple!

When you leave the TASEdit routine the cursor automatically moves to the Default Value field. Press the ENTER key 5 more times until you get to the Valid Expression field. Here enter the following (this is a slider field so you can enter it directly without pressing the F2 key first):

```
SO.NUMBER<>''
```

This tells the program that the entry is valid if the value of **SO.NUMBER** is not blank (two quote marks together represent a blank value). If the entry is blank, then we want a message to be displayed. When you press the ENTER key the cursor will move to the Valid Message field. Since this is a message field like the Help Message you need to edit this field by pressing the F2 key. Then enter the following:

```
'You must enter a number here.  This cannot be blank.'
```

As with the Help Message press the ENTER key when you have entered the entire message.

The program will ask if all is correct. Press ENTER or Y and the full screen will be redisplayed. You can go back and change any entry by moving the cursor to any of the characters that represent the field and press the F3 key. You will be back to the field entry screen and can then move the cursor to the appropriate field and make the changes desired.

Next, we're going to have the date value default automatically to today's date. Press the F6 key and the cursor will move to the date field. Press the F3 key, and move the cursor to the Default field. Enter the following:

```
DATE()
```

This function returns the current system date. As part of the Default option if the date field has no value (00/00/00) then the program will automatically set it to todays date. You (or your user) can override this value but defaults should be the most likely entry. By using this option the ENTER key will be the only entry for this field most of the time.

Why don't you try adding the appropriate entries so that a date of some sort has to be entered? Hint: Using the Valid options, the expression would be:

```
SO.DATE<>0
```

When you're finished, press the F10 key (if you haven't gotten to the end automatically) and return to the main screen.

Now press the F6 key one more time. You should now be at the customer code field. Press the F3 key to edit the field information. We're going to show you another change you can make.

Move the cursor to the Additional LU Fld entry. Press the F2 key. Now type in **CUSTN**. By the time you get to N the field **CUSTNAME** will be at the top of the list (part of our **FAST SEARCH** technology!). Now press the ENTER key to accept the value and **CUSTNAME** will be placed in the field. Remember when you entered the first record and we told you to press the F2 key to get a list of customers? When that list was displayed you only saw the customer code. It is difficult to make sure you have the correct customer just by the code. By entering **CUSTNAME** here the customer name will be displayed as part of the list along with the code. Now you will be sure which customer you are choosing. Press the F10 key, press the ENTER key for Y and you're back at the main screen.

One last change. Press the F2 key to get the Option menu. Press the C key 3 times until the cursor is on the **Chg File Info** line. Now press the ENTER key. The familiar file information screen should be displayed. Press the ENTER key once and the cursor should now be in the Sort By field. Press the ^U key to clear the current value and press the ENTER key. Then press the ESC key to return, again, to the main screen. Press the ESC key again and the program will ask if you want to save the script. Press ENTER key or Y and the name will be redisplayed. Press the ENTER key again (we don't want to change the name, however, we could) and when the question to create a program comes up press the ENTER key again. Now, since the program SALES already exists the create routine will alert you to this. You must enter Y here to create a new source file. Press the ENTER key or Y to compile and after the compiler is finished (there should be no errors) press the ENTER key once more to return to the screen painter. Press ESC to return to the Main Menu and run the SALES program.

Notice that a list of keys (or indices) is displayed before any entry. Since we removed the Sort By value in the field information screen the program is now asking us which key to use. This won't effect data entry at all but will determine the order in which the orders will be displayed during a search and the value to search for.

After you choose a key the cursor should be at the Order Number field. Try pressing just the ENTER key. If you entered the Valid Expression and Message properly, the message should appear at the bottom of the screen. Press the ENTER key and the cursor will return to the field. Try pressing the F1 key and see what happens.

Enter a value and the cursor will move to the Date field. Notice how it is automatically set to the system date. Press the ENTER key to accept the default and the cursor will move to the Customer Code field.

> If a record is displayed after leaving the Date field you forgot to set the Auto Srch value to N for this field.

At the Customer Code press the F2 key to get a list of customers. Notice how much easier it is to determine which customer to choose when you can see the customer name also!

You may continue entering orders until you feel comfortable with the process. You'll also want more than just one or two records for the next part which is creating a report from our CUSTOMER and SALES databases. But, before we leave this part, notice how easy it was to create a real application, with multiple files, lookup lists, help messages, and entry verification steps without writing a single line of code! TAS Professional 5.0 did it all for you. What's even better is that as you get more proficient, if you desire, there's a full featured language behind these utilities to let you rapidly create applications you never dreamed possible. In fact, every utility you have used is written in TAS Professional 5.0, including TASEdit! This completes Part 4 of the tutorial.

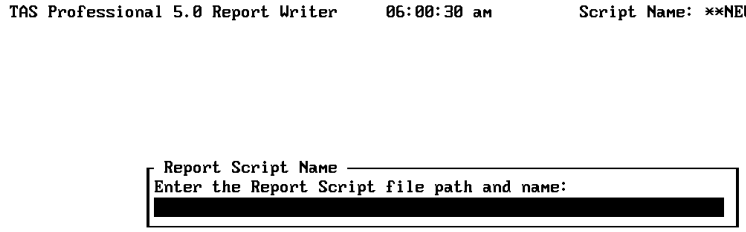
PART 5 - CREATING A TAS PROFESSIONAL 5.0 REPORT

In this section you will create a report showing sales and customer information. You'll be able to specify the sort field at run time and a range of dates or customers to include on the report.

To create reports with TAS Professional 5.0 you will probably always start with the **Edit Report Format** program (and the appropriate **Make Program** program). Together, these two programs make up the TAS Professional 5.0 Report Writer. This program will allow you to 'paint' the report on the screen similar to the Screen Painter. Most reports you might want can be created entirely using this method. However, for those situations where you need more, you can either 'include' other programs or modify the code created by the report writer.

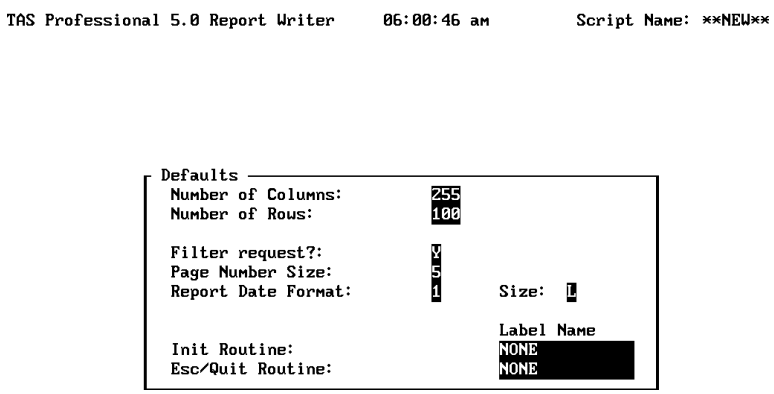
As with the screen painter, when you're finished with the report layout, you specify the primary file if there is more than one file, the sort by field (if desired), the select fields, and any file relationships, if the report will display information from more than one file. The **Report Format Make Program** then generates a report program based on the the information entered.

Let's get started. From the Main Menu select the Program topic menu. Press **R** to select the Report formats option, then press the ENTER key to select the Edit Report Format option.



Report Writer Opening Screen

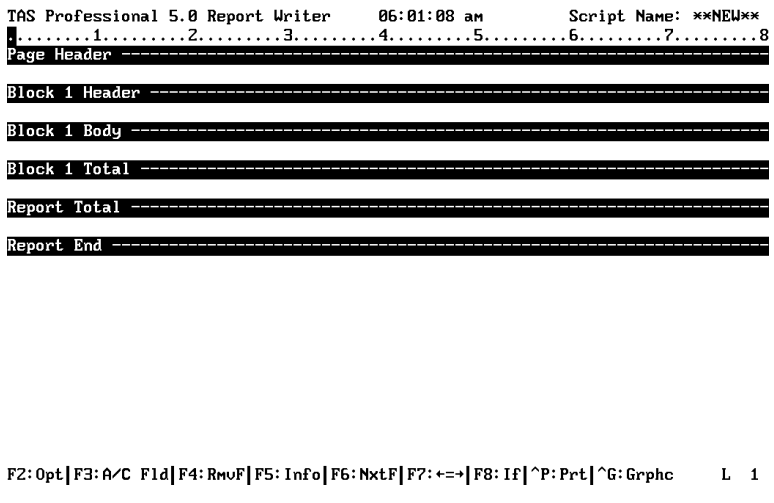
The first entry is the report format name. In this case we are creating a new format so just press the ENTER key.



Report Writer Defaults

The next screen displayed (shown above) allows you to enter general information about this report. For the purpose of this tutorial we aren't going to make any changes so press the ESC key.

The TAS Professional 5.0 Report Writer uses a 'block' or 'band' method of creating a report. You start out with 5 blocks or groups. They start with the first line after the heading and continue until the next heading.



Report Writer Main Screen

The five blocks or groups are described below:

Page Header - Everything in this group will be printed at the beginning of each page. This is generally the title of the report, page number, date, time, etc.

Block 1 Header - If there is only 1 block in the report this will print at the beginning of each page after the Page Header lines. If there are multiple blocks, this will print before each Block 1 body.

Block 1 Body - These should be the actual lines of data from the file.

Block 1 Total - This will print after all appropriate Block 1 Body lines have been printed.

Report Total - This will print only once at the end of the entire report. Generally you would use this to print number of records in the report, grand totals, etc.

Report End - Nothing can be placed after this line.

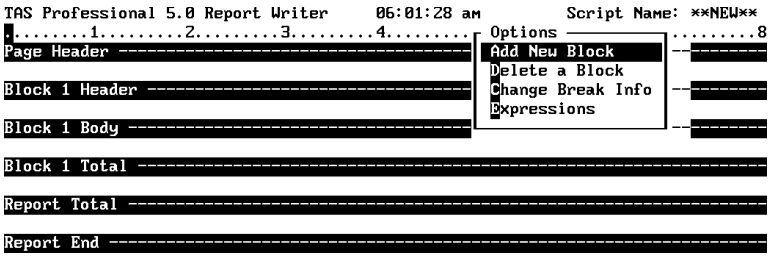
> You can have up to 9 sets of blocks or bands within one report format. Each one has Header, Body, and Total sections. They can be nested or can follow one another. In this report format we will use two.

As with the screen painter the screen displayed is basically a blank canvas. You can put anything you want on it. Across the top of the screen is a ruler marked off in 10 character increments. The 1 specifies the 10th character (the far left hand character is number 1). Across the bottom of the screen are the function keys available. They are similar to the ones described for the screen painter. If you need to please refer to that section for more information. The ones that are different are:

F7 - Center all the data on the current line within the number of columns.

F8 - Enter an If Print expression. This will allow you to control whether or not an entire section (if the If Print is entered while the cursor is on a section header line) or just an individual line is printed when the expression entered is evaluated. For the purposes of this tutorial we will not be using this option.

As with the screen painter an Option Menu appears if you press the F2 key. However, a different menu will be displayed depending on whether the cursor is on a section header line or a regular print line. We will first look at the options available when on a section header line.



Option list Header Line

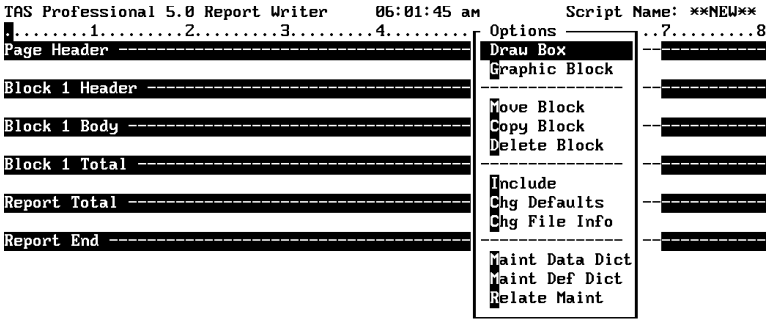
Add a new block - This will add the lines for a new block. Each new block contains a Header, Body, and Total section. It will be inserted directly before the cursor location. When you add a new block you will be asked to enter certain information. This includes the Break Field, i.e., when the value of this field changes we know we've displayed all the records we need to for this block, the Primary File for the block, and whether or not to do a page feed (go to new page) after the Total lines (if any) have been printed for this block.

Delete a block - This will remove the block lines, and all text and fields within them.

Change Break Info - You can change the information entered when you added a block: Break Field, Primary File, New Page.

Expressions - This option will allow you to enter calculations that will be executed at the beginning or end of the section. This would be used when you want to total a field value or count the number of records being printed.

If you are on a regular print line (not a section header) the following option menu is displayed. These options are similar to the ones in the Screen Painter; however, you may copy a block here since a report format may have duplicated fields.



Option list Format Line

Movement around the report format screen is also similar to the screen painter. However, since the format can be wider than the screen itself, there are two extra keys. These are the ^HOME and ^END. ^END goes to the extreme end of a line (which may be character position 255) and ^HOME goes to character 1, no matter where you are on the format.

Why don't you try moving the cursor around and familiarize yourself with movement in this program?

Now enter the report format. First you need to create some extra lines. Move the cursor to line 2. Press the ^A (add lines) key 2 times. This will make a total of 3 blank lines between the Page Header and Block 1 Header section lines. Now move to line 6 and press the ^A key 1 time. Move to line 9 and press the ^A key 4 times (5 lines in the Body section). Move to line 15 and press the ^D key (delete line) once; there won't be any Body 1 Total lines. Now move to line 16 and press the ^A key once. The blocks are now all set up for our entries. We didn't have to do this all at the same time: you can add and delete lines as desired while you're creating the format.

Using the information from the table below, place the fields and constant text onto the report format. This is the same as the screen painter except for the block section lines. When you place the fields **PAGE_NUM**, **RPT_DATE** and **RPT_TIME** you will notice that the FD name is **rptwrtr**. These fields are already pre-defined by the report writer and can be used without any programming. By just placing them on the format you will be able to print the page number, date and time for this report.

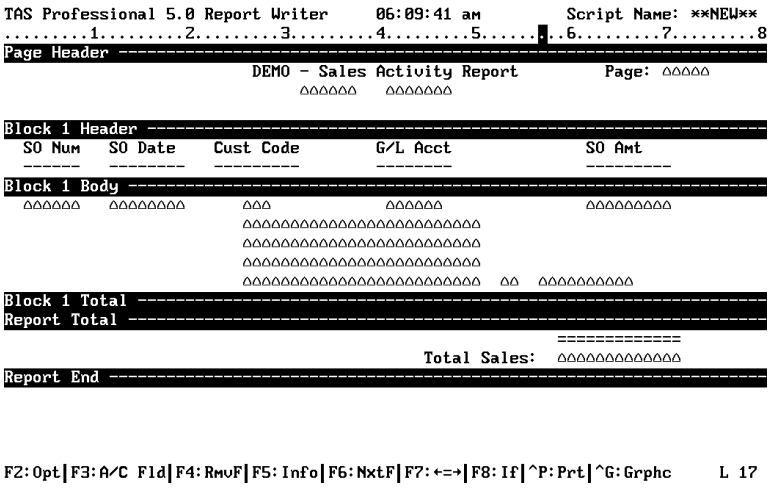
> You will probably realize that you haven't defined **TOT_SALES** yet, but when you place it on the format it knows what to do. This is because we defined it in the defined data dictionary already for you. When you place a field the program looks for it first in the regular data dictionary and then in the defined data dictionary. Through the use of this feature many fields that you will use again and again in programs, but aren't a part of any file, can be defined once and used whenever necessary.

The following are all the fields and text that need to be placed on the report format.

FIELD	COL	LINE	FIELD	COL	LINE
PAGE_NUM	70	2	RPT_DATE	32	3
RPT_TIME	41	3	SO.NUMBER	3	9
SO.DATE	12	9	SO.CUSTCODE	26	9
SO.GL.ACCTNUM	41	9	SO.SALES.AMT	62	9
CUSTNAME	26	10	CUSTCOMP	26	11
CUSTADDR	26	12	CUSTCITY	26	13
CUSTSTATE	53	13	CUSTZIP	57	13
TOT_SALES	59	17			

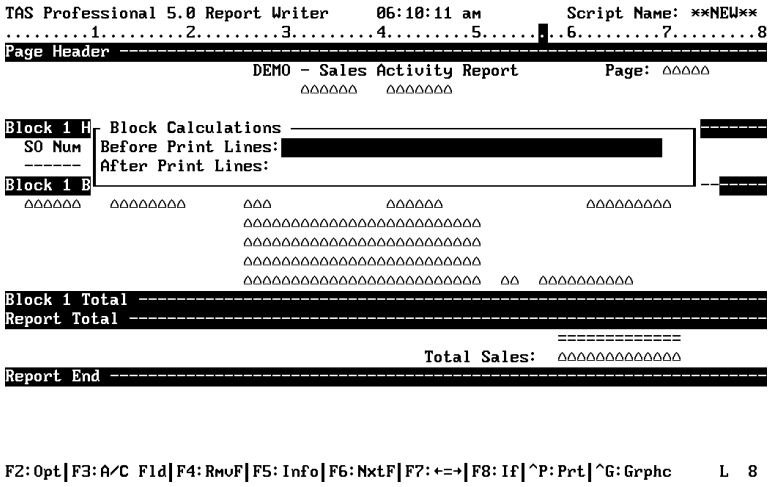
LITERAL TEXT	COL	LINE
DEMO - Sales Activity Report	27	2
Page:	64	2
SO Num	3	6
———	3	7
SO Date	12	6
————	12	7
Cust Code	23	6
————	23	7
G/L Acct	40	6
————	40	7
SO Amt	62	6
—————	62	7
==========	59	16
Total Sales:	45	17

When you're finished the screen should look like this:



Sales Order Report Format

Now put in the calculation that will give us the total sales figure at the end of the report. Move the cursor to line 8, the Block 1 Body section line. Press the F2 key to get the option menu and press E for Expressions. Press the ENTER key.



Expression Entry

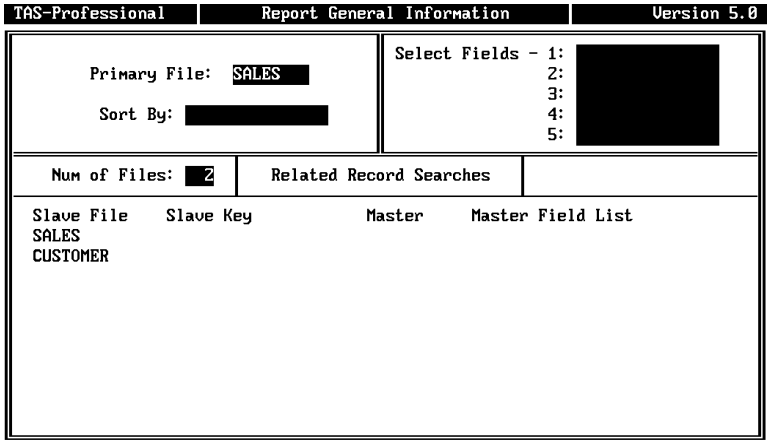
Press the ENTER key to move to the After Print Lines entry. We want to make sure that the record has been read before we total the sale figure. Now enter the following:

```
TOT_SALES=TOT_SALES+SO.SALES.AMT
```

After you press the ENTER key the program will return to the main screen. Now each time after a body record is read the amount of **SO.SALES.AMT** will be added into the **TOT_SALES** figure so that it will print properly at the end of the report.

> We don't have to worry about the beginning value of **TOT_SALES** since all numeric values are preset to 0 at the beginning of a program.

Now press the ESC key and we'll enter the file information. Enter Y to save the script, and then enter the format name, **SOREPORT**. The file information screen will then be displayed:



Report Writer File Info Screen

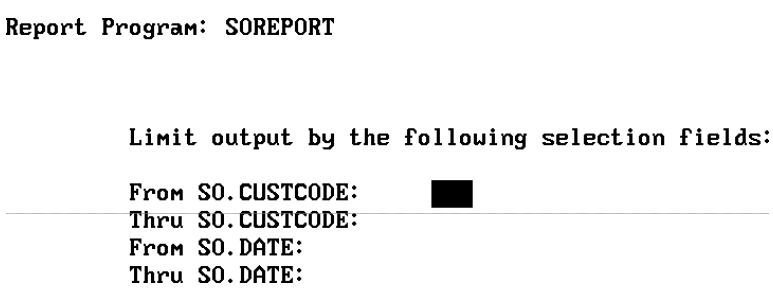
The primary file is already set to **SALES** (if it isn't enter that file name) and leave the Sort By field blank (we want to choose the index at runtime). You should now be in the first field of the Select Fields block. You can put up to 5 different field names here that will be used as From/Thru limits within the report. In the first block enter the field name **SO.CUSTCODE** (we're using this field instead of **CUSTCODE** since

the primary file is **SALES**). For the next value enter **SO.DATE**. This will allow us to enter the beginning and ending customer codes and sales dates. When the report is run, if we leave these values blank it will print all the records available. Press just the ENTER key at the third line and the cursor should move to the Related Record Searches block.

This is the same process as in the screen painter. Move the cursor down one line, press the ENTER key. The Slave Key is **CUSTCODE**, the Master is **SALES**, and the Master Field List is **SO.CUSTCODE**.

> You'll notice that that you are not asked whether you want to update the master fields as you were in the screen painter. This program doesn't save any records or create anything new. It just prints out information currently in the file. That's the reason for the difference.

Press ESC after entering the file information and the program will return to the main screen; save the report format; and ask if you want to create a program. Answer Y for the create and compile questions. Press the ENTER key after the compilation is finishedand ESC when you return to the Report Writer. Run the SOREPORT program from the Main Menu.



Sales Report Initial Screen

The first time through the report just press the ENTER key for all From/Thru values. This will allow all records to print. After you enter the From/Thru values the program will ask you to 'Enter the desired record filter'. For right now just press the ENTER key again.

Next the program will display the available keys; choose one or just press the ENTER key to choose the **SO.NUMBER** key.

The program will now ask if you want to print to the Screen, Printer or Disk. Press the ENTER key to accept printing to the screen.

> TAS Professional 5.0 can output reports to any of these three devices and even multiple printers all without any additional programming on your part.

Your output should look something like the example shown on the next page depending on the data you entered.

```
                        DEMO - Sales Activity Report            Page: 1
                             09/14/92 09:30 a

SO Num   SO Date    Cust Code        G/L Acct             SO Amt


1005     09/13/92      001            101                   451.25
                Abbot, Larry R.
                Bug-B-Gone Exterminators
                13145 Ant Street
                Tacoma                     WA  98422
1135     09/14/92      002            123                   149.95
                William, Ralph V.
                Pro Musica
                202 Maple Lane
                Bridgeport                 CT  01255
1234     09/14/92      003            108                   150.45
                Larson, David
                Little Greenhouse
                702 Monroe Ave.
                Monroe                     WA  98720
                                                        =============
                                       Total Sales:         751.65
```

Sales Order Report Output

You might want to try modifying the report format so that an extra line will print between each body record. To do this add a line after the last body record (^A) but before the Block 1 Total section line. Create the program again, recompile and rerun.

A more complex report

A problem could occur very quickly with the report above. What would happen if there were multiple sales for each customer? Obviously, you're going to print many more lines than you want and you won't have a total for each customer. We have already created a program for you that will take care of the problem. Rather than using the SALES file as primary it uses the CUSTOMER file. It also has a second block for the sales records. The format name is SORPT2.SRP and is located in the SAMPLE sub-directory (off the TAS40 directory). We copied this file to this sub-directory at the beginning of the tutorial. We have provided you with just the script file so if you want to run the program you need to load it into the Report Format edit program, save it, create a program and compile it.

Output from this report might look something like this (depending on the data you entered):

```
                DEMO - Sales Activity Report 2      Page:     1
                        09/14/92 10:47 a


  Cust Code       Customer Info
  ---------       -------------

    001           Abbot, Larry R.
                  Bug-B-Gone Exterminators
                  13145 Ant Street
                  Tacoma
                  WA  98422      (206) 555-1214

                  SO Num   Date       G/L Acct  Amount
                  ------   ----       --------  ------
                  1005     09/13/92    101       451.25
                  1567     09/14/92    123        39.34
                        Total Sales For Cust:    490.59
```

```
Cust Code       Customer Info
─────────       ─────────────
  002           William, Ralph V.
                Pro Musica
                202 Maple Lane
                Bridgeport
                CT  01255       (217) 444-4321

                SO Num   Date       G/L Acct  Amount
                ──────   ────       ────────  ──────
                1135     09/14/92    123       149.95
                      Total Sales For Cust:    149.95

Cust Code       Customer Info
─────────       ─────────────
  003           Larson, David
                Little Greenhouse
                702 Monroe Ave.
                Monroe
                WA  98720       (206) 234-8765

                SO Num   Date       G/L Acct  Amount
                ──────   ────       ────────  ──────
                1234     09/14/92    108       150.45
                1765     09/14/92    223        75.94
                      Total Sales For Cust:    226.39

                                             ==========
                        Grand Total:          866.93
```

Sales Order Report2 Output

This should give you a feeling for what can be done simply and easily with the TAS Professional 5.0 Report Writer. This completes Part 5 of the tutorial.

PART 6 - CREATING A TAS PROFESSIONAL 5.0 MENU

The backbone of an application is its menu. The program's user-friendliness will be judged partly by users' ease in operating the application using the menus provided. TAS Professional 5.0 has a sophisticated menu command designed to give you maximum flexibility in providing your applications with the kind of pull-down menus popular in today's computer software.

Programming graphics-oriented menus like the ones that TAS Professional 5.0 produces used to take days. You'll see in this example that it now takes minutes. In this part, you will create a menu to call the programs that you have created in previous parts. This part will also introduce you to the Program Editor.

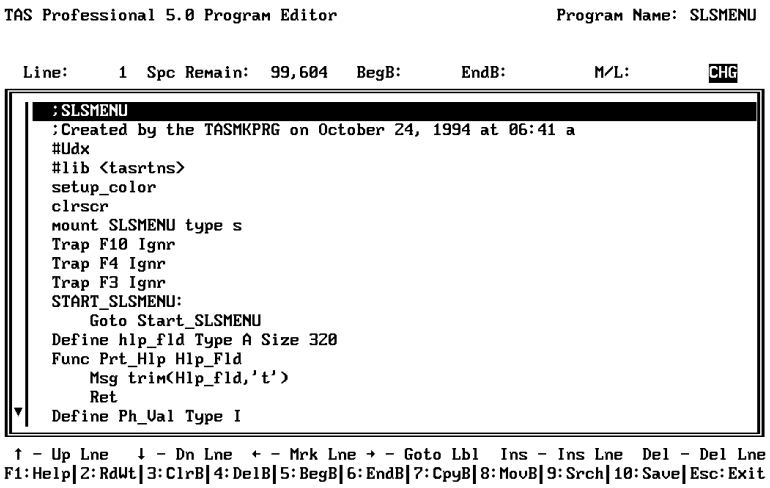
From the Main Menu choose the Program sub-menu and the Edit Screen Format program. We're going to enter a new format so just press the ENTER key at the format name. At the Defaults screen again press the ESC key. Now we're going to put a little design on the screen. You can make this anything you wish. It will act as the backdrop for the menu.

With the cursor in the upper left hand corner press the F2 key. Enter G for Graphic Block and press the ENTER key to choose this option. Press the ENTER key again (after reading the message) and then press the ^PG DOWN key to go to the extreme far lower right hand corner (we're going to fill the entire screen with graphic characters). Press the ENTER key and a list of graphic characters will be displayed. Press the

PG DOWN and/or DOWN ARROW key until the cursor is at number 177. Press the ENTER key to choose that character (or a different one if you desire). Enter Y to fill in the block and the cursor will be returned to the upper left hand corner. Now, staying on the first line move the cursor by pressing the TAB key to column 30 and enter the following:

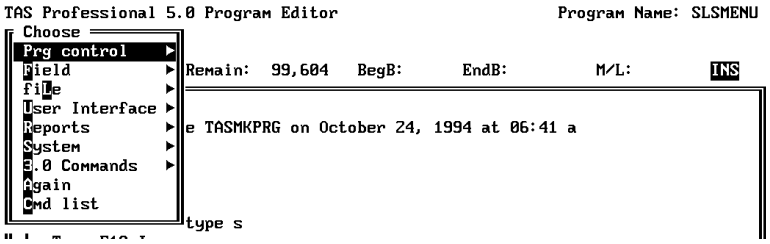
```
DEMO - Sales Order Application
```

This will clear the graphic characters you entered but only in that location. It will give the look of being highlighted. Now press the F10 key and save the screen format. Use the file name SLSMENU and create a program from it but don't compile this time. Return to the Main Menu and choose the Edit Program option (4 lines below the Screen Painter). Enter the file name SLSMENU and press the ENTER key. Your screen should now match the one below (except for the date the program was created).



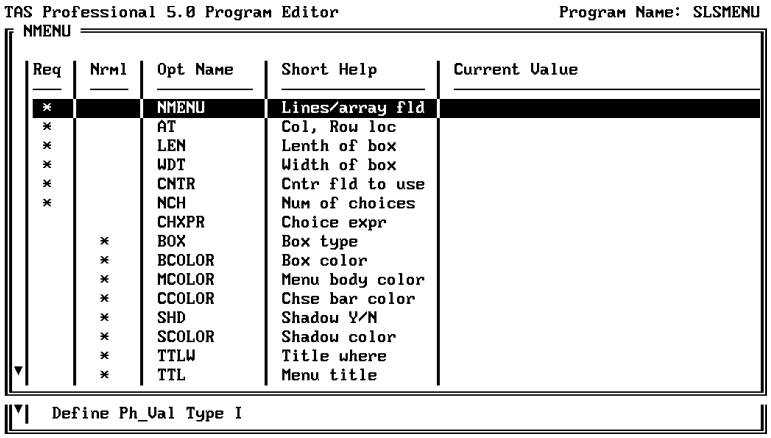
Edit Program with SLSMENU.SRC Displayed

Move the cursor bar until it is just after the line that says START_SLSMENU: You can also click on the line directly if you have a mouse installed. Press the INSERT key and the cursor should change to a small block. Now press the ENTER key and a choose menu will appear. The following should be displayed on your screen.



Edit Program Menu Command

At the Choose menu, enter U for User Interface, and 5 for 5.0 Nmenu. A list with the options for this command are displayed. Press the ENTER key to start entering values with the first option. You can also move the cursor bar to any other option or click on that line with the mouse.

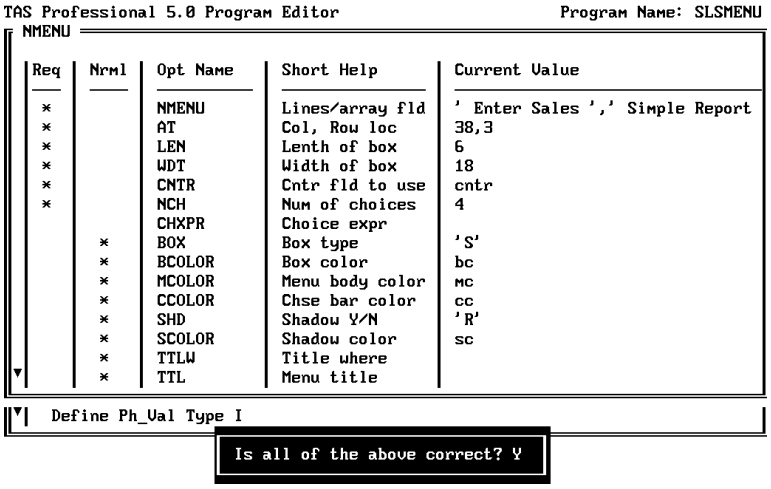


Press the ENTER key with the cursor bar on the first line and enter (the entry fields that allow for larger values that can be displayed are sliding fields):

```
' Enter Sales ',' Simple Report ',' Complex Report ',' Quit to DOS'
```

Don't forget the quote marks and commas between the values. Press the ENTER key again when you have completed the entry and you will move automatically to the next line.

Then the At entry; **38,3**. Len (length) is **6**; Wdt (width) is **18**; Cntr (counter) is **CNTR**; NCH (number of choices) is **4**; no entry at CHXPR (choice expression) just press the ENTER key; box is **'S'** (single line); BCOLOR (box color) is **BC**; MCOLOR (menu color) is **MC**; CCOLOR (choice color) is **CC**; SHD (shadow) is **'R'** (right); SCOLOR (shadow color) is **SC**; keep pressing the ENTER key until the cursor gets to the Auto field. Enter **Y** here. This tells the program to immediately exit the menu command after the user presses an acceptable choice character instead of just moving the to the line and waiting for the ENTER key. Now press F10 to save the command as is. You need not make any more entries. The screen should look like this:



Program Editor with Nmenu Command Listed

Enter Y to save the command. The command will be inserted into the program directly before the line you were on and you will be returned to the Choose menu, ready to enter another command. The command you

just entered will create the menu on the screen. You told it what to put on each line, how big it is to be and what colors to use.

> The colors BC, MC, CC and SC are all part of the file TASCOLOR.OVL. By accepting the defaults at the beginning of the screen painter we can use these fields instead of specifying exact values. Then all you have to do is change the values in that file and the colors will change also.

Press the U key for User Interface, then the W key for Windows and then S for Save Screen. You need not make any other entry; the program will save the screen to an internal buffer. Press F10 and then Y to save the command.

Press the ENTER key for Prg Control, then the ENTER key again for CASE and one more time for Select. Enter **CNTR** and then F10. Enter Y to save the command.

Now you need to enter information telling your application which programs correspond to which menu choices. Press the ENTER key for Prg Control again, then CASE again, then C for the actual CASE command. Enter **1** and then F10, then Y to save the command.

Press the S key for System, then the ENTER key for Other Programs, then C for Chain. Enter **'SALES'**, then F10, and Y. This tells the program to chain to the sales program when you choose the first option on the menu.

Press the ENTER key for Prg Control again, then CASE again, then C for the actual CASE command. Enter **2** and then F10, then Y to save the command.

Press the S key for System, then the ENTER key for Other Programs, then C for Chain. Enter **'SOREPORT'**, then F10, and Y. This tells the program to chain to the simple sales report program when you choose the second option on the menu.

Press the ENTER key for Prg Control again, then CASE again, then C for the actual CASE command. Enter **3** and then F10, then Y to save the command.

Press the S key for System, then the ENTER key for Other Programs, then C for Chain. Enter **'SORPT2'**, then F10, and Y. This tells the program to chain to the more complex sales report program when you choose the third option on the menu.

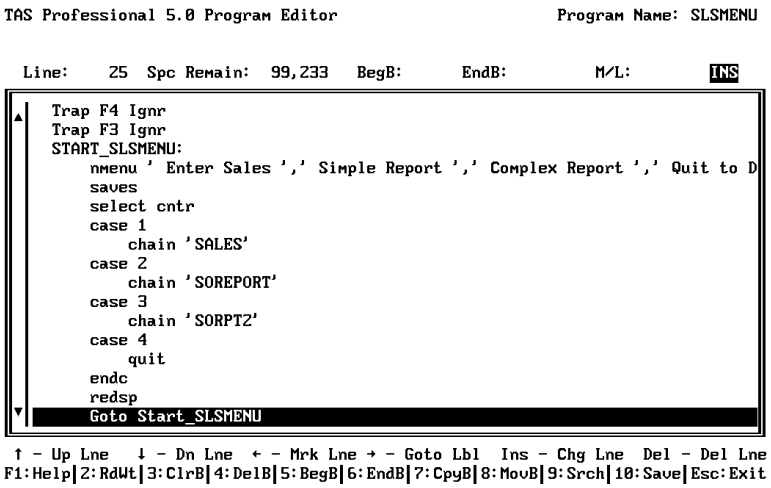
Press the ENTER key for Prg Control again, then CASE again, then C for the actual CASE command. Enter **4** and then F10, then Y to save the command.

Press the S key for System, then the Q key for QUIT command, F10 and Y. This tells the program to quit this menu and return to DOS or the previous program if the user chooses this option.

Press the ENTER key for Prg Control again, then CASE again, then E for the ENDCASE command, F10, then Y to save the command. This is the end of the choices possible.

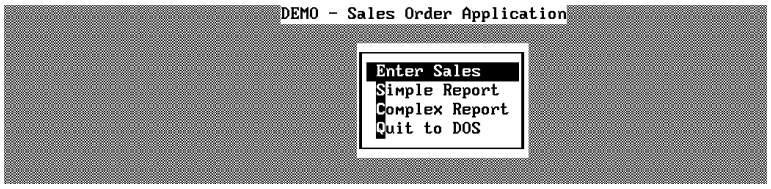
Press the U key for User Interface, then the W key for Windows and then R for Redisplay Screen. You need not make any other entry; the program will restore the screen from the internal buffer it was saved to above. Press F10 and then Y to save the command.

We now have our program. Press the ESC key and the following should be displayed.



Program Editor with Finished Program

Press the ESC key again, enter Y, then press just the ENTER key at the file name. Press ENTER for the compile option and the program should compile without problems. Press ESC to return to the Main Menu and run the SLSMENU program. The following should be displayed.



Menu Program

Press the E key and the order entry program will be run. Press ESC to quit that program and you return to the menu. Press the other keys and the appropriate program for each will be run. You can go back and make changes to your screen at any time. Just go back into the screen painter and either make changes to the screen directly or to the code attached to it.

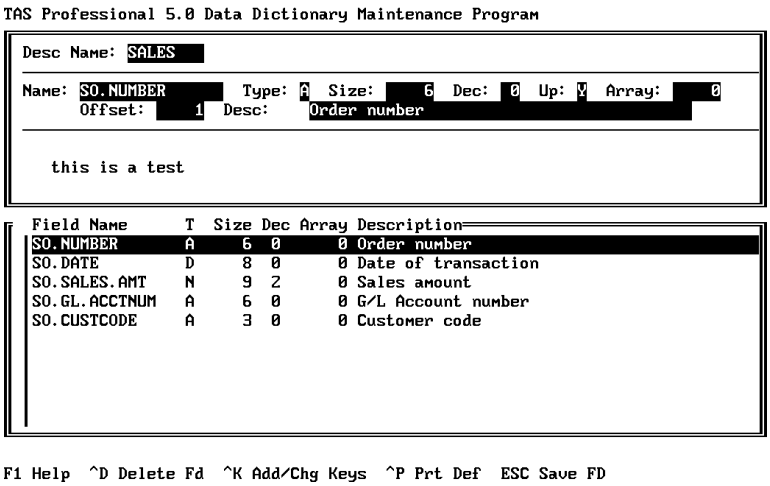
You now have a full application with a menu and two printing routines. This completes Part 6 of the tutorial.

PART 7 - MODIFYING AN EXISTING DATA FILE

Some times it is necessary to modify a data file that contains active records. In this section we will add a field, Part Number, to the Sales file. We will use **Maintain Data Dictionary** to add the field and the Restructure utility to actually modify the data file.

Maintaining the Data Dictionary

Select the Database topic menu from the Main Menu and press the ENTER key for **Maintain Data Dictionary**. Enter **SALES** as the FD name.



Maintain Sales File Layout

The file layout for the Sales file is shown above. Press the ^END (CTRL+END) key. This is like pressing the END key to get to the end of the list and then the DN ARROW key to move to a blank line and then the ENTER key. The cursor will now be in the entry section. Enter the following information for the new field:

Field Name:	**SO.PART.NUM**
Field Type:	**A**
Field Size:	**15**
Field Dec:	**0**
Upcase:	**Y**
Array:	**0**

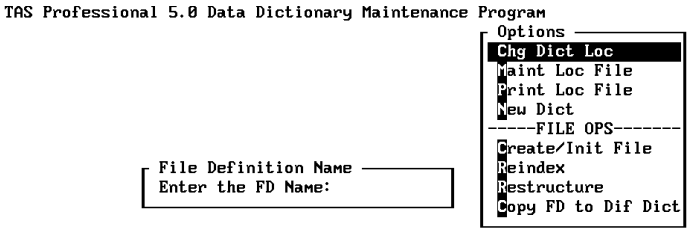
Press **Y** at the "Is all of the above correct" question. After you have entered the field press ESC to save the FD. Press **Y** at the "Is all of the above correct" question to actually update the file layout with the new field. The program will return to the initial screen.

Restructuring a Data File

After you have modified the file layout you can restructure the data file. Use this instead of the Initialize routine when there is data in the file you want to keep. This process will make the changes to the actual data that you have made to the FD. You can use the Restructure option to add fields, remove fields, make them smaller, larger, etc. It can even add or subtract array elements. The only thing Restructure can't do is change a field from one type to another. If there were no active records in the data file, you should use the Initialize option on the Database menu. If all you do is make changes to the keys (indexes) then you can use the Reindex option instead.

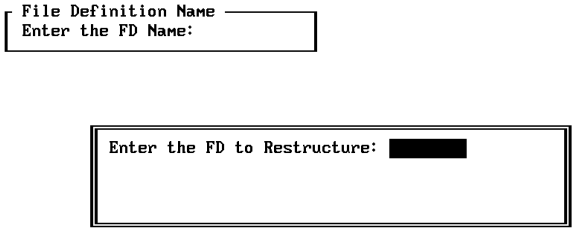
> In normal operation it is wise to make a backup of your data before you restructure the file. What is going to take place is on a par with major surgery. If anything goes wrong you will not be able to recover your data without that backup. In this case we won't because this is only test data.

Press the ^O key to get a listing of dictionary options. You should still be at the initial screen from the process above. If you are not please run the Maintain Data Dictionary program again.



Maintain Data Dictionary Options

Move the cursor to the Restructure option by pressing the DN ARROW key or pressing the R key twice (the first time it will go to the Reindex option). Press the ENTER key, a warning message will be displayed, enter Y and the following will be displayed.



Maintain Data Dictionary, Restructure Option

Enter the FD name SALES. The program will display a declining number as the records are written out to a fixed length file. Then the regular file is initialized, and the records are then read back in. The time it takes to do this is directly dependent on the size of the file.

> In this situation there was only one file using the SALES FD, so the Restructure routine will process all files using the same FD within the same data dictionary. This will make sure that all files have been updated properly.

Modifying the Sales Entry Program

After the data file has been modified and restructured, all programs that access that file must be updated to include the new data field. There are two programs created in this tutorial that access the Sales file: SALES - Enter Sales, and SOREPORT - Print Sales Activity.

To update the Sales entry program to include the new data field run the Screen Painter program and enter SALES as the script name. The familiar format will be displayed. Using the cursor keys position the cursor to column 12 and line 14. Enter **Part Number:** as the field prompt. Enter a space (press the space bar) and press **F3** (A/C Fld) to add the new field (**SO.PART.NUM**) on the screen format. You might want to add a help message or a validation process. It's up to you.

Press **ESC** to exit the Screen Painter and recreate the program. Don't forget: you'll have to enter Y to the overwrite question in the Create Program routine or it will not create the source code. Compile and run it. You will see that the field will be blank for the data currently there. You might want to call up the current records and add the inventory information or add new ones. The **SALES** program has now been updated to accommodate the new data.

Now, with what you know, you can go to the SOREPORT or SORPT2 program and make similar changes. If you don't you should at least recompile them. This can be accomplished by choosing the Compile Program option in the Program sub-menu. Enter the program name (no extension required) and the compiler will be called automatically. If any errors occur you will be notified. Otherwise the program will return to the Program sub-menu when completed.

As an exercise you may want to create a part number file and do a lookup when entering a sales order so you cannot enter an erroneous part number. The part number file might contain a description for the part number, the current selling price, the last sale date and the total sales to date.

Summary

As you can see from this tutorial, TAS Professional 5.0 has the flexibility to be used as a simple data manager and the power to build sophisticated applications. With the utilities available we think you will agree that TAS Professional 5.0 is truly the Professional's tool for today's demanding data management and application development requirements.

TAS
PROFESSIONAL
VERSION 5.1

Chapter 3

Main Menu Programs

INTENTIONALLY BLANK

INTRODUCTION

NOTE: Each of these programs makes substantial use of a feature called the List Window. Each of these works in very much the same way. You may move around the list window in several different ways. These are:

Up arrow — Will move 'up' the list one line. If there are lines 'above' the window an arrow will appear on the far upper left side of the window, between the vertical bar and the left edge. Each time the use presses the UP ARROW key the lines in the window will be moved down one and a previous line will be displayed. When there are no more lines above the window the arrow will be cleared and the program will sound the bell when the UP ARROW key is pressed.

Down arrow — Will move 'down' the list one line. If there are lines 'below' the window an arrow will appear on the far lower left side of the window, between the vertical bar and the left edge. Each time the user presses the DOWN ARROW key the lines in the window will be moved up one and a new line will be displayed. When there are no more lines below the window the arrow will be cleared and the program will sound the bell when the DOWN ARROW key is pressed.

In some instances you will be able to add new lines to the list. In these situations you will be able to press the DOWN ARROW key until a blank line is displayed. If the Enter key is pressed in this situation you will be able to add a new line at that location. (For more information please see the Add line description below.)

Enter character — You can enter the first character of the desired line and the program will move the cursor to that line. If the line is above or below the window it will be moved into the display space. You can move to the next instance by continuing to enter the same character. When the last line has been reached that matches the character entered, the program will start over with the first instance. If you enter a character which has no match the program will sound the bell.

Some of the list windows also use the FAST SEARCH™ technology. This means you can look up an entry by typing in the characters of the particular field. That is, in all programs when searching for a field name you can start to enter the first characters of that name. The program will find the record that most closely matches the characters entered and display that record at the top of the window. You can return to the beginning of the list by pressing the ^U (CTRL+U) key or backup one character by pressing the BACKSPACE key.

Page Up — Pressing this key will move the cursor up 1 window's worth of lines. This will only work if the up arrow character is displayed. Once the first line in the list is displayed at the top of the window this key will have no more effect.

Page Down — Pressing this key will move the cursor down 1 window's worth of lines. This will only work if the down arrow character is displayed. Once the last line in the list is displayed at the bottom of the window this key will have no more effect.

Home — If you press this key the first line in the list will be displayed at the top of the window and the cursor will be placed on that line.

End — If you press this key the last line in the list will be displayed at the bottom of the window and the cursor will be placed on that line.

Insert — In some instances you will be able to add lines to the list. One way to do this is by pressing the Insert key. The entry screen will be displayed and you will be able to

enter the new values. If everything is entered correctly and you answer the questions properly the new line will be entered immediately above the cursor location. The cursor will remain on the same line and the lines above will be moved up 1.

Delete — If you are able to make changes to the lines in the list you will be able to delete lines from the list. The way to do this is by pressing the Delete key. The entry screen will be displayed and you will be able to confirm the deletion. If you answer the questions properly the line will be deleted.

NOTE: You need to make sure that the line should be deleted since the program may physically delete the line from the disk at the time the confirmation is entered.

Enter — If you are able to make changes to the lines in the list this is the way to do just that. The program will display the appropriate screen and you will be able to make the changes desired.

Add line — Another way to add new lines is by moving to the end of the list (either through the use of the DOWN ARROW or END key) and then pressing the DOWN ARROW key once. A blank line will be displayed. When the ENTER key is pressed the program will display the appropriate screen and you will be allowed to enter a new line. If everything is entered properly and you confirm the entry the program will display the new line at the end of the window and the cursor will remain on that line. To add another new line press the down arrow key, then ENTER, etc. In some instances the program will remain in the add line loop without returning to the window until you press the Esc key or answer N to the confirm question.

Esc — This will cause the program to leave the window. You may be able to confirm that changes made were acceptable and should be saved to the appropriate files.

F2 and F3 — In some instances you will be able to execute some option by pressing either of these keys. The options, if available, will be displayed at the bottom of the screen.

TASEdit™

We have also included a runtime version of TASEdit. This is a small text editor (with automatic word wrap) that we have written entirely in TAS Pro 5.1. You may edit a file up to 64k in size with it; however, it was designed to handle smaller tasks, such as memo fields, editing small files, etc.

To move around within TASEdit use the following keys:

Rt Arrow or Ctrl_D	Move right 1 character
Lt Arrow or Ctrl_S	Move left 1 character
Ctrl_Rt Arrow or Ctrl_F	Move right 1 word
Ctrl_Lt Arrow or Ctrl_A	Move left 1 word
Dn Arrow or Ctrl_X	Move down 1 line
Up Arrow or Ctrl_E	Move up 1 line
Home	Goto beginning of line
End	Goto end of line
Ctrl_Home	Goto beginning of window
Ctrl_End	Goto end of window
Pg Up or Ctrl_R	Move up 1 window's number of rows
Pg Dn or Ctrl_C	Move dn 1 window's number of rows
Ctrl_Pg Up or Ctrl_QR	Goto beginning of field
Ctrl_Pg Dn or Ctrl_QC	Goto end of field
Ins	Insert toggle on and off

Backspace	Backspace 1 character (delete chr to left of cursor)
Del	Delete chr at cursor
F5 or Ctrl_KB	Block start
F6 or Ctrl_KK	Block end
F3 or Alt_U	Turn off block
F4 or Ctrl_KY	Delete marked block
F7 or Ctrl_KC	Copy marked block to cursor location
F8 or Ctrl_KV	Move marked block to cursor location
F9 or Ctrl_QF	Search for string
F10 or Ctrl_QA	Search for string and replace
ESC	Quit editor and return to calling program

NOTE: TASEdit cannot be run stand-alone, but must be called from a TAS Pro 5.1 program properly set up to do so.

MOUSE CONTROL

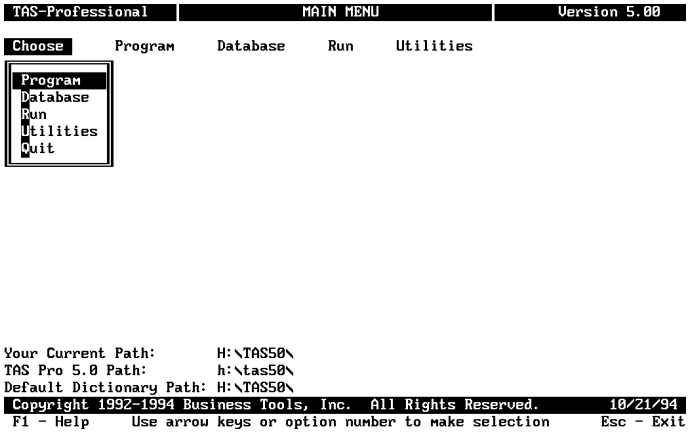
If you have a mouse attached to your computer and load the appropriate driver in DOS you can use it with your TAS Professional 5.1 programs. We have tried to keep all actions similar so that no matter whether you are in a list box, the editor or a entry field, all actions are similar. The following will give you an idea of how the mouse works in each of the following situations. At any point you can just press the buttons or move the mouse cursor. If nothing is set up for that particular situation then nothing will happen. NOTE: For information about turning the mouse on automatically when a program is run please see the **Set Configuration (U-G)** option in the **Utilities** sub-menu.

MENU	MOVE the mouse to the item you wish to choose. LEFT BUTTON - chooses the line being pointed to. RIGHT BUTTON - same effect as pressing the ESC key.
LIST BOXES	MOVE the mouse to the record/line you wish to choose. You can also move the mouse to the arrows (up/down or left/right) or the top or bottom of the box itself. LEFT BUTTON - When the mouse pointer is on a record/line this is the same as moving the choice line using the UP or DN ARROW keys. If the mouse pointer is on the current line (where the choice bar is) this is the equivalent of pressing the ENTER key. When the mouse is on the top line of the box it is the same as the PG_UP key, when on the bottom line of the box the PG_DN key. When the mouse pointer is on either the up or down arrows (if they are displayed) and the button is held down the program will scroll through the lines up or down as appropriate until the button is released. When on the left or right arrows (at the bottom of the box when displayed) then list will scroll left or right as appropriate, one move for each click. RIGHT BUTTON - When the mouse is within the box (not on one of the arrows) this is the same as pressing the ESC key. When the mouse is on the top line of the box it is equivalent to pressing the ^HOME (CTRL+HOME) key or go to first line in list. At the bottom of the box it's a ^END (CTRL+END) key or go to last line in list. NOTE: To add a new entry to the end of a list when the down arrow is displayed, move the mouse pointer to the box line below

the down arrow and press the right button, this will take you to the end of the list. Then move the mouse to the line just above the box (where the down arrow was) and press the left button. The list will move up one more line and you can now click the left button on the blank line to choose it.

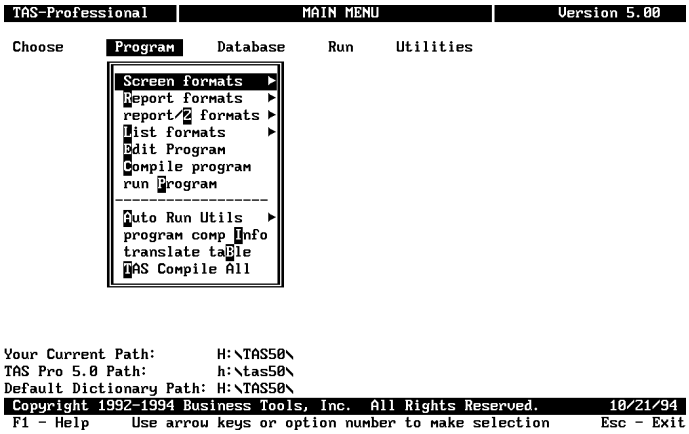
General Entry

MOVE the mouse pointer to the field you wish to enter and press the LEFT BUTTON. If the program will allow you to enter the field at that time the cursor will move to that field. You can also move the mouse within a chosen field and press the LEFT BUTTON and the cursor will move to that character location if possible. If the mouse pointer is at the same location as the cursor and you press the LEFT BUTTON this is the same as pressing the ENTER key. The RIGHT BUTTON is equivalent to pressing the ESC key.



CHOOSE

The first menu displayed when you run TAS Professional 5.1 is the Choose menu. This menu allows you to move directly to another menu or to choose to exit to DOS (QUIT). You can also move from one menu to the next by pressing the LEFT or RIGHT ARROW keys.



PROGRAM

This is the menu for the main programming tools.

Screen formats -> Edit screen format

You reach this option by placing the cursor on the **Screen formats** option line and pressing the ENTER key or by pressing the **S** key. Then press the ENTER key again to run the program.

Program name: TASEDSCR.RUN

This program creates and changes screen ‘scripts’. A script is a specialized format type that keeps all the information about that screen format but cannot actually be run.

You may edit and create a new program as many times as you desire from a screen format script. In fact, you may use a script as the basis for a new format by placing the name of the appropriate script as the format to be changed at the initial prompt and then saving it under a different name when you are finished.

All script files created from this program have the extension .SCP.

Edit screen format

Control Keys

> **NOTE:** When the character ‘^’ precedes the key name it means to press the CTRL key at the same time you press the other key.

Dn arrow - move down one line

Up arrow - move up one line

Pg Dn - move down to lower right hand corner of current screen.

^Pg Dn - move to the far lower corner of the screen format.

Pg Up - move to upper left hand corner of current screen.

^Pg Up - move to the far upper left hand corner of the screen format.

End - move to end of current line

Home - move to beginning of current line.

Tab - move to the right 5 spaces.

Shift-Tab - (Back Tab) - move to the left 5 spaces.

^A - Add a line to the format at the cursor position. Any characters on the last line will be lost.

^D - Delete a format line at the cursor position. Any characters on that line (including fields) will be removed.

F2 - Display the option menu (see options below).

F3 - Add a new field spec or change a current one (to change one the cursor has to be placed somewhere on the field marker).

F4 - Remove a current field from the format (the cursor has to be placed somewhere on the field marker).

F5 - Display (and/or print) the information about a field in the format.

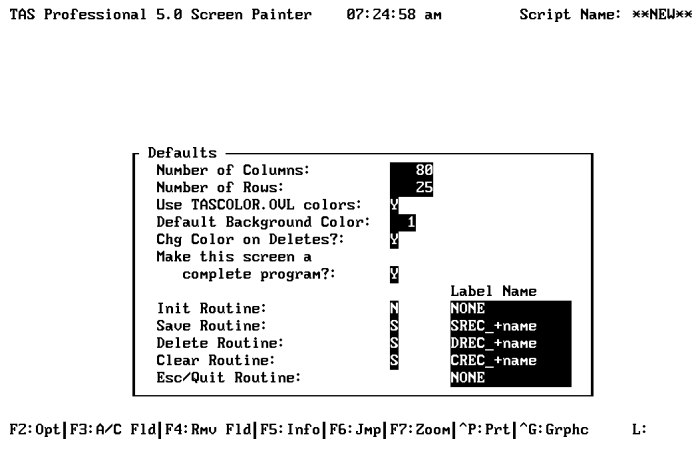
F6 - Starting with the next field in the format, this operation will move the cursor from field to field, left to right, top to bottom.

F7 - Display the format on the screen without any extraneous information. This is what the format will look the way the screen will appear once the format is compiled into a program and run.

^P - Print the format information.

^G - Display a menu of graphic characters and put one on the screen at the current cursor location.

Slider fields - Many entries are made into ‘slider’ fields. These are fields that display smaller on the screen than they really are. In many cases you can enter up to 128 characters. As you enter into these fields and the cursor reaches the right hand side of the entry window, the characters will ‘slide’ to the left. To get back to those characters just press the LEFT ARROW key. Again, when you reach the left hand side of the entry window the characters will slide to the right. You can get to the beginning of the field by pressing the HOME key and to the end by pressing the END key. You can move one word right by pressing the ^RIGHT ARROW key or one word left with the ^LEFT ARROW key.



Edit screen format

Defaults Screen

This is the initial entry when you are creating a new script. If you want to make changes to a current script then press the F2 key and choose Chg Defaults from the menu.

Number of columns, number of rows - maximum size of screen. Current maximum size is 80 x 25. These 2 values are the only ones that cannot be changed once accepted during the initial entry.

Use TASCOLOR.OVL colors? - If you enter **Y** here the program will use the values in the TASCOLOR.OVL file. The color values entered will relate to an 'array' of colors, 1 being the first, 2 the second, etc.

Default Background Color - The screen background color.

Chg Color on Deletes? - If you enter **Y** here the appropriate area on the screen will revert to the Default Background Color whenever you delete a character or move a block of characters. If you enter **N** here you will need to use the **Color Block** option if you need to change a color currently on the screen.

Make this screen a complete program? - If you enter **N** here, when you run Make Program it will create the enter commands only. This is helpful when you want to include this in another program you have written but don't want all the other parts, such as Defines, Opens, etc. If you are here from the Editor the screen format will be included automatically in the source code.

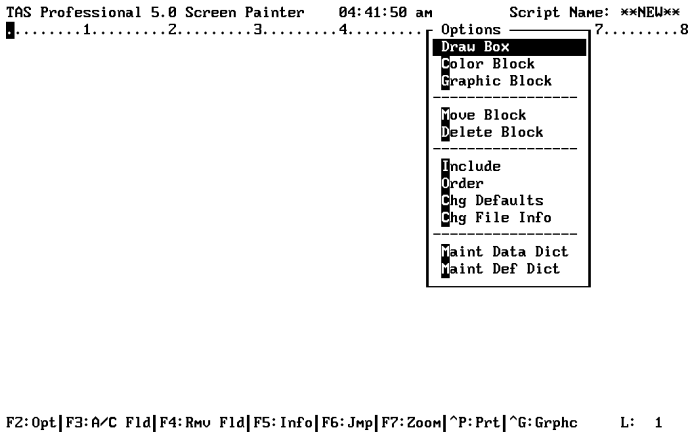
Init Routine - The name of the label preceding any routine you want to run previous to entering the fields. The routine is called with a **GOSUB** and should have a **RETURN** at the end. You can also set up where you want this subroutine called: **N** - no INIT routine (default); **B** - Call routine before the **MOUNT** screen command; **A** - Call routine after the **MOUNT** screen command.

Save, Delete, Clear Routines - These are the standard routines normally created in a program. If you make no changes here TAS 5.1 will create standard routines including all files opened. The routines will be preceded by line labels formed of SREC_ or DREC_ or CREC_ and the name of the screen format. If you do not want to have any of these routines included then

enter **N** instead of **S**. If you want to use your own routines in an **INCLUDE** file then enter **L** and the line label name where applicable.

Esc/Quit Routine - The line label to transfer control to if the user presses the ESC key. Normally the program will just QUIT and return to the previous program or DOS. This option is similar to the Init Routine.

NOTE: You must specify the file name to be included if you have entered **L** for any of the options above or the Init Routine/Esc name.



Edit screen format

Options Menu

Press the F2 key to display the options menu.

Draw box - Place a box on the screen

Color block - Mark a block and then choose a color. This will not change any regular characters; it will just fill the block with the chosen color.

Graphic Block - Mark a block and then choose a graphic character. This will fill the block with the chosen character. It will not overwrite any fields within the block.

Move block - Choose a block and then move the cursor to the new upper left corner location. The block may contain fields.

Delete block - Delete a block from the screen. The block may contain fields.

Include - Create a list of source code files that are to be included at compile time.

Order - Choose the order in which the fields are to be entered within the (compiled) program. If you do not specify an order here they will be entered left to right, top to bottom.

To specify the order you are given two lists. The one on the left is the ordered list, the one on the right is the available fields. To add an available field to the order list either press the ENTER key on a blank line or press the insert key where you want to insert a field. The cursor will go to the available list and you will be able to choose a field. Press ENTER again and it will be added to the order list. To remove a field from the order list put the cursor on the field

name and press the DEL key. If you answer **Y** to the remove question it will be taken out of the order list and will be put back into the available list.

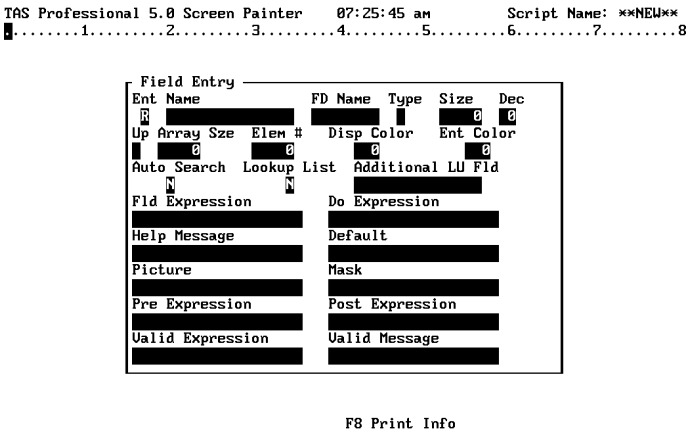
To exit this option press the ESC key.

Chg defaults - Change the default entries made when the screen format was first created.

Chg File Info - This is the information about which data file is the main one and what index to use, if any. You will also be able to set up any relationships between files using this option. When you create a screen format for the first time and there are fields from data files in that format, this entry will automatically occur. You will be able to change any entry made through the use of this option. For more details, please refer to Change File Information below.

Maint data dict - This will chain to (run) the File Data Dictionary Manager so that you may make changes directly to a file definition. All fields must be in the data dictionary or the defined data dictionary before they can be added to the format, or they must be defined as 'MEMORY' fields.

Maint def dict - This will chain to the Defined Data Dictionary Manager so that you may make changes directly to fields that are defined there. All fields must be in one of the data dictionaries before they can be added to the format, or they must be defined as 'MEMORY' fields within the screen.



Edit screen format

Field Info Screen

This screen is displayed when you press the F3 or F5 keys.

Ent - **R** - regular enter field, **D** - display only field, **O** - do something (when the user presses the F2 key execute a DO UDF), **A** - array type entry, **N** - enumerated entry, **X** - expression (put an expression directly on the screen). For the X type, you specify a size and an actual expression. No checking is done by TAS Professional 5.1 to make sure the expression is valid or that the data displayed is the proper size. You need to perform those checks within your program.

NOTE: For more information about the different types of **ENTER** options please refer to **Chapter 5 - Command Reference**.

Name - The name of the field. You can press the F2 key and get a list of fields in the file data dictionary, or press F3 and you get a list of fields in the defined data dictionary.

NOTE: If this is a modification to a field that was already on the format you will not be able to change the name. You must delete the field (remove it from the format) first, and then replace it.

FD Name - The file descriptor name. If this is a field from the data dictionary this will be the FD name for that field. If this is a defined (or to be defined) field this will be MEMORY.

Type, Size, Dec, Upcase - Standard field specs.

Array Sze - If this is a MEMORY field and has not been used before in the format you can specify a number of array elements, if applicable. If this is a file field and it is an array field then the number of elements is placed here.

Element num - If the Array Sze for this field is <>0 then you have to specify the element number. You may not use the same element number more than once in any format.

Disp Color - The color value to be used when displaying this field.

Enter Color - The color value to be used when entering this field.

Auto Search - If this is a data file field and it is also a key (index) then you will be able to choose

whether or not the program should automatically search for a record after entry. If you enter **Y** here the program will attempt to find a record that matches the data entered. If a record is already active for this file then no search will be performed. NOTE: The field name must match a key name or you won't be given the opportunity to enter this option.

Lookup List - If this field is a key (index) as above, you can also specify an automatic FAST SEARCH™ lookup list to be generated. This means that at runtime the user will be able to press the F2 key, when the cursor is positioned on that field, and get a list of records in the key order. The user will be able to choose a record based on the information displayed.

Additional LU Fld - If you have chosen to generate a Lookup List (above) you can also designate an additional field here that will be displayed along with the lookup field. An example of this would be if a lookup list was used for a customer code you would designate the customer name as the Additional Lookup Field.

Fld Expr - The field expression. If the Ent type is **D** or **O** then this is what will be displayed on the screen for this field. If the type is **X**, the value returned from evaluating the expression will be shown on the screen. If **N** this is the list of options that the user will have to choose from. If you are using alpha constants don't forget to surround them with quotes and separate them with commas, e.g.,

```
'DO','GOTO','GOSUB'
```

Do Expr - If the Ent type is **O** this is the UDF to execute when the user presses the F2 key. Don't forget this must be in standard function notation including the parentheses at the end of the UDF name. You can pass values to the UDF by including them within the parens.

Help Msg - This is the message to display if the user presses the F1 key while editing this field. If you are going to enter an alpha constant make sure it is surrounded by quotes.

To edit this field press the F2 key and TASEdit will be called. You will then be able to enter your message. You may enter a maximum of 320 characters.

Picture - This is the **PICT** option in the **ENTER** command. If this is an alpha (A) type field you can designate it as a 'slider'. A slider is a field that shows fewer characters on the screen than actually exist in the field. The user can see the other characters by pressing the LEFT and RIGHT ARROW keys and the characters will appear to 'slide' back and forth across the available space. This is very useful when you have several large fields and want them all to fit on the same screen. A slider field is specified with an alpha constant of "Sxx", where xx is the number of columns to display. For example, S20 will allow a slider field with 20 characters displayed at any time. If you are placing a field on the screen that cannot fit (would extend off the end of the screen starting at the current cursor location) you must designate a slider here or the program will not allow you continue.

Mask - This is the **MASK** option in the **ENTER** command. This is the allowable entry characters for this field. If this is an alpha constant be sure it is surrounded by quotes. This is a slider field.

Default - This is the **DFLT** option in the **ENTER** command. If the field is 0 or blank when entered this value will be displayed. This is a slider field.

Pre Expr - This is the **PRE** option in the **ENTER** command. It can be a UDF or just a standard expression. It must resolve to .T. or .F.

Post Expr - This is the **POST** option in the **ENTER** command. It can be a UDF or just a standard expression. It must resolve to .T. or .F.

Valid Expr - This is the **VLD** option in the **ENTER** command. It can be a UDF or just a standard expression. It must resolve to .T. or .F.

Valid Msg - This is the VLDM option in the ENTER command. It will be displayed if the Valid Expression returns .F.. If this is an alpha constant be sure it is surrounded by quotes.

To edit this field press the F2 key and TASEdit will be called. You will then be able to enter your message. You may enter a maximum of 320 characters.

NOTE: For more information about the options in the **ENTER** command please see **Chapter 5, Command Reference**.

Edit screen format

Operations

This section will explain in outline form how to accomplish the various functions that are available in this program.

Adding a new field on the screen

a) Move the cursor to the appropriate location. The entry window for the new field will start at this location and extend to the right. You need to make sure that the field won't overlap any other fields.

b) Press the F3 key and the Field Info window will be displayed. Enter the appropriate data.

c) Once you have entered the Name, type, size, and Ent type you can press the F10 key to complete the entry. Or continue entering the various options until you reach the end of the window.

d) The program will ask if all is correct, if you answer **Y** the field will be placed on the screen. The program uses a symbol that looks like a small house for each display character in the field.

e) To quit the add field process press the ESC key at any time. The program will ask if you want to return to the full screen, enter **Y**.

Removing a field from the screen

a) Move the cursor to any position in the field that is to be removed.

b) Press the F4 key.

c) The program will ask if you wish to remove the field, enter **Y**.

d) The field representation will be removed from the screen.

Inserting a line into the screen format

a) Move the cursor to the line above which the new line is to be added. The new line will be added at the cursor and all lines will be moved down one.

b) Press the ^A key.

c) A blank line will be added at the cursor. If there are any characters on the last line of the format they will be deleted. However, if there is a field on the last line you will be alerted to that fact and will have to enter **Y** to the 'are you sure' question to continue.

Deleting a line in the format

a) Move the cursor to the line you wish to delete.

b) Press the ^D key.

c) The line will be deleted and all lines below will be moved up 1. A blank line will be added to the end of the format. If there is a field on the line to be deleted you will be alerted to that fact and will have to enter **Y** to 'are you sure' question to continue.

Placing a box on the screen

a) Move the cursor to where you want the upper left hand corner of the box to begin.

b) Press the F2 key to display the options menu.

c) Press **D** or the ENTER key to choose the Draw Box option.

d) Now move the cursor to the lower right hand corner of the box and press the ENTER key. To interrupt the process and return to the normal window press the ESC key.

e) A window will appear that displays the start and stop coordinates.

f) Enter the box type: **0** - Clear a box already on the screen; **1** - Use single line characters, **2** - use double line characters, **3** - Use double line characters for the horizontal lines and single line characters for the vertical, **4** - Use single line characters for the horizontal lines and double line characters for the vertical.

g) Enter the color to use. If you wish to use the background color enter 0. You can also get the available color values by pressing the F2 key.

h) The box will be displayed (or cleared) and the regular window will be re-displayed with the cursor at the upper left hand corner of the box.

NOTE: To draw a line vertically or horizontally just move the cursor down without moving it to the right, or to the right (for a horizontal line) without moving it down.

Paint a color block

a) Move the cursor to where you want the upper left hand corner of the block to begin.

b) Press the F2 key to display the options menu.

c) Press **C** and the ENTER key to choose the Color Block option.

d) Now move the cursor to the lower right hand corner of the block and press the ENTER key. To interrupt the process and return to the normal window press the ESC key.

e) A window will appear that displays the start and stop coordinates.

f) Enter the color to use. You can get the available color values by pressing the F2 key.

g) The program will ask if all is correct, if you enter **Y** the color within the block will be changed appropriately and the regular window will be re-displayed with the cursor at the upper left hand corner of the block. If you answer **N** you will return to the normal window with nothing changed.

Create a block of graphic characters

a) Move the cursor to where you want the upper left hand corner of the block to begin.

b) Press the F2 key to display the options menu.

c) Press **G** and the ENTER key to choose the Graphic Block option.

d) Now move the cursor to the lower right hand corner of the block and press the ENTER key. To interrupt the process and return to the normal window press the ESC key.

e) A window will appear that displays the start and stop coordinates along with a menu of graphic characters.

f) Enter the character to use by moving the cursor to the appropriate line and pressing the ENTER key.

g) The program will ask if all is correct, and if you enter **Y** the block will be filled with the character and you will return to the regular window with the cursor at the upper left hand corner of the block. If you answer **N** you will return to the normal window with nothing changed.

Move a block

a) Move the cursor to where you want the upper left hand corner of the block to be moved.

b) Press the F2 key to display the options menu.

c) Press **M** and the ENTER key to choose the Move Block option.

d) Now move the cursor to the lower right hand corner of the block and press the ENTER key. To interrupt the process and return to the normal window press the ESC key.

e) Now move the cursor to the new location of the upper left hand corner of the block. The program will make sure you don't move anywhere on the screen where the block wouldn't be able to fit.

f) Press the ENTER key to designate the cursor location as the spot to move the block. The program will check for other fields in the area that might be overwritten by the move and will not allow you to move the block if there would be a problem.

g) The cursor will return to the location of the original upper left hand corner of the block.

Delete a block

a) Move the cursor to where you want the upper left hand corner of the block to be deleted.

b) Press the F2 key to display the options menu.

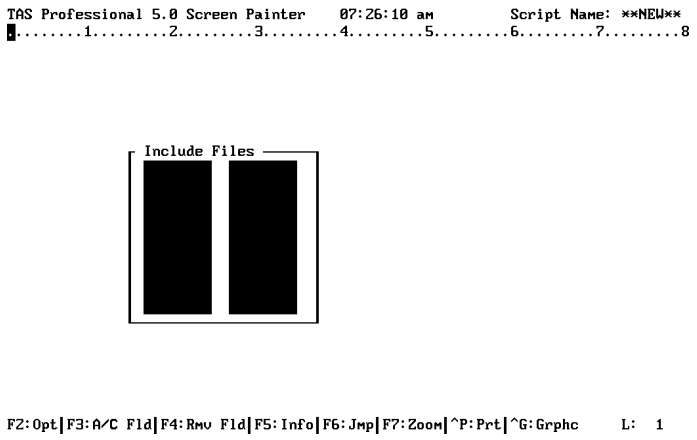
c) Press **D** and the ENTER key to choose the Delete Block option.

d) Now move the cursor to the lower right hand corner of the block and press the ENTER key. To interrupt the process and return to the normal window press the ESC key.

e) The program will ask if you're sure you want to delete the block. To do so press **Y**.

f) If you choose to delete the block it will be removed from the screen including any fields and the color will be replaced with the default.

g) The cursor will return to the location of the original upper left hand corner of the block.



Include other source files at compilation time

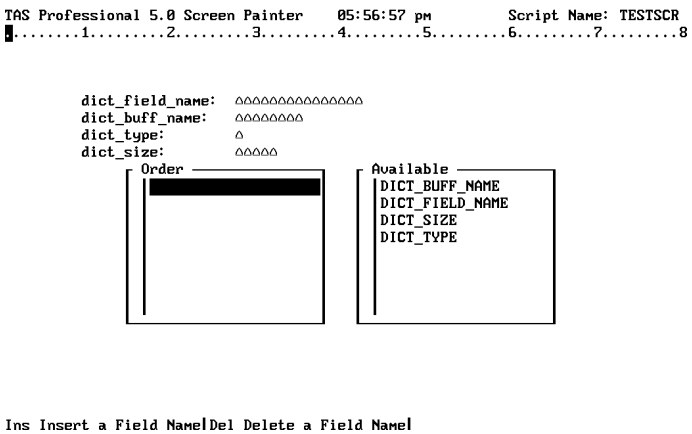
a) Choose this option if you have source files you wish to include with the main program at compile time.

b) Press the F2 key to display the options menu.

c) Press **I** and the ENTER key to choose the include option. The screen above will be displayed.

d) The cursor will move to the first line. To add a new include file name move the cursor to a blank line (unless the first line is blank) by pressing the DOWN ARROW key.

e) To delete a line move the cursor to the appropriate line and press the F4 key. You will be asked if you're sure. If you enter **Y** the line will be deleted and the remaining ones moved up 1.



Change the order of entry

a) Normally the fields are set up so that they are entered as they appear on the screen, starting with the field in closest to the top of the screen and the farthest left. It then proceeds to the right. Then to the next line, farthest left again, to the right, etc. So, fields are set up to enter top-left to bottom-right. To change this order choose this option.

b) Press the F2 key to display the options menu.

c) Press **O** and the ENTER key to choose the order option. The screen above will be displayed. In the right window appears the list of fields that are part of the screen format in name order. The left hand window will contain the fields in the order they are to be entered. You need not order all the fields; however, those you have will be entered first, and the balance will be entered in the normal order.

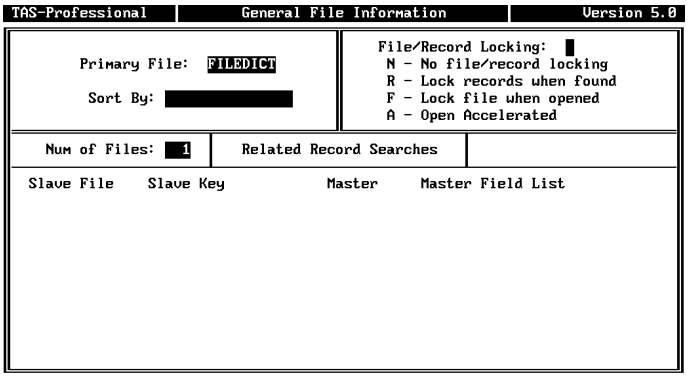
d) To move a field into the order list move the cursor bar to the appropriate location. Press the ENTER key if this field is to go to the end of the list (the cursor bar is a blank line) or press the INSERT key if you wish to insert a field before the cursor.

e) The cursor bar will move to the window that contains the available fields. Move the cursor to the appropriate field and press the ENTER key. It will be removed from the available list and move to the appropriate location in the order list.

f) To delete a field from the order list (and return it to the available list) move the cursor to that field (while in the order list window) and press the DELETE key. You will be asked if you are sure you want to remove the entry; if so press **Y**. The field will be removed from the order list window and added to the available list window.

g) When you are finished press the ESC key. The program will ask if all is correct; if you enter **Y** the information will be saved.

h) No matter what changes you make there will be no effect on the fields themselves. This will only change the order in which they are entered.



Change File Information

a) This information is generally entered the first time when you are finished creating a new screen format and you have gone through the initial saving stages. You can also change this information at any time.

b) Press the F2 key to display the options menu.

c) Press C three times (the first time will get you to the Color Block line, the second Chg Defaults) and the ENTER key to choose the Chg File Info option. The screen above will be displayed. The Primary File is the main or master file for the program. If all the fields placed are from one file only then this will automatically be that file name. The Sort By option is the name of the key (index) to be used when accessing records from this file. If you leave this field blank the program that is created from this screen format will ask you to choose an index at runtime. The File/Record locking option allows you to choose what type of locking (if this is a multi-user installation) to use if any. The second part of the screen displays the files used and allows you to set up relations between those files.

d) When the cursor is in the Primary file entry field you can press the F2 key to get a list of the files used in the program. Choose one.

e) To designate an index to be used you can enter that key name here. To get a list of keys available for the Primary file press the F2 key. If you leave this option blank the program will ask for a key to be chosen at runtime.

f) Next you will be asked for what type of file or record locking you wish to choose. You can specify record or file locks and still run the program in single user mode. This will only become effective if you are running in a multi-user situation.

g) Next you will be asked to set up the relationships between/among the various files. Even though it is listed you will not be able to choose the Primary file. Move the cursor bar to the file you wish to choose and press the ENTER key. The cursor bar will be removed and a standard entry field will be displayed under the Slave Key title. Here you need to enter the key or index name that will be used to find records in this file when you find a record in the master file. You have several choices of things to do here. You can: Press the F2 key to get a list of relations already set up for this file; Press the F3 key to get a list of indexes for the file; or Press the F5 key to tell the program to find the first record in this file when the program is run. This would mean that this file was not connected to the master file at all.

h) Next enter the Master file name. If there is only one other file then this would be that file name. However, you can have multiple relationships; not all files have to be related to the

master. To get a listing of all files press the F3 key. Enter an accurate file name.

i) Enter the fields in the Master that correspond to the key information in the Slave. This means that when the Master record is found the program will use the data in these fields when searching for the Slave record. If there are multiple fields then separate them with commas.

j) The last step is to determine whether you want to update the Master from the Slave data or vice versa. You can also choose to update neither. This will make sure when you're creating new records that the linkage information is properly updated. The data you're displaying on the screen (and allowing your user to enter) will determine which option to choose.

k) Continue setting up relationships for all other files. When you are finished, press the ESC key. If this is a new script the saving process will complete. If this is an old file that you are modifying the program will return to the main screen.

Save Screen Format Script

a) When you are finished setting up or changing your format press the ESC key.

b) The program will ask if you want to save the script file for this format. Enter **Y** if you do.

c) The program will then display a window in which you should enter the script name. If you have edited an existing file the name will be displayed automatically. You can put in a different name if you wish to save a different file.

d) After you enter the name the program will save the file and return to the original screen. If this is a new format and there are fields from at least one file the program will display the File Information screen (as above) and ask for the appropriate data.

COMPINFO.SCP has been included as an example of a typical screen format script.

Screen formats -> Make program

You reach this option by placing the cursor on the **Screen formats** option line and pressing the ENTER key or by pressing the **S** key. Then press the **M** key or move the cursor to the **Make program** option line and press the ENTER key to run the program.

Program name: TASMKPRG.RUN

This program creates the appropriate code from the screen format scripts created in TASEDSCR.RUN.

If this program is called from TASEDSCR the name of the format will already be on the screen. If it isn't you will be able to enter the format name.

The program will then ask you if you wish to create the appropriate source code. If you answer **Y** the program will also ask you if you want to compile the program after the creation process is complete. If you answer **Y** again the program will be created. If any errors have occurred during the compilation process (included files not found, UDFs not found, etc.) then the number will be displayed. If the program compiled correctly you will be able to run it at that point.

If you create a source file and one already exists the program will alert you to that and you will have to enter **Y** at the appropriate question to create the program.

Report formats -> Edit report format

You reach this option by placing the cursor on the **Report formats** option line and pressing the ENTER key or by pressing the **R** key. Then press the ENTER key again to run the program.

Program name: TASEDRPT.RUN

This program creates and changes report 'scripts'. A script is a specialized format type that keeps all the information about that report format but cannot actually be run.

You may edit and create a new program as many times as you desire from a report format script. In fact, you may use a script as the basis for a new format by placing the name of the appropriate script as the format to be changed at the initial prompt and then saving it under a different name when you are finished.

All script files created from this program have the extension .SRP.

Edit report format

Control Keys

> **NOTE:** When the character '^' precedes the key name it means to press the CTRL key at the same time you press the other key.

Dn arrow - move down one line.

Up arrow - move up one line.

Pg Dn - move down to lower right hand corner of current screen.

^Pg Dn - move to the far lower corner of the report format.

Pg Up - move to upper left hand corner of current screen.

^Pg Up - move to the far upper left hand corner of the report format.

End - move to end of current line

Home - move to beginning of current line.

Tab - move to the right 5 spaces.

Shift-Tab - (Back Tab) - move to the left 5 spaces.

^A - Add a line to the format at the cursor position. Any characters on the last line will be lost.

^D - Delete a format line at the cursor position. Any characters on that line (including fields) will be removed.

F2 - Display the option menu (see options below).

F3 - Add a new field spec or change a current one (to change one the cursor has to be placed somewhere on the field marker).

F4 - Remove a current field from the format (the cursor has to be placed somewhere on the field marker).

F5 - Display (and/or print) the information about a field in the format.

F6 - Starting with the next field in the format this operation will move from field to field, left to right, top to bottom.

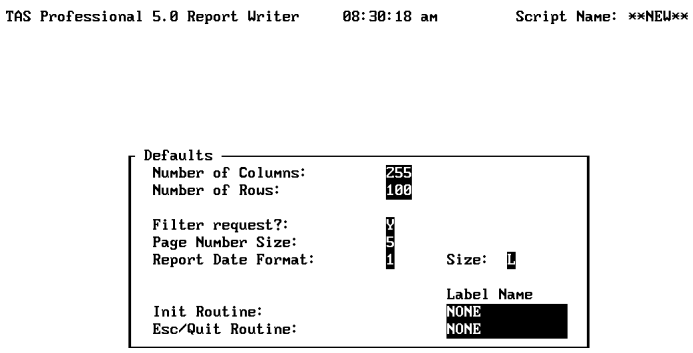
F7 - Center all characters on the line.

F8 - Enter an If Print expression.

^P - Print the format information.

^G - Display a menu of graphic characters and put one on the screen at the current cursor location.

Slider fields - Some entries are made into ‘slider’ fields. These are fields that display smaller on the screen than they really are. In many cases you can enter up to 128 characters. As you enter data into these fields and the cursor reaches the right hand side of the entry window, the characters will ‘slide’ to the left. To get back to those characters just press the LEFT ARROW key. Again, when you reach the left hand side of the entry window the characters will slide to the right. You can get to the beginning of the field by pressing the HOME key and to the end by pressing the END key. You can move one word right by pressing the ^RIGHT ARROW key or on word left with the ^LEFT ARROW key.



Edit report format

Defaults Screen

This is the initial entry when you are creating a new script. If you want to make changes to a current script then press the F2 key and choose Chg Defaults from the menu.

Number of columns, number of rows - maximum size of report. Current maximum size is 255 x 100. Each row can be a different format line. Each row can be repeated many times on one page or over the entire report. The rows do not necessarily correspond to the same line on the paper. These 2 values are the only ones that cannot be changed once accepted during the initial entry.

Filter Request - If you answer **Y** here the finished program will ask the user for a filter to be entered at runtime. This is used for restricting the records to be displayed in the report.

Page Number Size - The program keeps a default field for page numbers. You can specify the size

of that field here. It can range from 1 to 5 characters, the default is 5. To use this field in a report format, place the field name PAGE_NUM where you wish it to appear.

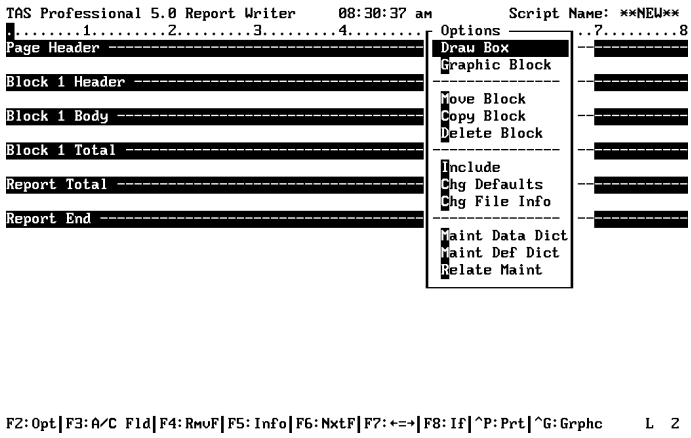
Report Data Format - This determines the program default date type. Use **1** for mm/dd/yy (will display in appropriate format as set up in the Set Configuration program), **2** for Month Day, Year (i.e., January 1, 1992), or **3** for Day Month Year (i.e., 1 January 1992). To use this field in a report format, place the field name RPT_DATE where you wish it to appear.

Size - If you choose **2** or **3** above you will also be able to set whether you want the names to print in long or short format, i.e., long (L) would be January, short (S) would be Jan.

NOTE: You can also place the time on your reports by using the field RPT_TIME. The size of this field is fixed at 7 which will produce hh:mm a/p (am or pm).

Init Routine - The name of the label preceding any routine you want to run prior to printing the report. The routine is called with a **GOSUB** and should have a **RETURN** at the end. It will be executed after all files are opened and before the report printing is started.

Esc/Quit Routine - The line label to transfer control to if the user presses the ESC key. Normally the program will just QUIT and return to the previous program or DOS.



Edit report format

Options Menu - Regular Report Format Lines

Press the F2 key when the cursor is on a regular report format line to display this options menu.

Draw box - Place a box on the screen

Graphic Block - Mark a block and then choose a graphic character. This will fill the block with the chosen character. It will not overwrite any fields within the block.

Move block - Choose a block and then move the cursor to the new upper left corner location. The block may contain fields.

Copy block - Choose a block and then move the cursor to the new upper left corner location. The block may contain fields. On a report format you can have multiple occurrences of the same field.

Delete block - Delete a block from the screen. The block may contain fields.

Include - Create a list of source code files that are to be included at compile time.

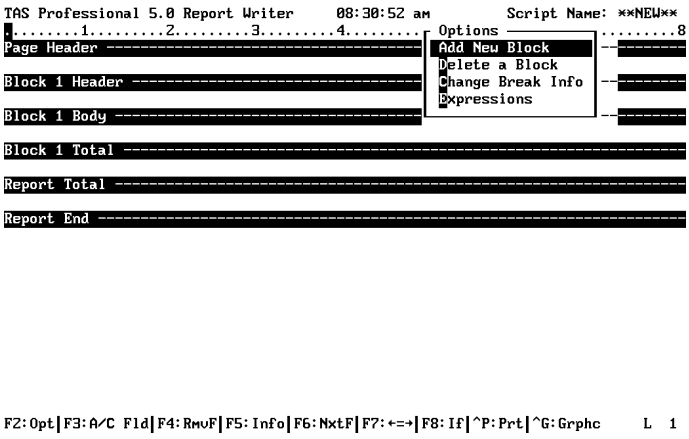
Chg defaults - Change the default entries made when the screen format was first created.

Chg File Info - This is the information about which file is the main one and what index to use, if any. You will also set up any relationships between files using this option. When you create a report format the first time, if there are fields from data files in that format, this entry will automatically occur. You will be able to change any entry made through the use of this option. For more please refer to Change File Information below.

Maint data dict - This will chain to (run) the File Data Dictionary Manager so that you may make changes directly to a file definition. All fields must be in the data dictionary or the defined data dictionary before they can be added to the format, or they must be defined as 'MEMORY' fields.

Maint def dict - This will chain to the Defined Data Dictionary Manager so that you may make changes directly to fields that are defined there. All fields must be in one of the data dictionaries before they can be added to the format, or they must be defined as 'MEMORY' fields within the screen.

Relate Maint - This will chain to the Maintain Relationships Manager.



Edit report format

Options Menu - Control Lines (Bands)

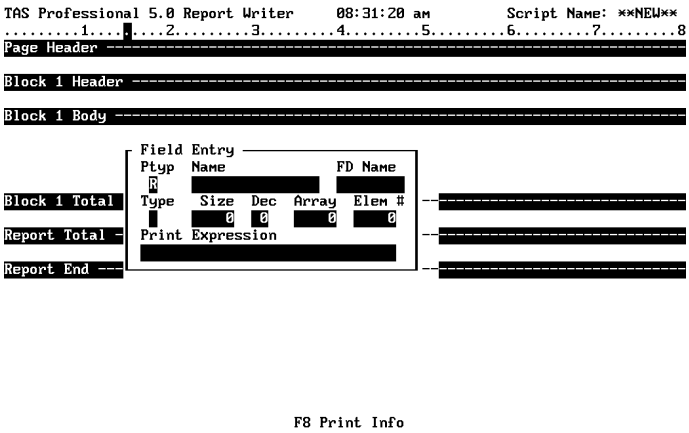
Press the F2 key when the cursor is on a control or band line to display this options menu.

Add a new block - Add a new set of block bands immediately before the current band. If you currently have one block in the format, control bands for Block 2 Header, Body and Total will be inserted.

Delete a block - Delete a set of block bands, including all regular report format lines currently part of that band. To do this you must have the cursor placed on the Header band line for that block.

Change Break Info - Change the file and key information for that block.

Expressions - Add or change expressions to be executed while that band is active during the report.



Edit report format

Field Info Screen

This screen is displayed when you press the F3 or F5 keys and the cursor is not on a control line (band).

Ptyp - The type of field/expression being placed in the report. If this is a regular field (data file or defined) then this should be **R**. If this is an expression then enter **X** here.

Name - The name of the field. You can press the F2 key and get a list of fields in the file data dictionary. Press F3 to get a list of fields in the defined data dictionary.

NOTE: If this is a modification to a field that was already on the format you will not be able to change the name. You must delete the field (remove it from the format) first, and then replace it.

FD Name - The file descriptor name. If this is a field from the data dictionary this will be the FD name for that field. If this is a defined (or to be defined) field this will be MEMORY. If it is a field created just for a report (date, time, page number) the FD name will be RPTWRTR.

Type, Size, Dec, Upcase - Standard field specs.

NOTE: To put an expression directly on the screen, use the **X** type. Specify a size, then enter the actual expression. No checking is done by TAS Professional 5.1 to make sure the expression is valid or that the data displayed is the proper size. You need to perform those checks within your program.

Array Sze - If this is a MEMORY field and has not be used before in the format you can specify a number of array elements, if applicable. If this is a file field and it is an array field then the number of elements is placed here.

Element num - If the Array Sze for this field is <>0 then you have to specify the element number.

Print Expression - If the Ptyp is **X** then this is where you would put the expression to be evaluated. The results of that expression will be printed.

Edit report format

Operations

This section will explain in outline form how to accomplish the various functions that are available in this program.

General

a) For a new report format 6 band lines are displayed, they are: Page Header; Block 1 Header; Block 1 Body; Block 1 Total; Report Total; and Report End. Nothing can be placed after the Report End line.

b) The Page Header prints at the beginning of each page. The other band lines depend on the information entered. The Report Total lines print once at the end of the report.

c) Each time a block is added the Header, Block, and Total lines for that block are added. You need not nest blocks; they can also be sequential.

d) A form feed is always printed at the end of the report.

e) The Make Report program uses the information entered here to create the report. For a more general explanation of how the report writer works we recommend that you look at the examples in the tutorial. They give a good overview of how to create a report. Also, setting up a report, running it, making changes, running it again, etc. is the quickest way to see all of the options that are available. This is a very powerful program and can be used to make most of the reports you might want.

Adding a new field on the report

a) Move the cursor to the appropriate location. This cannot be on a control line (band). The field display will start at this location and extend to the right. You need to make sure that the field won't overlap any other fields.

b) Press the F3 key and the Field Info window will be displayed. Enter the appropriate data. Refer to the Field Information Window documentation described in the Screen Format section in this chapter for more details.

d) The program will ask if all is correct, and if you answer **Y** the field will be placed on the screen. The program uses a symbol that looks like a small house for each display character in the field.

e) To quit the entry window press the ESC key at any time. The program will ask if you want to return to the full screen; enter **Y**.

Removing a field from the report

a) Move the cursor to any position in the field you wish to remove.

b) Press the F4 key.

c) The program will ask if you wish to remove the field; enter **Y**.

d) The field representation will be removed from the screen.

Inserting a line into the screen format

a) Move the cursor to the line above which the new line is to be inserted. The new line will be added at the cursor and all lines will be moved down one.

b) Press the ^A key.

c) A blank line will be added at the cursor. The band line Report End will always be the last line in the report. If you don't have enough extra rows (set in the Defaults entry as the number of rows) you will be alerted to this and will not be able to add a line.

Deleting a line in the format

a) Move the cursor to the line you wish to delete.

b) Press the ^D key.

c) The line will be deleted and all further lines will be moved up 1. If there is a field on the line to be deleted you will be alerted to that fact and will have to enter **Y** to the 'are you sure' question to continue. You cannot delete a band line.

Setting up a Print If expression

a) Use this option to control whether or not a line, or block of lines, will print.

b) By moving the cursor to a block band line you will control all the lines within that band. If the cursor is on a regular format line only that line will be affected. You can have both the block band line and individual lines within that block set.

c) Move the cursor to the appropriate line.

d) Press the F8 key.

e) Enter a standard TAS Professional 5.1 expression, such as:

```
sales_amount < 0
```

In this example this line will print only if the sales_amount value is less than 0.

f) Press the ENTER key to save the value and the program will return to the main screen. To make changes in that expression move the cursor to the appropriate line and press the F8 key again. The expression will be displayed and you can make changes to it if desired. To delete the expression press the ^U key and then the ENTER key so that the expression line is blank.

Placing a box on the screen

a) Move the cursor to where you want the upper left hand corner of the box to begin. The box must fit entirely within one block and cannot start or stop on a control line (band).

b) Press the F2 key to display the options menu.

c) Press **D** or the ENTER key to choose the Draw Box option.

d) Now move the cursor to the lower right hand corner of the box and press the ENTER key. To interrupt the process and return to the normal window press the ESC key. You cannot cross a band line.

e) A window will appear that displays the start and stop coordinates.

f) Enter the box type: **0** - Clear a box already on the screen; **1** - Use single line characters, **2** - use double line characters, **3** - Use double line characters for the horizontal lines and single line characters for the vertical, **4** - Use single line characters for the horizontal lines and double line characters for the vertical.

g) Enter the color to use. If you wish to use the background color enter **0**. You can also get the available color values by pressing the F2 key.

h) The box will be displayed (or cleared) and the regular window will be re-displayed with the cursor at the upper left hand corner of the box.

NOTE: To draw a line vertically or horizontally just move the cursor down without moving it to the right, or to the right (for a horizontal line) without moving it down.

Create a block of graphic characters

a) Move the cursor to non-control line (band) where you want the upper left hand corner of the block to begin.

b) Press the F2 key to display the options menu.

c) Press **G** and the ENTER key to choose the Graphic Block option.

d) Now move the cursor to the lower right hand corner of the block and press the ENTER key. To interrupt the process and return to the normal window press the ESC key. You cannot cross a band line.

e) A window will appear that displays the start and stop coordinates along with a menu of graphic characters.

f) Enter the character to use by moving the cursor to the appropriate line and pressing the ENTER key.

g) The program will ask if all is correct, and if you enter **Y** the block will be filled with the character. You will return to the regular window with the cursor at the upper left hand corner of the block. If you answer **N** you will return to the normal window with nothing changed. Any text characters (non-field) will be converted to ghe graphics character if they are within the block.

Move a block

a) Move the cursor to the upper left hand corner of the block to be moved.

b) Press the F2 key to display the options menu.

c) Press **M** and the ENTER key to choose the Move Block option.

d) Now move the cursor to the lower right hand corner of the block and press the ENTER key. To interrupt the process and return to the normal window, press the ESC key. You cannot cross a band line while choosing a block; however, you can move the block anywhere.

e) Now move the cursor to the new location of the upper left hand corner of the block. The program will make sure you don't move anywhere on the screen where the block wouldn't be able to fit. However, if all the program needs to do is add extra lines to the band block it will do so automatically.

f) Press the ENTER key to designate the cursor location as the spot to move the block. The program will check for other fields in the area that might be overwritten by the move and will not allow you to move the block if there would be a problem.

g) The cursor will return to the location of the original upper left hand corner of the block.

Copy a block

a) Move the cursor to the upper left hand corner of the block to be copied.

b) Press the F2 key to display the options menu.

c) Press **C** and the ENTER key to choose the Copy Block option.

d) Now move the cursor to the lower right hand corner of the block and press the ENTER key. To interrupt the process and return to the normal window press the ESC key. You cannot cross a band line while choosing a block; however, you can copy the block anywhere.

e) Now move the cursor to the new location of the upper left hand corner of the block. The program will make sure you don't copy anywhere on the screen where the block wouldn't be able to fit. However, if all the program needs to do is add extra lines to the band block it will do so automatically.

f) Press the ENTER key to designate the cursor location as the spot to copy the block. The program will check for other fields in the area that might be overwritten by the block and will not allow you to copy the block if there would be a problem.

g) The cursor will return to the location of the original upper left hand corner of the block.

Delete a block

a) Move the cursor to the upper left hand corner of the block you want to be deleted.

b) Press the F2 key to display the options menu.

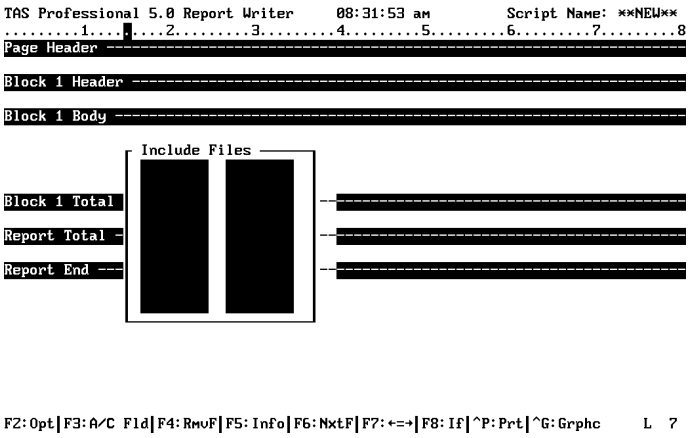
c) Press **D** and the ENTER key to choose the Delete Block option.

d) Now move the cursor to the lower right hand corner of the block and press the ENTER key. To interrupt the process and return to the normal window press the ESC key.

e) The program will ask if you're sure you want to delete the block. To do so press **Y**.

f) If you choose to delete the block it will be removed from the screen including any fields and the color will be replaced with the default.

g) The cursor will return to the location of the original upper left hand corner of the block.



Include other source files at compilation time

a) Choose this option if you have source files you wish to include with the main program at compile time. If you specified an Init or Esc/Quit routine on the Defaults screen when you started this format, you will need to enter the file name(s) containing these routines here.

b) Press the F2 key to display the options menu. The cursor must be on a regular report format line and not a block or band line.

c) Press **I** and the ENTER key to choose the include option. The screen above will be displayed.

d) The cursor will move to the first line. To add a new include file name move the cursor to a blank line (unless the first line is blank) by pressing the DOWN ARROW key.

e) To delete a line move the cursor to the appropriate line and press the F4 key. You will be asked if you're sure. If you enter **Y** the line will be deleted and the remaining ones moved up 1.

The following options are available when you place the cursor on a band or block line and press the F2 key.

Add New Block

a) Use this option to add a new block immediately before the cursor location. You would do this when you need to access a different file as part of the same report.

b) Move the cursor to the block line above which the new block will be inserted.

c) Press the F2 key to get the options menu.

d) Press **A** and then the ENTER key. The three new block lines (Header, Body, Total) with the next block number will be displayed on the screen.

e) The Break Field Info entry block will be displayed and you can enter the appropriate information for this block. For more about this operation, please refer to the Break Field Information section below.

f) To add spaces so that you can actually place information within the block move the cursor to the appropriate location and press the ^A key. For example, if you want to add lines to the header block move the cursor to the Body line and press the ^A key. The Body line (and everything below it) will be moved down one and a blank line will be inserted.

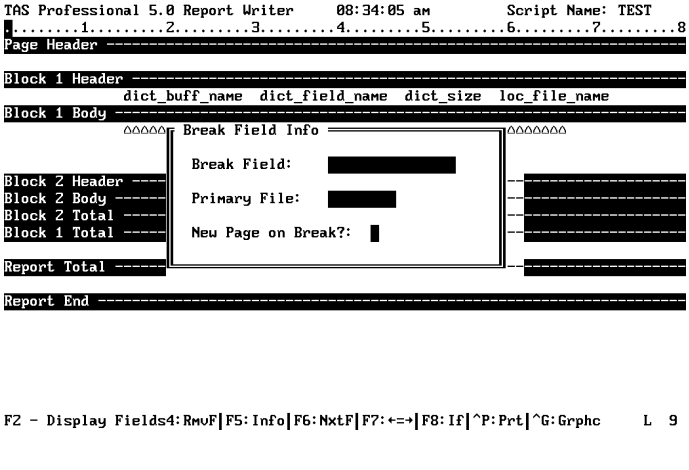
Delete a Block

a) Move the cursor to the Header line for the block you wish to delete.

b) Press the F2 key to get the options menu.

c) Press **D** and then the ENTER key.

d) The block and all characters and fields that were a part of the block will be deleted.



Change Break Info

a) Use this to set up the break field and file for this block. For Block 1 you would use the Chg File Info instead of this.

b) Move the cursor to any of the band lines for the appropriate block.

c) Press the F2 key to get the options menu.

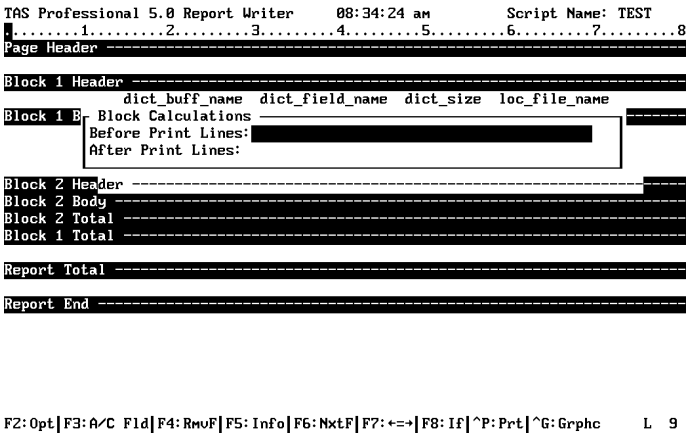
d) Press **C** and the ENTER key. The entry screen above will be displayed.

e) The first entry is the Break Field. During a report, when this field changes value, the program will know that it is finished with this section. When this block is run the program expects that the initial record is active in memory. This will occur if you have set the proper relationships between this file and the appropriate master file in the File Info section (see above). The Break Field does not have to be a key.

f) Next you need to enter the primary file for this block. This would be the file that is also set up in the File Info section. You may have fields from multiple files in one block, but you need to make sure that appropriate relationships are set up so that all the necessary records will be found during the report.

g) The last option is whether or not to go to a new page on break. If you enter **Y** here the report will do a form feed (eject the current page) when the value in the Break Field no longer matches the next record (when the block breaks). The total lines, if any, will be printed first.

h) After the block breaks (all appropriate lines have been printed for that group), the program will return to the next block up (if the blocks are nested), or if sequential, will go to the next block in the format. If the report is at the end, the Report Total lines, if any, will be printed and the report will be finished.



Expressions

a) This allows you to put standard TAS Professional 5.1 expressions directly into your report. You may attach them to any block band line and you may specify whether they are to be calculated before or after the lines are printed. The results may be used in any line.

b) Move the cursor to the appropriate block band line.

c) Press the F2 key to get the options menu.

d) Press **E** and the ENTER key. The entry screen above will be displayed.

e) You can move the cursor to the After Print Lines entry simply by pressing the ENTER key. To return to the main screen press the ENTER key again or the ESC key.

f) Enter the expression in standard TAS Professional 5.1 format. For example, to count the number of records printed, you might use:

```
counter=counter+1 or
inc counter
```

To total a value into a holder you might use:

```
total_sales=total_sales+this_sale
```

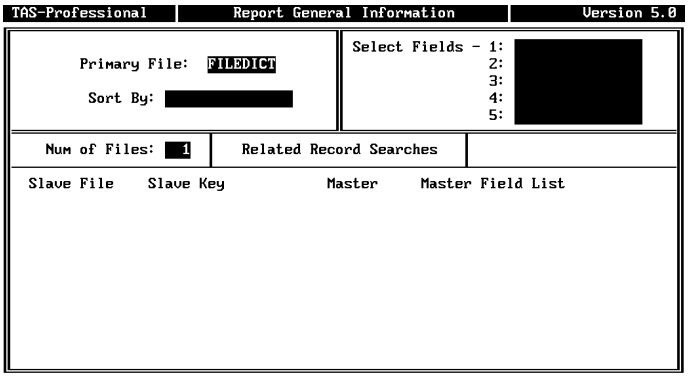
You can also do multiple expressions by separating them with semi-colons (;), i.e.,

```
counter=counter+1; total_sales=total_sales+this_sale
```

g) Since each block band can have its own expressions you will need to return to the exact band line if you wish to make changes to entries made previously.

h) To delete an entry just clear the line with the ^U key or delete just the appropriate expression with the DELETE key.

i) You will be able to see where the expressions appear in the program after you have created it either by loading the source file with a text editor or through the use of the Program Editor.



Change File Information

a) This information is generally entered the first time when you are finished creating a new report format and you have gone through the initial saving stages. You can also change that information at any time.

b) Press the F2 key to display the options menu.

c) Press **C** three times (the first time will get you to the Copy Block line, the second Chg Defaults) and the ENTER key to choose the Chg File Info option. The screen above will be displayed. The Primary File is the main or master file for the program. If all the fields placed are from one file only then this will automatically be that file name. The Sort By option is the name of the key (index) to be used when accessing records from this file. If you leave this field blank the compiled report program will ask the user to choose an index at runtime. The Select fields option allows you to specify the fields to be used as From/Thru selections. The second part of the screen displays the files used and allows you to set up relations between those files.

d) When the cursor is in the Primary file entry field you can press the F2 key to get a list of the files used in the program. Choose one.

e) To designate an index to be used you can enter that key name here. To get a list of keys available for the Primary file press the F2 key. If you leave this option blank the program will ask for a key to be chosen at runtime.

f) Next you will be asked for Select fields. For each field you enter here the report will ask for a From/Thru value to be entered by the user at runtime. The report will then limit the print-out to those values, if any are entered. If no entry is made then the report will not limit the output by that field.

g) Next you will be asked to set up the relationships between/among the various files. Even though it is listed you will not be able to choose the Primary file. Move the cursor bar to the file you wish to choose and press the ENTER key. The cursor bar will be removed and a standard entry field will be displayed under the Slave Key title. Here you need to enter the key or index name that will be used to find records in this file when you find a record in the master file. You have several choices of things to do here. You can: Press the F2 key to get a list of relations already set up for this file; Press the F3 key to get a list of indexes for the file; or Press the F5 key to tell the report program to find the first record in this file when the report is run. This would mean that this file was not connected to the master file at all.

h) Next enter the Master file name. If there is only one other file then this would be that file name. However, you can have multiple relationships; not all files have to be related to the master. To get a listing of all files press the F3 key. Enter an appropriate file name.

i) Enter the fields in the Master that correspond to the key information in the Slave. This means that when the Master record is found the program will use the data in these fields when searching for the Slave record. If there are multiple fields then separate them with commas.

k) Continue setting up relationships for all other files. When you are finished, press the ESC key. If this is a new script the saving process will complete. If this is an old file that you are modifying, the program will return to the main screen.

Save Report Format Script

a) When you are finished setting up or changing your format press the ESC key.

b) The program will ask if you want to save the script file for this format. Enter **Y** if you do.

c) The program will then display a window in which you should enter the script name. If you have edited an existing file the name will be displayed automatically. You can put in a different name if you wish to save a new file.

d) After you enter the name the program will save the file and return to the original screen. If this is a new format and there are fields from at least one file the program will display the File Information screen (as above) and ask for the appropriate data.

Report formats -> Make program

You reach this option by placing the cursor on the **Report formats** option line and pressing the ENTER key or press the **S** key. Then press the **M** key or move the cursor to the **Make program** option line and press the ENTER key to run the program.

Program name: TASMKRPT.RUN

This program creates the appropriate code from the screen format scripts created in TASEDRPT.RUN.

If this program is called from TASEDRPT the name of the format will already be on the screen. If it isn't you will be able to enter the format name.

The program will then ask you if you wish to create the appropriate source code. If you answer **Y** the program will also ask you if you want to compile the program after the creation process is complete. If you answer **Y** again the program will be created. If any errors have occurred during the compilation process (included files not found, UDFs not found, etc.) then you will be informed of that. Also, a list of any errors will be written out to file rptfmt.ERR where rptfmt represents the name of the format compiled. If the program compiled correctly you will need to return to the Main Menu before you are able to run the program.

If you create a source file and one already exists the program will alert you to that and you will have to enter **Y** at the appropriate question to create the program.

NOTE: If the report is wider than 80 columns the program will automatically bind in the code necessary to allow the user to scroll back and forth when it is printed to the screen.

report/2 formats -> Edit report/2 format

You reach this option by placing the cursor on the **report/2 formats** option line and pressing the ENTER key or by pressing the **2** key. Then press the **E** key or move the cursor to the **Edit report/2 format** option line and press the ENTER key to run the program.

Program Name: TASRPT.RUN

This program is similar to the standard Report Format except that it creates code that is accessed in your program rather than creating a separate stand-alone program.

The concept behind this program is to create a process whereby you can easily and quickly create very complex reports and maintain the ability to make changes to them just by changing information in one of the blocks or bands.

In these report formats the blocks can be considered stand-alone objects. The block is called by addressing the name of the block as a UDF. If whatever the block is supposed to do happens properly, i.e., the user doesn't press the ESC key, etc., then the result of the UDF is .T. If it doesn't, the result is .F. Given this result you can determine what to do within the program. For example, if you have named a block BODY_LINES you might say in a program:

```
IF .N. BODY_LINES() GOTO EXIT_PROG
```

This would tell your program that if the UDF BODY_LINES() returned .F. then it should transfer control to the line EXIT_PROG; otherwise continue with the next line in the program. The UDF BODY_LINES() may actually be several lines of printing, it may include print_if options, or expressions, etc. However, with this process you need only call the entire block by this one command. This allows you to change the actual print-out by just changing the format, recreating the program (using the Report/2 Formats -> Make Program option (TASMKRPB.RUN)), and recompiling. You wouldn't have to make any actual code changes yourself. This is, in a sense, object-oriented programming.

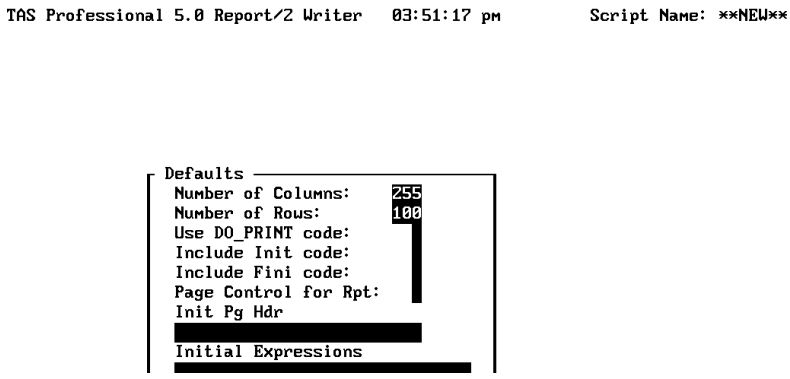
There are several differences between the standard Report Format program and the Report/2 Format program: Report/2 does expect you to control the files directly so that all data that is needed is there when the block is called. You can have up to 30 different block or bands. They are not interdependent on each other at all. Any block can be the header, and there are no specific sub-headers or sub-totals. You can name each block and there are several other differences within the block which will be explored later. As in the other Report Format program there are a maximum of 100 format lines, and each line cannot be more than 255 characters. You do not specify file names, keys, relationships, etc. since that would be set up within your report program. This format creates just the actual report itself, but allows much more flexibility than the other.

The files created by the Report/2 Format program have the extension .SRB.

Edit report/2 format

Control Keys

For information on the control keys used with the Report/2 format editing program, please refer to the section on control keys under **Edit report format** earlier in this chapter.



Edit report/2 format

Defaults Screen

These are the initial entries when you are creating a new script. If you want to make changes to a current script then move the cursor to a non-band line, press the F2 key and choose Chg Defaults from the menu.

Number of columns, number of rows - maximum size of report. Current maximum size is 255 x 100. Each row can be a different format line. Each row can be repeated many times on one page or over the entire report. The rows do not necessarily correspond to the same line on the paper. These 2 values are the only ones that cannot be changed once accepted during the initial entry.

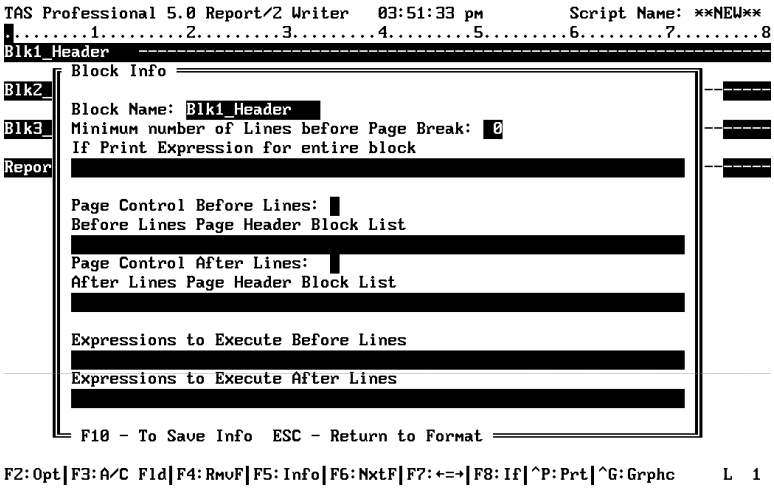
Include Init Code - If you answer **Y** here the program will create the appropriate initialization code. This includes the MOUNT command and other setup type lines.

Include Fini Code - If you answer **Y** here the program will create the code to finish the report. This includes the final QUIT command.

Page Control for Report - You can choose to do a page break before the report (**B**), at the end of the report (**E**), both beginning and end (**Y**), or no special page break (**N**).

Init Pg Hdr - The is the name of the block/band that will be the initial header lines for the report.

Initial Expressions - These are expressions you want executed before the report starts. You can have multiple expressions separated by semi-colons (;).



Edit report/2 format

Block Information Screen

These are the initial entries when you are creating a new block/band or changing an existing one. If you want to make changes to a current block then move the cursor to the appropriate block line, press the F2 key and choose Chg Block Info from the menu.

Block Name - This is what you want to call this block of lines. Any 14 character name is acceptable; no spaces are allowed within the name. You will use this name as the UDF within your program.

Minimum number of lines before page break - You can use this value to determine how many lines must be remaining on the page before this block can be started. By using this feature you can be sure that a line won't be orphaned on the previous page if that would not be acceptable.

If Print Expression for entire block - If you want to control whether or not the entire block executes (printing, expressions, etc.), enter a standard TAS Professional 5.1 expression here.

Page Control Before Lines - This would occur before any lines are printed: **Y** - Do a page break before lines and print the appropriate header / **T** - Move to the beginning of the next page but don't print any header lines / **H** - Print header lines now, don't move to beginning of next page / **N** - No special page control.

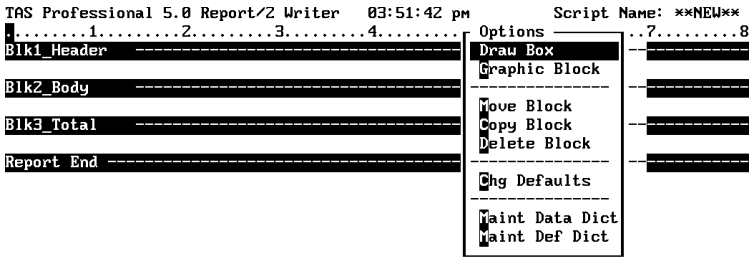
Before Lines Page Header Block List - This is a list of block names separated by semi-colons (;) that would make up the new page header (in order) before any lines are printed. This new header would be used if a new page was required any time during the printing of the lines.

Page Control After Lines - This would occur after any lines are printed: **Y** - Do a page break after lines and print the appropriate header / **T** - Move to the beginning of the next page but don't print any header lines / **H** - Print header lines now, don't move to beginning of next page / **N** - No special page control.

After Lines Page Header Block List - This is a list of block names separated by semi-colons (;) that would make up the new page header (in order) after all lines are printed. This new header would be used if a new page was required any time after the printing of the lines.

Expressions to Execute Before Lines - Standard TAS Professional 5.1 expressions to be executed before any lines are printed. Separate expressions with semi-colons (;).

Expressions to Execute After Lines - Standard TAS Professional 5.1 expressions to be executed after any lines are printed. Separate expressions with semi-colons (;).



Edit report/2 format

Options Menu

Press the F2 key when the cursor is on a regular report format line to display this options menu.

Draw box - Place a box on the screen - same as Report Format

Graphic Block - Mark a block and then choose a graphic character. This will fill the block with the chosen character. It will not overwrite any fields within the block - same as Report Format.

Move block - Choose a block and then move the cursor to the new upper left corner location. The block may contain fields - same as Report Format.

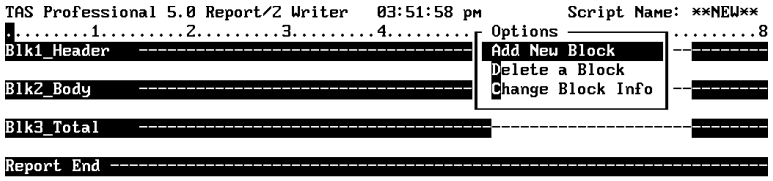
Copy block - Choose a block and then move the cursor to the new upper left corner location. The block may contain fields. On a report format you can have multiple occurrences of the same field - same as Report Format.

Delete block - Delete a block from the screen. The block may contain fields - same as Report Format.

Chg defaults - Change the default entries made when the Report/2 format was first created.

Maint data dict - This will chain to the File Data Dictionary Manager so that you may make changes directly to a file definition. All fields must be in the data dictionary or the defined data dictionary before they can be added to the format, or they must be defined as 'MEMORY' fields - same as Report Format.

Maint def dict - This will chain to the Defined Data Dictionary Manager so that you may make changes directly to fields that are defined there. All fields must be in one of the data dictionaries before they can be added to the format, or they must be defined as 'MEMORY' fields within the screen - same as Report Format.



Edit report/2 format

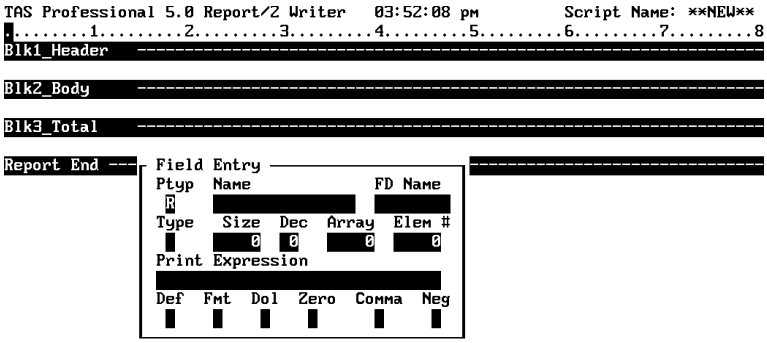
Block option menu

Press the F2 key when the cursor is on a block or band line to display this options menu.

Add a new block - Add a new block immediately before the current one - same as Report Format.

Delete a block - Delete a block, including all lines currently part of that block - same as Report Format.

Change Block Info - Change the control information for that block - same as Report Format.



Edit report/2 format

Field Info Screen

This screen is displayed when you press the F3 or F5 keys.

Ptyp - Print type - if this is a regular field then enter R, if an expression enter X.

Name - The name of the field. You can press the F2 key and get a list of fields in the file data dictionary, press F3 and you get a list of fields in the defined data dictionary.

FD Name - The file descriptor name. If this is a field from the data dictionary this will be the FD name for that field. If this is a defined (or to be defined) field this will be MEMORY.

Type, Size, Dec - Standard field specs.

Array Sze - If this is a MEMORY field and has not been used before in the format you can specify a number of array elements, if applicable. If this is a file field and it is an array field then the number of elements is placed here.

Element num - If the Array Sze for this field is <>0 then you have to specify the element number.

Print Expression - If Ptyp='X' then you will be able to enter the expression here. The results of this expression will be printed at the appropriate location. The program doesn't check to make sure that everything will work properly; that is your responsibility.

Def - Define Field - If the FD is MEMORY and you enter **Y** here the appropriate DEFINE code will be created when you create the code for this script. This would mean that you don't access this field outside of this report.

Fmt - Format Field - If DEF='Y' and the type of this field is N, T or D you can enter **Y** here to create the code to format the data.

Dol - Dollar Sign - If FMT='Y' and type='N' then enter **N** here to not add a floating dollar sign in the field.

Zero - Zero values - If FMT='Y' then enter **N** here to not print zeros if the value of the field is zero. The result would be just a blank.

Comma - If FMT='Y' and type='N' then enter **N** here to not add commas in the field.

Neg - Show Negative Value How - **L** - Leading minus sign / **T** - Trailing minus sign / **P** - Parentheses around value / **C** - Trailing CR / **A** - Angle brackets.

Edit report/2 format

Operations

This section will explain in outline form how to accomplish the various functions that are available in this program.

Adding a new field to the report

a) Move the cursor to the appropriate location. The field display will start at this location and extend to the right. You need to make sure that the field won't overlap any other fields.

b) Press the F3 key and the Field Info window will be displayed. Enter the appropriate data.

d) The program will ask if all is correct. If you answer **Y** the field will be placed on the screen. The program uses a symbol that looks like a small house for each display character in the field.

e) To quit the entry window press the ESC key at any time. The program will ask if you want to return to the full screen; enter **Y**.

Removing a field from the report format

a) Move the cursor to any position in the field you wish to remove.

b) Press the F4 key.

c) The program will ask if you wish to remove the field; enter **Y**.

d) The field representation will be removed from the screen.

Inserting a line into the report format

a) Move the cursor to the line above which the new line is to be inserted. The new line will be added at the cursor and all lines will be moved down one.

b) Press the ^A key.

c) A blank line will be added at the cursor. The band line Report End will always be the last line in the report. If you don't have enough extra rows (set in the Defaults entry as the number of rows) you will be alerted to this fact and will not be able to add a line.

Deleting a line in the format

a) Move the cursor to the line you wish to delete.

b) Press the ^D key.

c) The line will be deleted and all further lines will be moved up 1. If there is a field on the line to be deleted you will be alerted to that fact and will have to enter **Y** to 'are you sure' question to continue. You cannot delete a band line.

Setting up a Print If expression

a) Use this option to control whether or not a line will print.

b) To control whether a block will print or not move the cursor to the block line, press the F2 key, choose **C** (Change Block Info), and enter the If Print expression there. To control the printing of a single line press the F8 key when the cursor is on that line.

c) Move the cursor to the appropriate line.

d) Press the F8 key.

e) Enter a standard TAS Professional 5.1 expression, such as:

```
sales_amount < 0
```

In this example this line will print only if the sales_amount value is less than 0.

f) Press the ENTER key to save the value and the program will return to the main screen. To make changes in that expression move the cursor to the appropriate line and press the F8 key again. The expression will be displayed and you can make changes to it if desired. To delete the expression press the ^U key and then the ENTER key so that the expression line is blank.

Placing a box on the screen

a) Move the cursor to where you want the upper left hand corner of the box to begin.

b) Press the F2 key to display the options menu.

c) Press **D** or the ENTER key to choose the Draw Box option.

d) Now move the cursor to the lower right hand corner of the box and press the ENTER key. To interrupt the process and return to the normal window press the ESC key. You cannot cross a band line.

e) A window will appear that displays the start and stop coordinates.

f) Enter the box type: **0** - Clear a box already on the screen; **1** - Use single line characters, **2** - use double line characters, **3** - Use double line characters for the horizontal lines and single line characters for the vertical, **4** - Use single line characters for the horizontal lines and double line characters for the vertical.

g) Enter the color to use. If you wish to use the background color enter **0**. You can also get the available color values by pressing the F2 key.

h) The box will be displayed (or cleared) and the regular window will be returned with the cursor at the upper left hand corner of the box.

NOTE: To draw a line vertically or horizontally just move the cursor down without moving it to the right, or to the right (for a horizontal line) without moving it down.

Create a block of graphic characters

a) Move the cursor to where you want the upper left hand corner of the block to begin.

b) Press the F2 key to display the options menu.

c) Press **G** and the ENTER key to choose the Graphic Block option.

d) Now move the cursor to the lower right hand corner of the block and press the ENTER key. To interrupt the process and return to the normal window press the ESC key. You cannot cross a band line.

e) A window will appear that displays the start and stop coordinates along with a menu of graphic characters.

f) Enter the character to use by moving the cursor to the appropriate line and pressing the ENTER key.

g) The program will ask if all is correct. If you enter **Y** the block will be filled with the character and you will return to the regular window with the cursor at the upper left hand corner of the block. If you answer **N** you will return to the normal window with nothing changed.

Move a block

a) Move the cursor to the upper left hand corner of the block to be moved.

b) Press the F2 key to display the options menu.

c) Press **M** and the ENTER key to choose the Move Block option.

d) Now move the cursor to the lower right hand corner of the block and press the ENTER key. To interrupt the process and return to the normal window press the ESC key. You cannot cross a band line while choosing a block; however, you can move the block anywhere.

e) Now move the cursor to the new location of the upper left hand corner of the block. The program will make sure you don't move anywhere on the screen where the block wouldn't be able to fit. However, if all the program needs to do is add extra lines to the band block it will do so automatically.

f) Press the ENTER key to designate the cursor location as the spot to move the block. The program will check for other fields in the area that might be overwritten by the move and will not allow you to move the block if there would be a problem.

g) The cursor will return to the location of the original upper left hand corner of the block.

Copy a block

a) Move the cursor to the upper left hand corner of the block to be copied.

b) Press the F2 key to display the options menu.

c) Press **C** and the ENTER key to choose the Copy Block option.

d) Now move the cursor to the lower right hand corner of the block and press the ENTER key. To interrupt the process and return to the normal window press the ESC key. You cannot cross a band line while choosing a block; however, you can copy the block anywhere.

e) Now move the cursor to the new location of the upper left hand corner of the block. The program will make sure you don't copy anywhere on the screen where the block wouldn't be able to fit. However, if all the program needs to do is add extra lines to the band block it will do so automatically.

f) Press the ENTER key to designate the cursor location as the spot to copy the block. The program will check for other fields in the area that might be overwritten by the block and will not allow you to copy the block if there would be a problem.

g) The cursor will return to the location of the original upper left hand corner of the block.

Delete a block

a) Move the cursor to the upper left hand corner of the block you want to be deleted.

b) Press the F2 key to display the options menu.

c) Press **D** and the ENTER key to choose the Delete Block option.

d) Now move the cursor to the lower right hand corner of the block and press the ENTER key. To interrupt the process and return to the normal window press the ESC key.

e) The program will ask if you're sure you want to delete the block. To do so press **Y**.

f) If you choose to delete the block it will be removed from the screen including any fields and the color will be replaced with the default.

g) The cursor will return to the location of the original upper left hand corner of the block.

The following options are available when you place the cursor on a band or block line and press the F2 key.

Add New Block

a) Use this option to add a new block immediately before the cursor location.

b) Move the cursor to the block line above which the new block is to be inserted.

c) Press the F2 key to get the options menu.

d) Press **A** and then the ENTER key.

e) The Block Information Screen (as above) will be displayed and you can enter the appropriate information for this block.

f) To add spaces so that you can actually place information within the block, move the cursor to the appropriate location and press the ^A key.

Delete a Block

a) Move the cursor to the line for the block you wish to delete.

b) Press the F2 key to get the options menu.

c) Press **D** and then the ENTER key.

d) The block and all characters and fields that were a part of it will be deleted.

Change Block Info

This will allow you to change information entered previously for this block. Please refer to the discussion of the Block Information Screen earlier in this section.

report/2 formats -> Make program

You reach this option by placing the cursor on the **report/2 formats** option line and pressing the ENTER key or by pressing the **2** key. Then press the **M** key or move the cursor to the **Make Program** option line and press the ENTER key to run the program.

Program name: TASMKRPB.RUN

When you exit the Report/2 format editing program, it will ask if you wish to save the script. If you answer **Y** the program will then ask if you wish to create a program from the script. If you answer **Y** again the program will automatically chain to TASMKRPB.RUN and will create the appropriate .SRC code with the same name as the script.

You can also create the source code separately by choosing the Report/2 option from the Program menu, then choosing the Make Program option from the sub-menu as described above.

> **NOTE:** You must include the code created in your own program either using the #INC compiler directive or actually adding it to the appropriate source code file.

We have included a simple example of the new report format. The main source code is PRTTEST.SRC, the Report/2 format is PRTTEST2.SRB, and the source code from that format is PRTTEST2.SRC.

List formats -> Edit List Format

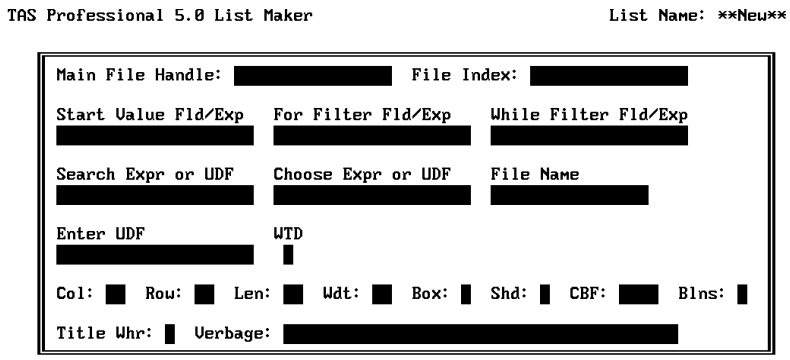
You reach this option by placing the cursor on the **List formats** option line and pressing the ENTER key or by pressing the **L** key. Then press the ENTER key again to run the program.

Program name: TASEDLST.RUN

This program helps to automate the creation of scrolling list/choice boxes displaying records from data files using TAS Professional 5.1's LISTF command.

When you specify the titles that will appear on the columns in the LISTF commands, the spacing can become very tricky. If the title is longer than the field, you need to put extra spaces in the field. When

the title is shorter than the field, you need to space that out. As the LISTF command becomes more complex, you must make certain that all the appropriate information is specified. This program helps you specify all the required data.



Edit list format

Main Screen

This is the main screen for the List. The fields and their meanings are as follows:

Main File Handle - The name of the field that contains the FNUM (file handle number) value for the main file to be used in this list.

File Index - The name of the index to use or the the key number. A key number may be either a numeric constant or contained within a numeric field. If you use the key number instead of the actual index name, you must preface this value (field name or constant) with the @ symbol. If you don't want to use a key at all enter @0.

Start Value, For Filter, While Filter, Search Expression and Choose Expression are the same as explained in the command information for LISTF (see **Chapter 5, Command Reference** for more information about the **LISTF** command).

File Name - If you want to allow the user to choose an index at runtime then enter the actual main file name here. The program will then use that when displaying the available indexes if the user presses the F3 key when the list is displayed.

Enter UDF - This is the UDF you have set up elsewhere as the entry routine for this list.

WTD - What to Do - If you have specified an Enter UDF name above the program will allow you to choose what the user will be able to do. The options are: 1 - Change Records; 2 - Add Records and 4 - Delete Records. Determine the correct entry by adding the options together. To allow all three enter 7, to just allow changes and deletions enter 5.

Col (column), Row, Len (length), Wdt (width) - Location and size of window to use with this list.

Box - The box type: S (single line), D (double line), C (custom) or leave it blank for no box.

Shd - If you want to display a shadow behind the list window then enter R (right) or L (left) here.

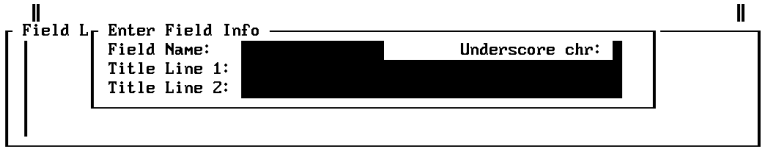
CBF - Characters Between Fields - You can specify up to 4 characters that will be put between each field on the list. If you want a space after the field but before the character then leave the

first character blank. The program will automatically put a space after any characters you enter.

Blns - Blank Lines - If you want 1 or more blank lines between the top of the window and the first line in the list (either a regular list line or a title line) then enter that number here.

Title Whr - If you wish to include a title for the window then this is where to put it. L - left, R - Right or C - Center.

Verbage - The actual title.



Edit list format

Field Entry Screen

This is the screen you will see when adding a new field or changing/deleting an existing field in the field list.

Field Name - The actual field name you want to include in the list. This can also be an expression, however, the total size (including parentheses) may be no greater than 15 characters.

Underscore character - The character to be used under the title. This defaults to the standard PC line character.

Title Line 1 & Title Line 2 - Each field can have up to 2 lines of titles. This is only a limit in the List Maker; in the LISTF command, you can have as many title lines as the length of the window, if desired. If either or both of the title lines are not used for all fields then nothing will be displayed and the space will be used for actual list lines.

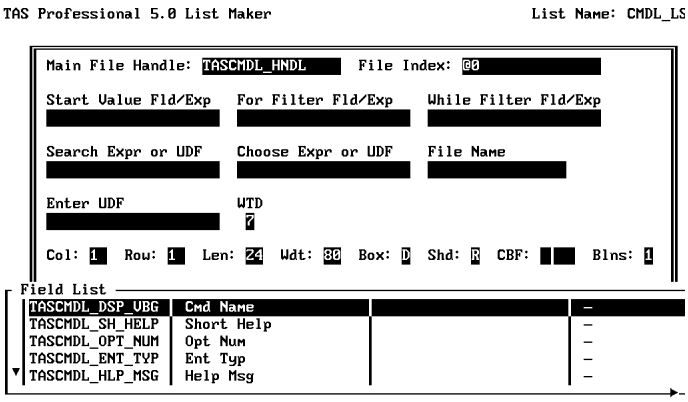
Edit list format

Operations

When you first choose the List Formats/Edit List Format option from the Program Menu a standard Enter Script Name window will be displayed. Press the F2 key to get a list of .STL files available. To enter a new list just leave the script name blank and press the ENTER key.

The program will take you through the options available (see Main Screen Fields above). After entering all the options the program will return, automatically, to the Main File Handle field. To save the script you must press the F10 key. This can be done at any time.

To get a list of the fields that are a part of this script press the F3 key at any time.



Adding fields to the list

a) If you aren't already in the field list then press the F3 key to display it (see example above).

b) Press the ^END (ctrl+END) key to go to the next blank line and execute the enter routine.

c) You will then be able to enter the Field Name, underscore character and title lines as appropriate. You may press the F2 key when in the Field Name entry to get a list of available fields.

d) When you are finished with your entry either press F10 to save it or press ENTER until the program asks if the entries were correct.

e) The program will return to the main list. To return to the main screen press ESC.

Deleting a field from the list

a) If you aren't already in the field list then press the F3 key to display it.

b) Move the cursor bar to the correct field entry.

c) Press the DEL key.

d) The program will ask you to confirm the deletion. If you enter **Y** the field will be deleted from the list.

e) The program will return to the main list. To return to the main screen press ESC.

Inserting a field into the list

a) If you aren't already in the field list then press the F3 key to display it.

b) Move the cursor bar to the field entry before which the new field will be added. The new field will be placed immediately before the field highlighted by the cursor bar.

c) Press the INS key.

d) The balance of the entry is identical to adding a field as described above.

e) The program will return to the main list. To return to the main screen press ESC.

Changing a field in the list

a) If you aren't already in the field list then press the F3 key to display it.

b) Move the cursor bar to the correct field entry.

c) Press the ENTER key.

d) You will be able to change any of the values for that entry. Press F10 when you are finished.

e) The program will return to the main list. To return to the main screen press ESC.

List formats -> Make List Program

You reach this option by placing the cursor on the **List formats** option line and pressing the ENTER key or by pressing the **L** key. Then select the second option to run the program.

Program name: TASMKLST.RUN

This program creates the actual list code (a .SRC file with the same name as the script).

When you exit the Edit List Program procedeure, it will ask if you wish to save the script. If you answer **Y** the program will then ask if you wish to create list code from the script. If you answer **Y** here the Edit program will automatically chain to this Make routine and will create the appropriate .SRC code.

You can also create the source code separately by choosing the List formats option from the Program menu, then choosing the Make List Program option from the sub-menu.

> **NOTE:** You must include the code created in your own program by using the #INC compiler directive or actually adding the code created here to the appropriate program.

We have included a simple example of the new list format. The list format is LIST_TST.SLT, and the source code from that format is LIST_TST.SRC.

Edit Program

You reach this option by placing the cursor on the **Edit program** option line and pressing the ENTER key or by pressing the E key.

Program name: TASEDPRG.RUN

You may enter a program name at the initial prompt or just press the ENTER key for a new file.

If the program exists it will be loaded and the lines displayed. If a program doesn’t exist two remark lines will be displayed. You may change one or both, and you may insert lines before the first line; however, you must have at least one line for each program.

In TAS Professional 5.1, the source file size for the program editor is variable. The editor uses the value in the file TASEDPRG.SZZ which needs to be in the same sub-directory as the .RUN file. The only value in that file should be the size of the source file buffer in bytes. If you have no value there, the program uses a default value of 100,000. If you attempt to load a file larger than the buffer, an error will be displayed and the program will exit. Please note: if you set this value, make sure it's at least 100 bytes larger than the largest program you might edit.

Edit Program

Control Keys

NOTE: When the character '^' precedes the key name it means to press the CTRL key at the same time you press the other key.

Dn Arrow - Cursor moves down one line in the program.

Up Arrow - Cursor moves up one line in the program.

Pg Up - Move up 16 lines in the program.

Pg Down - Move down 16 lines in the program.

Home - Move to the first line currently on the screen.

^Home - Move to the first line in the program.

End - Move to the last line currently on the screen.

^End - Move to the last line in the program.

Left Arrow - Mark the current line so that you can easily return to it, or clear the mark if it is already set.

Right Arrow - If a line is marked this will go directly to that line. If not the program will ask for a label name or line number.

NOTE: If there are no labels in the program the cursor will go directly to the line number option.

Insert - Turn on the insert mode. The cursor will appear as a slightly larger block. If insert mode is already on and you press the Ins key it will be turned off (go back to change mode). If you press the ENTER key during insert mode the program will insert lines at the current cursor location. If you press the ENTER key during change mode the program will allow you to change the line at the current cursor location.

Delete - Delete the current line.

F2 - If a block is set then this will ask what file name to write the block to. If it is not set then this will ask what file to read. If you are reading the lines will be inserted starting at the current cursor location.

F3 - Clear a currently set block.

F4 - Delete a currently set block.

F5 - Mark the beginning of a block.

F6 - Mark the end of a block.

F7 - Copy a marked block to another location.

F8 - Move a marked block to another location.

F9 - Search and replace.

F10 - Save the program back to disk.

^A - Display all of the line in a window at the bottom of the screen. You can go to the next and previous lines by pressing the DOWN and UP ARROW keys respectively. This will allow you to see the entire line even if it extends beyond the edge of the window.

^D - Chain to the defined field data dictionary manager.

^E - Edit the current line directly using the TASEdit program. This will allow you to make changes to a line without having to go through the menu system. To exit from editing a single line press the ESC or ENTER key. If you press the ESC key the program will exit back to the main program and will not update the line. If you press the ENTER key, the changes made will be reflected in the current line and you will be automatically placed on the next line ready to edit that one.

^F - Chain to the regular file data dictionary manager.

^L - Display a list of label names.

^P - Print out the program. If the lines will not fit within the screen or on the page they are automatically wrapped. The program puts continuation characters at the end of the lines as required. This is strictly for print-out purposes only, and the lines are not broken up within the program itself. If you mark the beginning and end of a block and press the ^P key to get a print-out, the program will use the beginning and ending lines of the block as the defaults for the lines to be printed.

^R - Remark/Unremark the current line. If the line is currently a non-remark line and you press ^R the program will insert a semi-colon ';' at the front of the line, effectively making that line a remark. If you press the ^R key on a remark line the semi-colon will be removed.

^S - Get a list of Screen and Report formats in the program. This will allow you to edit the format or create a new one.

Edit Program

Information Line

There is a line of information that is displayed just above the window that shows the program code. The items on that line are:

Line - This is the location of the cursor in the program, the physical line number.

Spc Remain - The amount of bytes remaining. This will keep you informed as to how much of that you have left. This value changes each time you add, change or delete a line. The maximum number of lines that can be edited in this program is 10,000.

BegB - If you have set a block this is the beginning line number.

EndB - If you have set a block this is the ending line number.

M/L - If you have marked a line (by pressing the LEFT ARROW key) this is the line number.

CHG/INS - This last item will change to indicate whether you are in Change (CHG) or Insert (INS) mode. Also, in Change mode the cursor is a flat line; in INS mode it is a small box (half block cursor).

Edit Program

Menu Tree

The following is a tree diagram of the menu options. The commands are at the end of the line. These are not necessarily the actual command names but what is listed in the editor.

PRG CONTROL	CASE	SELECT
		CASE
		OTHERWISE
		ENDC
	FOR	FOR
		EXIT
		EXIT_IF
		LOOP
		LOOP_IF
		NEXT
	IF	IF
		IFDUP
		IFNA
		ELSE
		ELSE_IF
		ENDIF
	WHILE	WHILE
		EXIT
		EXIT_IF
		LOOP
		LOOP_IF
		ENDW
	GOTO/GOSUB	GOTO
		GOTO LINE
		GOSUB
		GOSUB LINE
		ON
		RET
		POP STACK
	TRAP	TRAP
		PUSH TRAP
		POP TRAP
		XTRAP
	UP ARROW	
	LINE LABEL	
	GET LABEL LN#	
FIELD	EQUAL	
	CREATE/CHG	DEFINE
		ADD
		ALLOCATE
		REDEFINE
		DEALLOCATE
		FORMAT
	ARRAY	ALLOCATE
		REMOVE
		UPDATE
		SORT
		DISP ARRAY
	ALPHA FLD CMDS	MID (STUFF)
		DELETE CHRS
		FILL
		TRIM
		UPCASE
		INSERT

	POINTER		
	MOVE DATA		
	INCREMENT		
	DECREMENT		
	PUSH FIELD		
	POP FIELD		
FILE	OPEN/CLOSE	OPEN	
		OPEN VARIABLE	
		CLOSE	
		OWNER	
		REOPEN	
		SET ACTIVE	
	MULT REC CMDS	DELETE ALL	
		IMPORT	
		EXPORT	
		READ ARRAY	
		WRITE ARRAY	
		REPLACE	
		LIST RECORDS	
		SCAN	SCAN
			EXIT
			EXIT_IF
			LOOP
			LOOP_IF
			ENDS
	FIND	FIND	
		FIND VARIABLE	
		RECORD NUMBER	
		SEARCH FILE	
		RELATE	
		FILTER	
	SAVE		
	DELETE		
	CLEAR RECORD		
	READ		
	WRITE		
	TRANSACTIONS		
	UNLOCK ALL		
USER INTERFACE	ENTER		
	COLOR CONTROL	BACKGROUND	
		COLOR	
		REVERSE	
		FOREGROUND	
		PAINT	
	WINDOWS	WINDOW	
		DEFINE WIND	
		ACTIVATE WIND	
		SAVE SCRN	
		REDSPLY SCRN	
		FORCE	
	LISTS	FILE LIST	
		ARRAY LIST	
		AUTOINC	
		AUTODEC	
		AUTOENTER	
		REDISPLAY LIST	
		NO REDISPLAY	
		NO RESTART	
		EXIT LIST	
		SET FIRST LINES	
	MESSAGES	PRINT MESSAGE	
		DISP MESSAGE	
		ASK	
		ERROR	
		CLEAR PRG ERROR	

	SCREEN CONTROL	MOUNT
		SAY
		CLEAR SCREEN
		CLEAR LINE
		CLEAR SCRN FLDS
		CURSOR
		RESET SCREEN
		PRINT ALL
		PRINT BOX
		SCROLL
		SCREEN CHR
		SCRN L/U/R/S
		REMOUNT
	4.0 NMENU	
	3.0 MENU	
	WRAP	
	REWRAP	
	REENT	
	NO VALID MSG	
REPORTS	PRINTER	SETUP
		NUMBER
		DRIVER
		SHOW PRT LINE
	MOUNT	
	CLOSE PTD FILE	
	PRT FORMAT	
	PRT BLANK LNS	
	PRT CHR STRG	
	PRT ON	
	PRT TOP OF FORM	
	PRT VERT TAB	
SYSTEM	OTHER PROGRAMS	CHAIN
		CHAIN RAP
		RAP
		PARAMETERS
		EXECUTE PROGRAM
	FILE COMMANDS	DELETE FILE
		RENAME FILE
		INIT FILE
	PROGRAMMING	COMPILE PROG
		COMP DIRECTIVE
		BREAK
		UDC
		DO UDC
		UDF
		REMARK
		CHG DICT PATH
		INTERRUPT
		PEEK
		POKE
		TRACE
	DATE/TIME	DATE
		TIME
		CLOCK
	AUTO-RUN	
	BELL	
	COMPANY CODE	
	KEYBD UP CASE	
	MOUSE	
	QUIT	
	SOUND	
3.0 COMMANDS	A - CLS NON-TAS	
	B - DISP MEM	
	C - EQU DAY	
	D - EQU MID	
	E - EQU MNTH	
	F - FILL MEM	

G - FORCE3
H - MEM PTR
I - MEM SPACE
J - OPN NON-TAS
K - PUT FLD
L - RD RECORD
M - REDSP3
N - RUN
O - SAVES3
P - SORT3
Q - SSPCF
R - WRAP3
S - WT RECORD

AGAIN - DO LAST COMMAND AGAIN
CMD LIST - A FAST SEARCH LIST OF COMMANDS

Edit Program

Operations

Changing a line

a) Move the cursor to the appropriate line.

b) Make sure you are in change mode. The cursor should be a small line and CHG should be at the end of the information line. Press the ENTER key.

c) A window will be opened at the top of the screen and the full line will be displayed. You will have the option to **E** - Edit, **D** - Delete, or **R** - Replace the command line. The default is **E** - Edit. To edit the line press the ENTER key. You can change to another line at this time by pressing the UP or DOWN ARROW keys. You can also change to Insert mode by pressing the ^I key (the Choose Menu will be displayed).

d) Another window will be displayed replacing the last one. These are the options available for this command.

e) You may not be able to enter certain options (the cursor will skip them) depending on choices you have made. You may move from one option to another by pressing the ENTER key or the UP or DOWN ARROW key.

f) Each option has its own help message. To display the message, press the F1 key while in the appropriate field. The message will be displayed at the bottom of the screen in a list window. If more information exists than can fit in a single screen you will be able to scroll up and down by using the UP and DOWN ARROW keys. To return to the entry press the ESC key.

g) To save the command entry press the F10 key; to return to the previous window press the ESC key.

h) If you save the command entry the program will automatically move to the next line.

i) To return to the main screen press the ESC key while you are at the Edit choice window.

Inserting a line

a) Move the cursor to the appropriate line. The new line will be inserted immediately before this line.

b) Make sure you are in insert mode. The cursor should be a small block and INS should be at the end of the information line.

b) Press the ENTER key.

c) The Choose Menu will be displayed. You can switch into Change mode by pressing the ^C key. You can also move where the line will be inserted by pressing the ^Z key to move down or the ^Y key to move up. The cursor in the main window will move, showing you the new insertion point.

d) Choose the command either by working through the menu options or by pressing **C** and choosing the command from the FAST SEARCH list.

e) Another window will be displayed replacing the menu. These are the options available for this command.

f) You may not be able to enter certain options (the cursor will skip them) depending on the choices you have made. You may move from one option to another by pressing the ENTER key or the UP or DOWN ARROW key.

g) Each option has its own help message. To display the message, press the F1 key while in the appropriate field. The message will be displayed at the bottom of the screen in a list window. If more information exists than can fit in a single screen, you will be able to scroll up and down by using the UP and DOWN ARROW keys. To return to the entry press the ESC key.

h) To save the command entry press the F10 key; to return to the Choose menu press the ESC key.

i) If you save the command entry the program will automatically move the insertion point down one line and you will be able to insert a new line.

j) To return to the main screen press the ESC key while you are at the Choose menu.

Delete a line

a) Move the cursor to the appropriate line.

b) Press the DELETE key.

c) Answer **Y** to the are you sure question.

Mark a block and block operations

a) Move the cursor to the line that is the beginning of the block you wish to mark.

b) Press the F5 key (nothing will happen until you also mark the end of the block).

c) Move the cursor to the last line of the block.

d) Press the F6 key. A line will be displayed at the left of the code designating the block and the beginning and ending line numbers will be displayed in the information line at the top of the code window.

e) To delete the block press the F4 key.

f) To move the block to a new location move the cursor to the line above which the block will be placed and press the F8 key.

g) To copy the block move the cursor to the line above which the block will be placed and press the F7 key.

h) To write the block to a disk file press the F2 key. The program will ask for the file name.

i) To clear a marked block press the F3 key.

Search and Replace

a) To search for a string of characters in a program press the F9 key.

b) Enter the string you wish to search for. If you want to match case then enter the characters exactly as they should appear in the program.

c) To search from the beginning of the program enter **Y** at the From Begin option.

d) To search from the end of the program instead enter **N** at From Begin and **Y** at From End.

e) To search from the current cursor location enter **N** for both From Begin and From End.

f) If you don't care whether the string you're searching for matches case (upper and lower case characters) exactly then enter **Y** at Ignore Case.

g) If you actually want to replace the string with another then enter **Y** at Replace. You will then be able to enter the characters to use as the replacement and can choose whether you wish to be asked before each replacement.

h) If this is a search, the program will display the found line in the middle of the program window and will display the entire line at the bottom of the screen in a separate window and will ask if you want to search again. If you answer **Y** the program will continue looking for the next occurrence.

i) If this is a search and replace you will be asked if you wish to replace (if appropriate) and then will be asked if you wish to search again.

j) To stop the search press **N** at the search again question. When the program cannot find another occurrence of the string being searched for it will automatically stop and return to the main window.

Get a list of Screen/Report formats in the program and edit one

a) Press the ^S key. A list of screen and report formats will be displayed along with a New Fmt line. The type of format will be to the right of the name, **S** - Screen, **R** - Report, **B** - Report/2.

b) Move the cursor to the appropriate line and press the ENTER key.

If you wish to add a new format then move the cursor to the first line in the list (New Fmt *) and press the ENTER key. You will be asked the type of format and the name to use. When you return from the screen/report editor the format will be automatically added to the source file you are editing (if you answered Y to the save format question in the appropriate editor).

NOTE: If you don't save the source file when exiting from the Editor the format created or the changes made will be lost.

c) The appropriate editor will be loaded automatically.

d) When you exit the editor you will be returned to the main window.

Import a Screen/Report format

Use this option to add a screen/report format that is saved as a separate file on the disk to the source code file currently being edited.

a) Press the ^I key.

b) A window will be displayed with an entry field allowing you to specify the format to be added to this source code file. You must enter the entire format name including the path and appropriate extension. Screen formats - .SCP; Report Formats - .SRP; Report/2 Formats - .SRB. If you press the F2 key you will get a list of formats in the current sub-directory.

c) Once you have entered the format name it will be added to the source program, if it is found. You can now edit that format by pressing the ^S key and choosing it.

NOTE: If you don't save the source file when exiting from the Editor the format imported will not be added to the source code file.

Export a Screen/Report format

Use this option to save a screen/report format that is a part of the source code file currently being edited to a separated file on the disk. You might do this if you want to be able to include this same format in other programs.

a) Press the ^X key.

b) A list of the formats that are a part of the current program will be displayed.

c) Choose the format you wish to export by moving the cursor bar to the appropriate line and pressing the enter key. A file will be created using the format name and the appropriate extension in the current sub-directory.

Delete a Screen/Report format

a) Press the ^S key. A list of screen and report formats will be displayed.

b) Move the cursor to the appropriate line and press the ENTER key.

c) The Editor will ask you to confirm that you wish to delete the screen/report format. If you answer Y the format will be removed from the source code file being edited.

NOTE: If you don't save the source file when exiting from the Editor the format deleted will not be removed from the source code file.

Special WINDOW/WINDEF command option

If you are editing a **WINDOW** or **WINDEF** command you will be able to chain to a visual window editor. To do this press the F5 key at the first option in the command. When you return the appropriate values will be put into the command. If values already exist they will be used in the editor as the beginning window.

Compile program

You reach this option by placing the cursor on the **Compile program** option line and pressing the ENTER key or by pressing the **C** key.

Use this option to compile a TAS Professional 5.1 program directly. When the compilation is complete the compiler will tell you if there were any errors.

> **NOTE:** To compile the program with the Debug flag on press the ^D key. The DEBUG message will be displayed at the top of the window.

If you need to change any of the compiler specifications for this program you will either need to make the appropriate entry into the compiler information file (see below) or compile the program from the DOS prompt using the proper flags.

run Program

You reach this option by placing the cursor on the **run Program** option line and pressing the ENTER key or by pressing the **P** key. This option is part of the Main Menu and is not a separate program.

Enter the name of the TAS Professional 5.1 program you wish to run. If you have just compiled a program the name will be placed automatically in the entry field so all you need do is press the ENTER key to run the program.

Auto Run Utilities

These utilities are for handling files generated by TAS Professional 5.1's **AUTO_RUN** command. **AUTO_RUN** enables you to record and play back user keystrokes from within an application. For more information on the **AUTO_RUN** command please refer to **Chapter 5, Command Reference**.

Auto Run Utils -> convert .CHR to .ARN

You reach this option by placing the cursor on the **Auto Run Utilties** option line and pressing the ENTER key or by pressing the **A** key. Then press the ENTER key again to run the program.

Program name: CNVT_ARN.RUN

This program creates the binary .ARN files that are used by the AUTO_RUN command to fill the keyboard buffer with characters. When you choose this option a screen will be displayed asking for the Auto Run Name. Enter the 8 character file name of the .CHR file you wish to convert. A file with the same name will be created with the extension .ARN. Both files will be in your current sub-directory.

For an example of the files created and a sample program please see the program AUTOTEST.SRC and the files AUTOTEST.ARN and AUTOTEST.CHR in the SAMPLE sub-directory.

Auto Run Utils -> convert .ARN to .CHR

You reach this option by placing the cursor on the **Auto Run Utilties** option line and pressing the ENTER key or by pressing the **A** key. Then press the DN_ARROW key (or the **A** key) and the ENTER key again to run the program.

Program name: CNVT_CHR.RUN

This program creates the readable .CHR files from the .ARN files that are created by the AUTO_RUN command RECORD option. When you choose this option a screen will be displayed asking for the Auto Run Name. Enter the 8 character file name of the .ARN file you wish to convert. A file with the same name will be created with the extension .CHR. Both files will be in your current sub-directory.

This file created will have a listing of all characters entered during the RECORD option phase of the AUTO_RUN command. This will include all mistakes and corrections, literally every character. You can edit this file with any text editor to remove characters you don't want or to add ones you do. All control characters must be on separate lines and are prefaced with the tilde (~). For example, the following lines will enter a value into a field, backspace two characters, and finish entering:

```
Tom Joan
~H
~H
nes
```

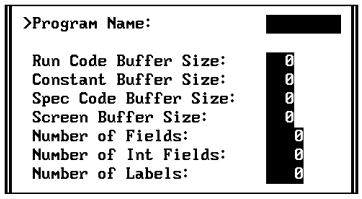
A list of control characters and their numeric equivalents are in the file CNVT_ARN.LST. For example, there is a line in this file like CR,13. This means that when you put ~CR in the .CHR file and convert it to a .ARN file the line ~CR will be converted to a binary 13. You can edit this CNVT_ARN.LST file with any text editor.

program comp Info

You reach this option by placing the cursor on the **program comp Info** option line and pressing the ENTER key or by pressing the **I** key.

Program name: COMPINFO.RUN

When TAS Professional 5.1 compiles a program certain buffers are created to temporarily hold data during the process. Most programs can be easily compiled with the default sizes that are set up. However, for those that can’t, use this program to enter compiler information about programs that cannot be compiled within the default buffer sizes. You will know when you have a problem since the error file will state that the compiler ran out of space in a certain buffer.



Enter the program name in the initial entry or you can use the standard file search keys to find a program already in the file. You can also press F2 to get a list of records.

NOTE: Each of the buffer sizes are in k bytes. Also, the actual buffer sizes that will be loaded during runtime are strictly dependent on actual sizes and are not directly affected by any of the entries you make here. These are just for the use of the compiler.

Program Name - The actual name of the program. For example, we have a record for TASEDSCR since it requires non-standard buffer sizes to compile.

Run Code Buffer Size - This is the buffer that keeps the actual line pointers. Each line requires 10 bytes. Generally this can be set to 10.

Constant Buffer Size - Each expression, numeric constant and alpha constant take up some room in this buffer. This is generally, however, the buffer that will grow the largest. How big to make it, however, is completely dependent on your program.

Spec Code Buffer Size - Each command creates an entry in this buffer. The size of each spec code is dependent on the command. Complex commands have a larger spec code. This is also a buffer that can grow rather large in complex programs. If you are compiling with the Debug flag on you might find that what compiled easily before doesn't now. This is where the extra line and program information needed is stored when programs are compiled with that option.

Screen Buffer Size - This is a buffer to hold the largest compiled screen or report format. Generally 10 is more than enough; however, if you have a very complex report format you may need to increase this.

Number of Fields - Each field entry requires 32 bytes. You may specify a maximum of 2000 fields.

Number of Int Fields - If you should run out of internal fields during program operation you can increase the number here. If you should find yourself in that situation, however, we recommend that you decrease the complexity of your program by breaking up expressions that are on a single line. In all of our programs we never have exceeded this limit.

Number of Labels - Each UDC and UDF in a program uses a label buffer along with all regular labels. Each label buffer is 16 bytes.

NOTE: One of the records in this file is called DEFAULT. This record can change the default compiler values and will affect all programs that don't have their own record. If you wish to change the default values change this record. In most cases with the memory management capabilities of TAS Professional 5.1 you will need only this DEFAULT record. If you should find that certain buffers are not large enough then change this record first before entering a record

specifically for that program. You should only have to enter a specific record if your memory resources are limited.

translate taBle

You reach this option by placing the cursor on the **translate taBle** option line and pressing the ENTER key or by pressing the **B** key.

Program Name: TASTRMGR.RUN

TAS Professional 5.1 has a unique option that allows you to set up your own command or option names quickly and easily. Through the use of this program all you need do is enter the new command or option name and then the actual one. Then you can use the new name just as you would the actual one.

TAS Compile All

You reach this option by placing the cursor on the **TAS Compile All** option line and pressing the ENTER key or by pressing the **T** key.

Program Name: TASCMPAL.RUN

This program gives you the capability of recompiling any or all programs where the source file (.SRC) is newer than the run file. When you choose this option a screen will be displayed that will allow you to enter the path and file name to check. You may use the wildcard characters ? and *. The program will search the path entered for the appropriate files. If a .RUN file is not found, or if it is older than the .SRC file the program will be compiled. A list of all programs compiled and the results of each is kept in a file called TASCMPAL.LST. This is a standard text file. Also, individual .ERR files are created for each program compiled.

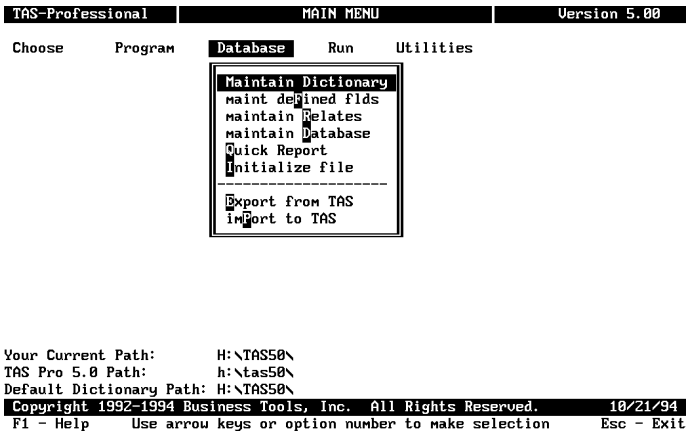
errOr list

You reach this option by placing the cursor on the **errOr list** option line and pressing the ENTER key or by pressing the **O** key.

Program Name: ERRLIST.RUN

This is a very simple program that checks all the .ERR files in the current sub-directory and creates a file with the names off those that have errors reported. This means you can compile a large group of files at one time, choose this option, and then looking at the file ERR.LST see the names of the programs that didn't compile correctly.

NOTE: Don't forget to erase old .ERR files before mass compiles so that you don't get false results for other programs that might not even have been in the compile group.



DATABASE

This is the menu for the programs that directly manipulate the different database files.

Maintain Dictionary

You reach this option by placing the cursor on the **Maintain Dictionary** option line and pressing the ENTER key or by pressing the **M** key.

Program name: TASDMGR.RUN

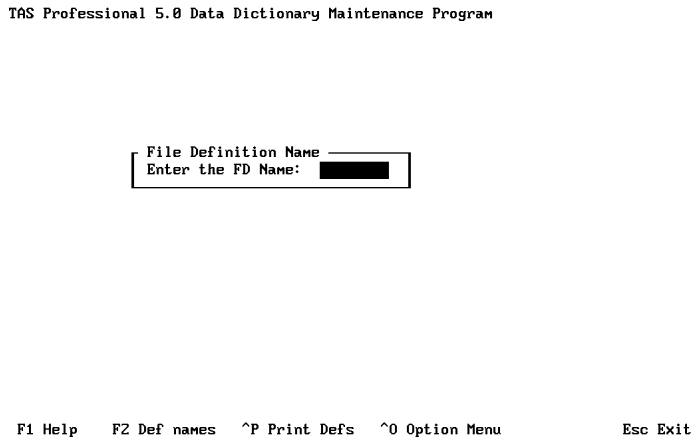
This program maintains the data dictionary files. All files may be defined in the data dictionary, whether they are TAS or non-TAS. Each file may have entries in all 4 files that make up the entire dictionary. These files are:

FILELOC.B - This file maintains the location, type, extension and size for each file in the dictionary. Multiple files that each use the same FD should have individual entries in this file. This entry will include the name and extension which make up a unique record. By keeping all files that use the same FD in this file, TAS Professional 5.1 is able to make sure that all files are changed when the FD changes. Also, it significantly eases the effort required to maintain separate groups of files (e.g., all the files for a company we maintain in our accounting programs).

FILEDICT.B - This is the actual dictionary definition of the file, or FD. Each field has a record in this file. This record determines the type, size, decimal chrs, upper case option and number of array elements, if any. You can also keep a 40 character description for each field.

FILEKEY.B - Each FD, whether it is for a Btrieve or non-Btrieve file, has a single record in this file. This keeps track of the indexes and how they are set up.

FILEKNUM.B - This file has a different listing of keys. Each key has a single record in this file which includes the FD name, the key name and the internal key number. A record must exist in this file for each key accessed in a program or you will receive errors during compilation or runtime. To make sure that this data is accurate you must either Initialize the file after changes have been made or Reindex or Restructure the file.



Maintain Dictionary

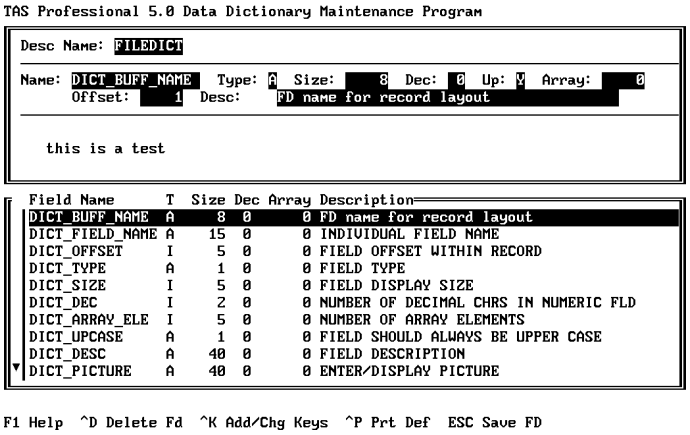
Operations

Creating a new FD

a) At the initial Maintain Dictionary Screen shown above enter the new FD name.

b) The program will alert you that the name doesn't exist in the Data Dictionary; answer **Y** to the create question.

c) The program will switch to the actual maintenance screen and you will be able to enter new lines as follows.



Adding/Inserting new lines in an FD

a) Enter the FD name in the initial screen. If this is a new FD answer **Y** to the create question.

b) The screen will switch to that shown above and the cursor will be in the lower half of the screen, the list window.

c) To add a new line move the cursor to the end of the list by pressing the END key and then the DOWN ARROW key to get to a blank line. If this is a new FD the cursor is probably already

on a blank line so you don't have to do anything else. To insert a line move the cursor to the field you wish to insert directly before.

d) Press the ENTER key (INSERT key in the case of an insert) and the cursor will move to the top part of the screen, the actual entry window.

e) Enter the name of the field.

f) Next enter the type. To get a list of types press the F1 key. You can choose one from the menu by pressing the highlighted character.

g) Next comes the size. Help information appropriate to the field type is displayed.

h) If the field type is **N** you will be able to enter the number of decimal characters.

i) If the field type is **A** you will be able to enter whether or not all entries should be in upper case.

j) You may enter the number of array elements for this field. Don't forget: an index field cannot have array elements.

k) If this is an overlay field, type **V**, you will be able to enter the offset value. The first character is position 1 within this program (this is converted to 0 when saved back to the dictionary file).

l) And finally you may enter a 40-character description for this field.

m) The program will ask if all is correct, and if you enter **Y** the line will be added to the end of the list. If this was an INSERT the line will be added at the appropriate location.

n) Don't forget: if you have changed an FD that has existing files with data you must Restructure the files so that they match the new entries in the dictionary.

Changing a line

a) Move the cursor to the appropriate line and press the ENTER key.

b) You may change any of the current entries including the field name. However, if this field is part of a key you must remember to also change the entries for that key.

c) Don't forget: if you have changed an FD that has existing files with data you must Restructure the files so that they match the new entries in the dictionary.

Deleting a line

a) Move the cursor to the appropriate line.

b) Press the DELETE key.

c) The program will ask if you're sure you want to delete the line. If you are enter **Y**.

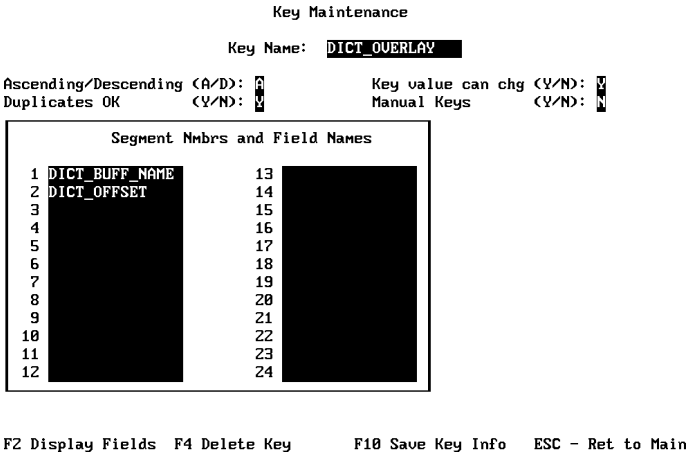
d) The line will be removed.

Deleting an entire FD

a) Once the FD name has been entered on the initial screen and the list of fields is displayed, press the ^D key while in the list window.

b) The program will ask if you're sure you want to delete the entire FD and all files that use it. If you're sure enter **Y**.

c) This process is irretrievable. All dictionary records for this FD will be deleted and all files that use it will be deleted from the disk.



Adding a new key to an FD

a) Press the ^K key while in the list window.

b) A new window will be displayed and the first option in it is to Add a new key. Press the ENTER key to choose that option. The screen above will be displayed.

c) Enter the key name. This can be the same as a field name or can be totally different. If there is only one field as part of the key we suggest you use the field name as the key name. This will allow you to access that field as a key in any program that will allow it. If the names are different the program will not know that the field is a key. Also, if you have multiple fields as part of the key we suggest that you use a different name so that there won't be any confusion about the data required to find a record using that key.

d) Now choose whether the key should be kept in Ascending or Descending order.

e) If you can have duplicate values for the same key enter **Y** here; for no duplicates enter **N**. If the user tries to save a record that would cause a duplicate value and you have entered **N** here the program will give an error (Btrieve #5) during the save process.

f) If the key value can change after it has been saved the first time enter **Y** here; if you don't want to allow changes enter **N**. If the user tries to save a record where the key has been changed and you have entered **N** here the program will give an error (Btrieve #10) during the save process.

g) If you enter **Y** at the Manual Keys question, it creates the following situation. When you save a record, if any segment in the key is null, i.e., all binary 0s, the key will not be added to the index. If you attempt to search for the record by that index you will not find it. However, if

you know you don't want to add records to that index when any segment in that key is blank then it will save time and disk space if you set this option to **Y**. This will not affect any other index for this record and does not affect how the master record information is saved in any case. The default entry is **N**, which causes the index to be updated each time a record is saved whether or not a key segment is null.

h) Next you need to enter the individual field names that make up this key. Each field must currently exist in the FD. You can get a list of fields by pressing the F2 key. You may have up to 24 different indexes each with one field per index (one segment) or a single index with 24 segments, or anything in between. The only maximum is you cannot exceed 24 different segments within one file.

i) To finish the entry either press the ENTER key at a blank segment name or the F10 key.

j) Answer the everything ok question. If you answer **Y** the information will be saved and the program will return to the main screen. If you answer **N** you will be asked if you wish to lose the changes made, and if you enter **Y** the program will return to the main screen and the changes will not be recorded.

Changing an existing key

a) Press the ^K key while in the field list window to get a list of keys.

b) Move the cursor to the key you wish to change and press the ENTER key.

c) The key information will be displayed.

d) To delete a segment from the key move the cursor to the appropriate segment name and press the F4 key. You will be asked if you're sure you want to delete the line, and if you enter **Y** it will be removed, and all segments after that will be moved up one.

e) To save the changes press the F10 key and return to the main screen.

Deleting an existing key

a) Press the ^K key while in the field list window to get a list of keys.

b) Move the cursor to the key you wish to delete and press the ENTER key.

c) While the cursor is still in the key name field press the F4 key. The program will ask if you're sure you want to delete the key. If you enter **Y** it will be marked for deletion and you will return to the main screen.

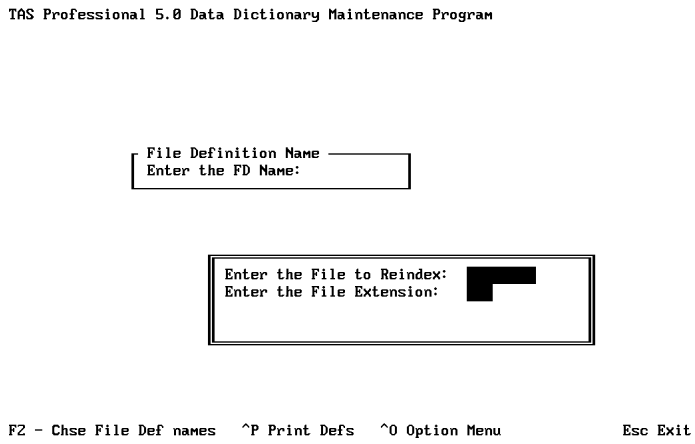
d) To unmark a key, choose it again and when the cursor is in the key name field, press the F4 key again. The program this time will ask if you wish to undelete the key. Enter **Y** and the deletion mark will be removed (when you list the keys there will be no star by the key name).

Update FD

a) When you are finished making entries to the FD press the ESC key.

b) If no changes were made the program will return automatically to the initial screen. If changes were made you will be asked if you wish to update the FD. If you enter **N** none of the changes you made will be saved. If you enter **Y** the old FD records will be deleted and the new ones will be added.

c) If you have made changes to an existing FD and there are files that contain data that use that FD, you must Restructure those files so that everything is up to date. We recommend that you do so immediately upon making changes to the FD. You will also need to recompile any programs that use fields in that FD so that everything matches up.



Reindexing a file

a) From the initial screen press the ^O key to display the options menu.

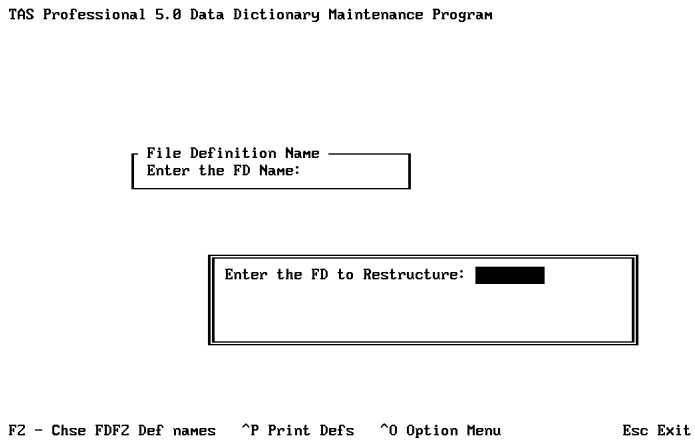
b) Enter **R** and press the ENTER key to choose the Reindex option.

c) The program will ask for the name of the file to be reindexed. You may press the F2 key to get a list of files.

d) You will need to enter both the file name and extension to specify an exact file.

e) Once you have entered both, and the program verifies that the file exists, it will rename the file on the disk, create a new file (you will see the Initialize message) and then give a running tally on the number of records remaining.

f) When the process is complete the program will return to the initial entry screen.



Restructuring a file

a) From the initial screen press the ^O key to display the options menu.

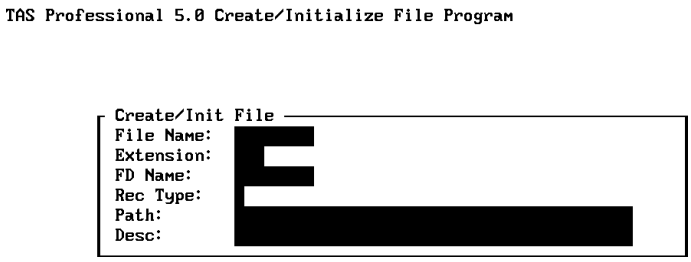
b) Enter **R** twice (the first time you will get Reindex) and press the ENTER key to choose the Restructure option. You would use this process when you have made changes to an FD and there are files with data existing that are based on that FD. This will force the data in the files to match the information in the dictionary. **MAKE SURE YOU HAVE A GOOD BACKUP FOR ALL FILES THAT USE THIS FD. THIS PROCESS CAN IRRETRIEVABLY DESTROY THE FILES IF ANYTHING SHOULD GO WRONG.**

NOTE: This process cannot change a field from one type to another. It can be used when making a field larger or smaller, when adding new fields or deleting current ones, or adding to or subtracting from the number of array elements.

c) The program will ask for the name of the FD to be restructured. You may press the F2 key to get a list of FDs.

e) Once you have entered the FD, and the program verifies that it exists, it will proceed. The process is basically to write out the records to a fixed length file, initialize the file with the new specifications, and then save the records back with the changes made to each record as appropriate. The program will tell you how many files remain and how many records remain for the current file.

f) When the process is complete the program will return to the initial entry screen.



Initialize a file

a) From the initial screen press the ^O key to display the options menu.

b) Press the **C** key to choose the Create/Init File option and press the ENTER key. The screen above will be displayed.

NOTE: This is the same process that would occur if you chose the Initialize File option from the Main Menu.

c) Enter the name of the file to be initialized. If the file already exists you can get a list by pressing the F2 key. If it is new then enter the name now.

d) Next enter the extension. The default extension for TAS Professional 5.1 files is B.

e) If the file exists the balance of the information should automatically appear once you have entered the extension. If this is a new file then you need to enter the FD name. Don't forget: the filename need not match the FD name.

f) Next enter the Record type: **T** - TAS Professional 5.1 file; **B** - non-TAS Btrieve file; **X** - Text type ASCII file; **F** - Fixed length ASCII file or **D** - dBASEIII+ file.

NOTE: In the case of any file type other than T or B this process will just create the file on the disk. It will not, in the case of dBASEIII+ files, create the format. However, all files must be initialized so that internal values can be set properly in the data dictionary.

g) If the file will be in some other path than the default (current path) then enter that information next in standard path notation. The program will make sure that you have a backslash '\' at the end of the path name.

h) You may also enter a 40 character description about this file.

i) After you enter the description, the program will process with the initialization process. If the file already exists you will be notified and will have the opportunity to stop the process.

j) Once it is complete the program will return to the initial screen.

Print all or a group of FDs

a) From the initial screen press the ^P key.

b) A window will be displayed so that you may enter the beginning and ending FD name. You may press the F2 key to get a listing of FDs to choose from.

c) Each FD will be printed starting on a new page.

d) To get a listing of all FDs just press the ENTER key at both the beginning and ending entry fields.

e) When finished the program will return to the initial screen.

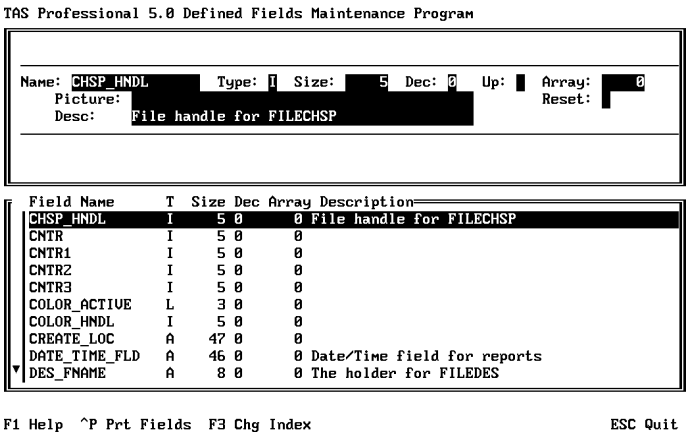
maint deFined flds

Use this program to maintain a group of fields that will be used in your programs but are not a part of any file. This is the equivalent of defining the field in your program. During compilation the program will look for a defined field within your program, and if it doesn't find it there it will look for it in the regular file data dictionary, and finally, will look to this file.

You reach this option by placing the cursor on the **maint deFined flds** option line and pressing the ENTER key or by pressing the **F** key.

Program name: TASDFLDM.RUN

This program works similarly to the Maintain Dictionary program except that there are no specific FDs. There is just a list of all fields that are a part of the Defined Field Dictionary database.



maint deFined flds

Operations

a) To change a field move the cursor (using the UP and DOWN ARROW keys, PAGE UP or PAGE DOWN, HOME or END, or by entering the name of the field) to the appropriate field name. Then press the ENTER key and the cursor will move to the top part of the screen. You will see that as you move the cursor in the list the field information also changes in the top part of the window.

b) To insert a new field press the END key then the DOWN ARROW key to display a blank line. Then press the ENTER key and you will be able to enter a new field. You can also press the INSERT key to enter a new field.

c) To Delete a field move the cursor to the appropriate field and then press the DELETE (DEL) key. If you confirm the deletion the field will be removed from the file.

NOTE: When you change or delete a field in this program it is actually changed or deleted, and you won't have the opportunity to reverse your action.

Normally the list is in field name order. However, you can change the order by pressing the F3 key. This will redisplay the list in type/size order. All other processes will remain the same.

maintain Relates

Use this program to maintain the relationships between files. This can be used in the Edit Screen and Edit Report programs when entering the file information.

You reach this option by placing the cursor on the **maintain Relates** option line and pressing the ENTER key or by pressing the **R** key.

Program name: TASRLATE.RUN

This program works similarly to the Maintain Defined Fld Dictionary program.

The process TAS Professional 5.1 uses when searching for related files is as follows:

a) After a record is found, the program checks to see if any relationships have been set up. These must be part of the program; just putting records in this file will not automatically set up the relationships in a program.

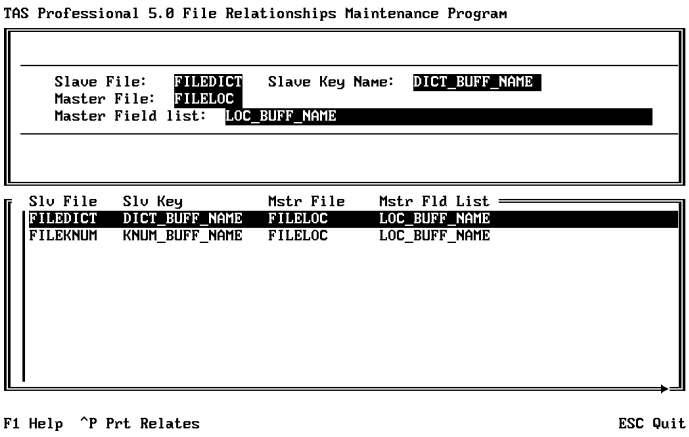
b) If there are relationships, the program will check to see if the file just accessed is listed as a Master file in any of the relationships.

c) If it is then the data in the Master Field List is moved into the key buffer for the Slave file.

d) The program then attempts to search, with a Match Exact setting, for the record. If one is not found the buffer in the Slave record is cleared.

e) The program does not check to make sure that the Master Field List matches the segment or segments in the Slave key. It is your responsibility to make sure they match up.

f) This checking process continues until all related records are found. You can set sub-relationships up so that a Slave is actually a Master to another file.



maintain Relates

Operations

a) To change a relation record move the cursor (using the UP and DOWN ARROW keys, PAGE UP or PAGE DOWN, HOME or END) to the appropriate line. Then press the ENTER key and the cursor will move to the top part of the screen. You will see that as you move the cursor in the list the relationship information also changes in the top part of the window.

b) To insert a new relation press the END key then the DOWN ARROW key to display a blank line. Then press the ENTER key and you will be able to enter a new record. You can also press the INSERT key to enter a new record.

c) To Delete a relationship move the cursor to the appropriate line and then press the DELETE (DEL) key. If you confirm the deletion it will be removed from the file.

d) When the cursor is in the top window you will be able to set up the relationship. The first entry is the Slave File name. Press the F2 key to get a list of available files.

e) Next is the Slave Key. Here you need to enter the key or index name that will be used to find records in this file when you find a record in the Master file. Press the F2 key to get a list of indexes for the file.

f) Next enter the Master file name. To get a listing of all files press the F2 key. Enter an accurate file name.

g) Enter the fields in the Master that correspond to the key information in the Slave. This means that when the Master record is found the program will use the data in these fields when searching for the Slave. If there are multiple fields then separate them with commas. You may list fields in the Master by pressing the F2 key.

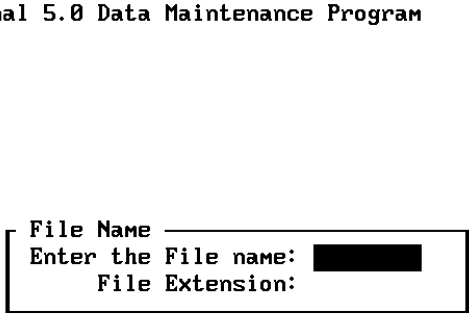
NOTE: When you change or delete a record in this program it is actually changed or deleted and you won't have the opportunity to reverse your action.

maintain Database

Use this program to get direct access to the data in your files. You can call up data for observation or modification.

You reach this option by placing the cursor on the **maintain Database** option line and pressing the ENTER key or by pressing the **D** key.

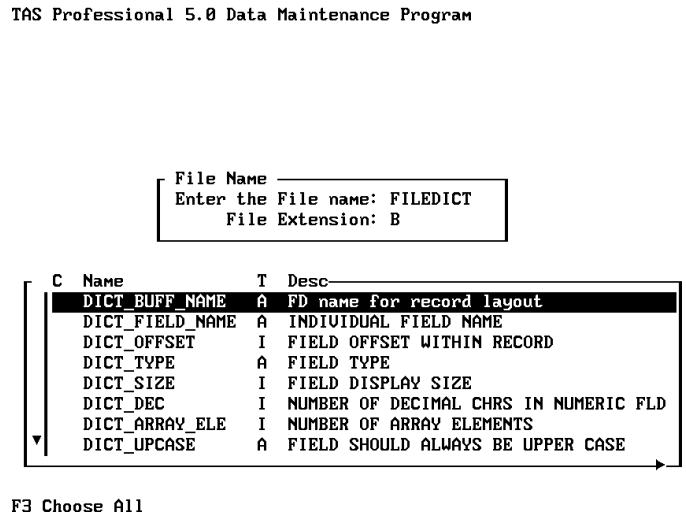
Program name: TASDATAM.RUN & TASDATA2.RUN



maintain Database

Operations

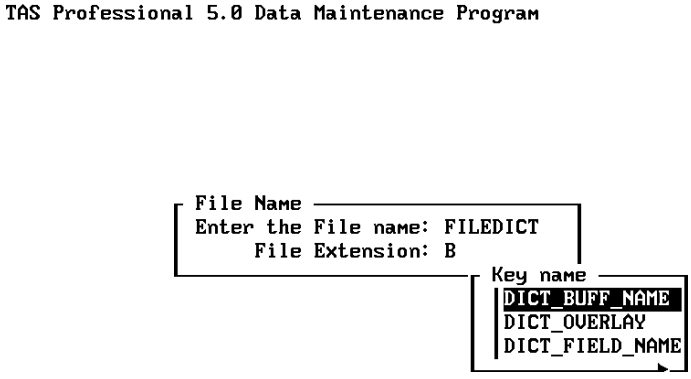
a) Enter the File name and extension. You may get a list of available files by pressing the F2 key.



b) After you choose the file the fields within that file are displayed. You may choose all the fields in a file by pressing the F3 key or moving the cursor to the proper line in the choose list and pressing the ENTER key. When a field is chosen a star '*' appears to the left of the name. To unchoose a field press the ENTER key again and it will disappear. The cursor automatically proceeds to the next line when choosing manually.

A maximum of 260 fields may be chosen. However, an array field only uses one line.

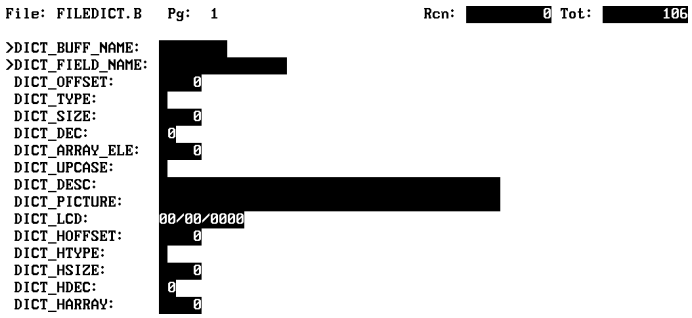
c) Once you have chosen all the appropriate fields press the ESC key. If there is an Owner defined for this file (the file is protected) you will have to enter the password before you will be allowed to access it.



d) If there are any keys set up for this file the names will be displayed. You can either choose a key or press the ESC key and you will be able to access any key. The difference in operation is if you choose a single key you will be able to scan forward (F8) or backward (F7), find first (F5), or last (F6) without having the cursor in the appropriate key field. All operations will be accomplished using that key (index) only. However, if you wish to search by any of the indexes then you would press the ESC key. When the fields are displayed, any of the fields

that are also key names will be displayed with a '>' to the left of the name. You can then use the standard record search keys (F5-F9) when the cursor is in that entry field. This allows you to access all keys.

If you wish to use a key that is not also a field name, such as in a multi-segment key or an overlay field you must choose the key name when it is displayed. Otherwise, you will not have the chance to use it.



e) The names of the fields will be displayed down the left hand side of the screen and the actual entry fields are just to the right.

f) If you have chosen an array field you will notice the standard array element notation as part of the field name. If you move the cursor to that field you can 'page' through the elements in that field by pressing the PAGE UP (previous element), PAGE DOWN (next element), HOME (first element) or END (last element) keys. The current element number is displayed as part of the field name.

g) If there are more fields than will fit one page you can reach subsequent pages by pressing the ^PAGE DOWN key or to go to previous pages the ^PAGE UP key.

h) To save a record press the F10 key, to delete a record press the F4 key, and to clear the record buffer press the F3 key.

i) When you are finished, press the ESC key and you will return to the initial entry screen. Press ESC again and you will return to the Main Menu.

fd differeNces

This program allows you to compare the current data dictionary to one in another sub-directory.

You reach this option by placing the cursor on the **fd differeNces** option line and pressing the ENTER key or by pressing the **N** key.

Program name: FD_DIFF.RUN

When you choose this option the program will ask you to enter the path of the dictionary to be compare. You will also be able to specify which FDs to check. You may use the wildcards ? and * here. If you wish to compare all then just leave this field blank.

The program will run through the FDs in the current dictionary that match your entry and then will check for those same names in the remote dictionary. Differences in field sizes, types, decimal characters, even whether they exist or not will be recorded in the file FD_DIFF.LST. The program will also report whether the keys are different. The FD_DIFF.LST file is a standard text type file.

This program makes no changes to either dictionary.

Quick Report

This program allows you to select certain files and fields for printing to a printer, screen, or disk file. You can use a filter to restrict your report to certain records.

You reach this option by placing the cursor on the **Quick Report** option line and pressing the ENTER key or by pressing the **Q** key.

Program name: TASRPNOW.RUN & TASRPNW2.RUN

In several TAS Professional 5.1 programs you can use **filters** to restrict the records you access, print, change, and so on. A filter is an expression involving data fields that constrains future operations (such as a print) to records that meet the expression's criteria (i.e., make the expression evaluate to .T. or "true"). For example, the expression **BKAR.INV.CUSNME = "Larry Jones"** would 'filter' the records in the BKARINV file, returning only those where the CUSNME field was set to Larry Jones.

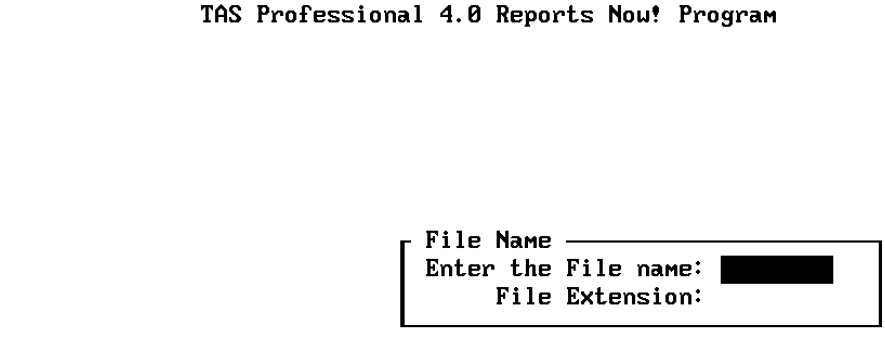
An expression can contain field-names and constants or literals separated by Boolean operators:

>	greater than
<	less than
=	equal to
<>	not equal to
>=	greater than or equal to
<=	less than or equal to
$	is included in ("in-string" operator)

Expressions can be joined or chained using connectives:

.a.	AND
.o.	OR
.n.	NOT

For example, the expression **(BKAR.INV.NUM < 5000) .a. (BKAR.INV.CUSST = "CA")** would return those invoice records for invoices numbered below 5000 where the customer state was California. You may use any standard (intrinsic) TAS Professional 5.1 function or operator. Also: in the filter expression you may only use fields which have been selected or chosen (marked with an asterisk).



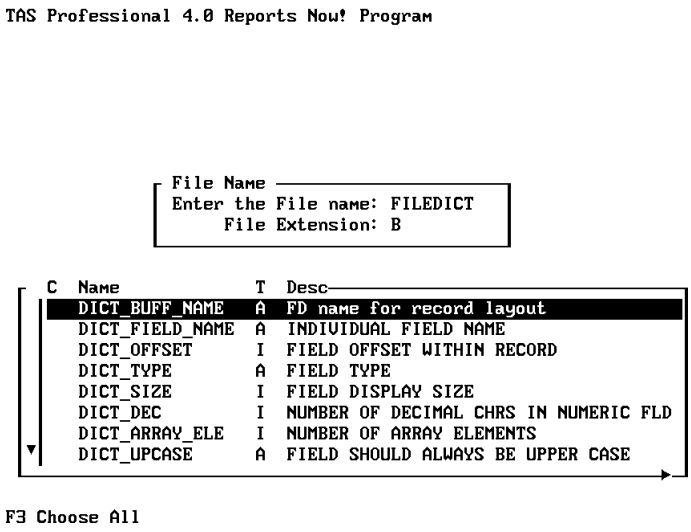
Quick Report

Operations

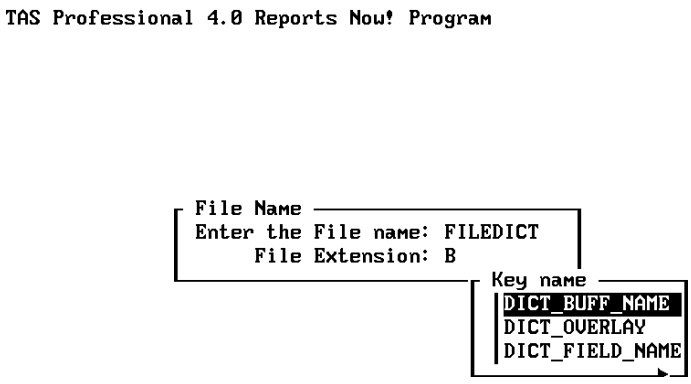
a) Enter the File name and extension. You may get a list of available files by pressing the F2 key.

b) After you choose the file, the fields within that file are displayed. You may choose all the fields in a file by pressing the F3 key or moving the cursor to the proper line in the choose list and pressing the ENTER key. When a field is chosen a star '*' appears to the left of the name. To unchoose a field press the ENTER key again and it will disappear. The cursor automatically proceeds to the next line when choosing manually.

A maximum of 200 fields may be chosen with a maximum print width of 800 characters. If you choose an array field you will also be asked to choose which elements. Each element is treated as a separate field. If you choose all fields by pressing the F3 key only the first element of any array will be displayed.



c) Once you have chosen all the appropriate fields press the ESC key. If there is an Owner defined for this file (the file is protected) you will have to enter the password before you will be allowed to access it.



d) If there are any keys set up for this file the names will be displayed. You can either choose a key or press the ESC key to print in record number order.

e) Next you may enter a filter expression. You may use any standard (intrinsic) TAS Pro 5.1 function or operator and any field that has been chosen to be printed in the expression.

f) The program then asks where you want to print. If you choose to print to the printer and the report is over 80 characters it will ask if you want to print compressed. If you answer **Y** (and there is an entry in the current printer driver for PCMP) the program will attempt to set the printer. If the report is wider than 128 characters or you don't want compressed it will word wrap the output.

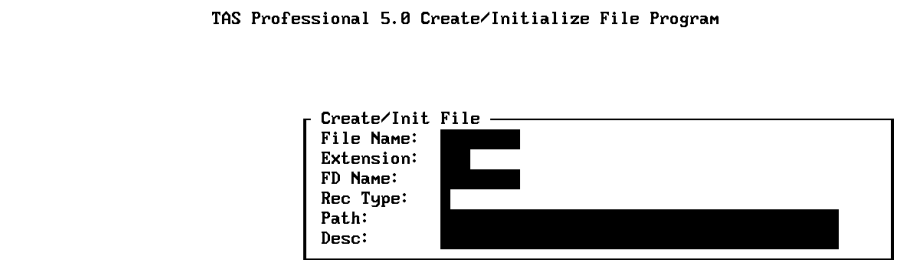
g) If you print to the screen and the report is wider than 80 characters the program will automatically use the screen output scrolling routines so that you can move the report left and right by pressing the left or right arrow keys (or Ctrl_left and Ctrl_right to move 40 characters at a time).

h) You may interrupt the report by pressing ESC at any time. After the report is finished it will ask you if you wish to print again. If you answer **Y** you will be able to use the same choices while changing the filter expression so that you get different results. If you enter **N** to the print again message the program will return to the initial screen. Press the ESC key to return to the Main Menu.

Initialize File

You reach this option by placing the cursor on the **Initialize file** option line and pressing the ENTER key or by pressing the **I** key.

Program name: TASINIT.RUN



Initialize File

Operations

a) Enter the name of the file to be initialized. If the file already exists you can get a list by pressing the F2 key. If it is new then enter the name now.

b) Next enter the extension. The default extension for TAS Professional 5.1 files is B.

c) If the file exists the balance of the information should automatically appear once you have entered the extension. If this is a new file then you need to enter the FD name. Don't forget: the filename need not match the FD name. You may press the F2 key to get a list of available FDs.

f) Next enter the Record type: **T** - TAS Professional 5.1 file; **B** - non-TAS Btrieve file; **X** - Text type ASCII file; **F** - Fixed length ASCII file or **D** - dBASEIII+ file.

NOTE: In the case of any file type other than T or B this process will just create the file on the disk. It will not, in the case of dBASEIII+ files, create the format. However, all files must be initialized so that internal values can be set properly in the data dictionary.

g) If the file will be in some other path than the default (current path) then enter that information next in standard path notation. The program will make sure that you have a backslash '\' at the end of the path name.

h) You may also enter a 40 character description about this file.

i) After you enter the description, the program will proceed with the initialization process. If the file already exists you will be notified and will have the opportunity to stop the process.

j) Once it is complete the program will return to the Main Menu.

Export from TAS

This program will allow you to choose a file and fields to be exported to 6 different non-TAS file types.

You reach this option by placing the cursor on the **Export from TAS** option line and pressing the ENTER key or by pressing the **E** key.

Program name: TASXPORT.RUN & TASXPRT2.RUN

Export from TAS

Operations

a) Enter the File name and extension. You may get a list of available files by pressing the F2 key.

b) After you choose the file, the fields within that file are displayed. You may choose all the fields in a file by pressing the F3 key or moving the cursor to the proper line in the choose list and pressing the ENTER key. When a field is chosen a star '*' appears to the left of the name. To unchoose a field press the ENTER key again and it will disappear. The cursor automatically proceeds to the next line when choosing manually.

A maximum of 200 fields may be chosen. If you choose an array field you will also be asked to choose which elements. Each element is treated as a separate field. If you choose all fields by pressing the F3 key only the first element of any array will be displayed.

c) Once you have chosen all the appropriate fields press the ESC key. If there is an Owner defined for this file (the file is protected) you will have to enter the password before you will be allowed to access it.

d) Next you may enter a filter expression. You may use any standard (intrinsic) TAS Pro 5.1 function or operator and any field that has been chosen to be printed in the expression.

e) If there are any keys set up for this file the names will be displayed. You can either choose a key or press the ESC key to print in record number order.

f) You are then asked which type of export file. You may choose from the following list:

dBase III	**D**
Delimited	**L**
Fixed/SDF	**F**
SYLK	**Y**
DIF	**I**
Text	**X**

g) And finally the export file name. You can get a list of files on the disk by pressing the F2 key. To limit the listing (a maximum of 250 files may be displayed) enter a skeleton file name before you press the F2 key. For example, to see all the files with the extension LST you would enter *.LST in the export file name entry field and then press the F2 key. Don't forget: you must enter a legal file name before you can start the export process.

h) If you choose a file that already exists the program will alert you to that. Your first option is to overwrite the file, i.e., create a new file. If you answer **N**, you will be given the option to append the new records to the current file (it is your responsibility to make sure that the records are of the same type and make up before you choose this option). If you answer **N** here the program will return to the export file name entry field.

i) After you choose a file the records will be exported. A field displaying the current record location in the TAS Professional 5.1 file will be displayed.

j) When the process is complete the program will return to the initial screen. To return to the Main Menu press the ESC key.

iMport to TAS

This option will allow you to choose a file and fields to be imported from 6 different non-TAS file types.

You reach this option by placing the cursor on the **imPort to TAS** option line and pressing the ENTER key or by pressing the **P** key.

Program name: TASMPORT.RUN & TASMPRT2.RUN

iMport to TAS

Operations

a) Enter the File name and extension of the file you wish to import to. You can get a list of available files and descriptions by pressing the F2 key. The file names are drawn from the subdirectory where your data dictionary is located.

b) After you choose the file the fields within that file are displayed. You may choose all the fields in a file by pressing the F3 key, or you can select individual fields by moving the cursor to the proper line in the choose list and pressing the ENTER key. When a field is chosen, the program asks you to assign a number indicating the order in which it appears in the import file; these assigned numbers may or may not mimic the order of the fields in the data dictionary. However, for all file types except delimited, the numbers assigned must run consecutively (no gaps). If you press the F3 key to select all, the fields are numbered in the order in which they appear in the dictionary file. If you are importing from a delimited file (only) you

may skip fields in the import file by leaving gaps in the field numbers you assign. The first field in the record is 1. On all other file types, the numbers must run consecutively or the import will not work properly.

EXAMPLES:

F3 Chosen	**Select #1**	**Select #2**	**Delimited Only**
1 * FIELD1	3 * FIELD1	FIELD1	3 * FIELD1
2 * FIELD2	1 * FIELD2	1* FIELD2	6 * FIELD2
3 * FIELD3	2 * FIELD3	FIELD3	1 * FIELD3

A maximum of 200 fields may be chosen. If you choose an array field you will also be asked to choose which elements. Each element is treated as a separate field. If you choose all fields by pressing the F3 key only the first element of any array will be displayed.

c) Once you have chosen all the appropriate fields press the ESC key. If there is an Owner defined for this file (the file is protected) you will have to enter the password before you will be allowed to access it.

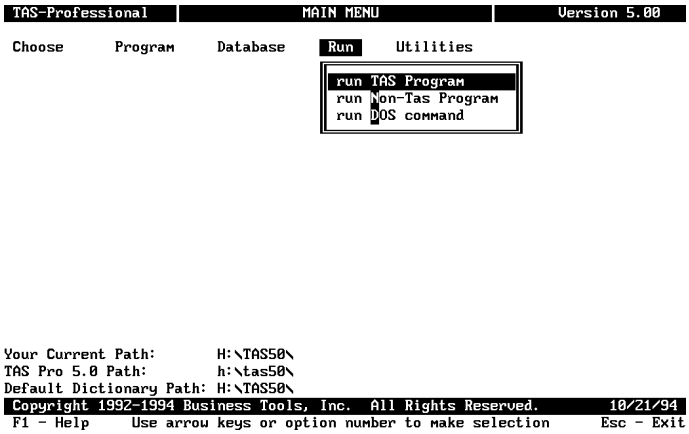
d) Next you may enter a filter expression. You may use any standard (intrinsic) TAS Professional 5.1 function or operator and any field that has been chosen to be printed in the expression.

e) You are then asked which type of import file. See the list of types in the previous section.

f) And finally the name of the file you wish to import. You can get a list of files on the disk by pressing the F2 key. To limit the listing (a maximum of 250 files may be displayed) enter a skeleton file name before you press the F2 key. For example, to see all the files with the extension LST you would enter *.LST in the export file name entry field and then press the F2 key. Don't forget: you must enter a file name that actually exists before you can start the import process.

g) After you choose a file, the records will be imported. A field displaying the current record location in the TAS Professional 5.1 file will be displayed.

h) When the process is complete the program will return to the initial screen. To return to the Main Menu, press the ESC key.



RUN

These are the options that will allow you to choose to run a TAS program, a non-TAS program or a DOS command.

run TAS Program

You reach this option by placing the cursor on the **run TAS Program** option line and pressing the ENTER key or by pressing the **T** key. This option is part of the Main Menu and is not a separate program.

Enter the name of the TAS Professional 5.1 program you wish to run. If you have just compiled a program the name will be placed automatically in the entry field so all you need do is press the ENTER key to run the program.

You can get a list of run programs on the disk by pressing the F2 key. To limit the listing (a maximum of 250 files may be displayed) enter a skeleton file name before you press the F2 key. For example, to see all the run programs that begin with TAS you would enter TAS* in the run file name entry field and then press the F2 key. The program automatically limits the search to any file with the extension .RUN.

To return to the Main Menu press the ESC key.

run Non-TAS Program

You reach this option by placing the cursor on the **run Non-TAS Program** option line and pressing the ENTER key or by pressing the **N** key. This option is part of the Main Menu and is not a separate program.

Enter the name of the non-TAS program you wish to run. This would generally be a .EXE or .COM program, such as a Text editor.

You can get a list of programs on the disk by pressing the F2 key. To limit the listing (a maximum of 250 files may be displayed) enter a skeleton file name before you press the F2 key. For example, to see all the programs that begin with the letters TAS you would enter TAS* in the file name entry field and then press the F2 key. The program automatically limits the search to those files with the extension .EXE or .COM.

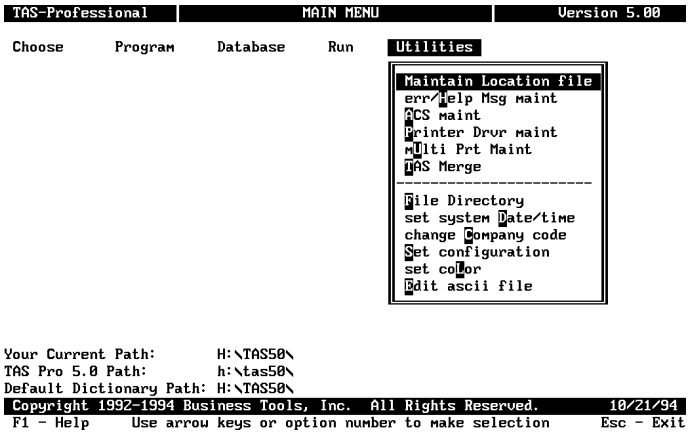
To return to the Main Menu press the ESC key.

run DOS Command

You reach this option by placing the cursor on the **run DOS Command** option line and pressing the ENTER key or by pressing the **D** key. This option is part of the Main Menu and is not a separate program.

Enter the name of the DOS command you wish to run. You can get a list of DOS commands by pressing the F2 key.

To return to the Main Menu press the ESC key.



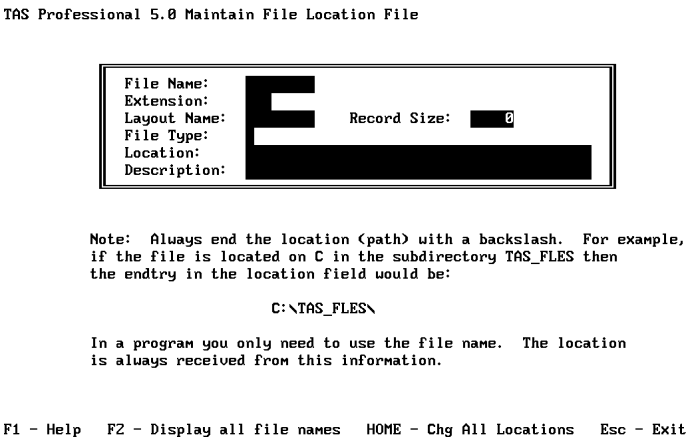
UTILITIES

These are the other support programs for TAS Pro 5.1 that maintain certain data files important to the running of your programs.

Maintain Location File

You reach this option by placing the cursor on the **Maintain Location file** option line and pressing the ENTER key or by pressing the **M** key. Then press the ENTER key again to choose this program.

Program name: TASFLOC.RUN



Maintain Location File

Operations

a) Enter the name of the file. If the file already exists you can get a list by pressing the F2 key. If it is new then enter the name now. If you press the HOME key, you will be able to change the location for all files. When you choose this option, you will also be given the opportunity to specify a Company Code (for use with Advanced Accounting). This will allow you to easily change the path for just a single company without having to change each file separately.

NOTE: If you are running Advanced Accounting multiuser and multi-company, you must have each company's data files in a separate subdirectory. This applies to the data only, not the programs. Failure to keep the data for each company separate could result in a Btrieve error, and you might lose some of your data.

b) Next enter the extension. The default extension for TAS Professional 5.1 files is B.

c) If the file exists the balance of the information should automatically appear once you have entered the extension. If this is a new file then you need to enter the FD name. Don't forget: the filename need not match the FD name. You may press the F2 key to get a list of available FDs.

f) Next enter the Record type: **T** - TAS Professional 5.1 file; **B** - non-TAS Btrieve file; **X** - Text type ASCII file; **F** - Fixed length ASCII file or **D** - dBASEIII+ file.

NOTE: In the case of any file type other than T or B this process will just create the file on the disk. It will not, in the case of dBASEIII+ files, create the format. However, all files must be initialized so that internal values can be set properly in the data dictionary.

g) If the file will be in some other path than the default (current path) then enter that information next in standard path notation. The program will make sure that you have a backslash '\' at the end of the path name. If you change the path for an existing file you will have to physically copy the file to the new location. This program will not do that.

h) You may also enter a 40 character description about this file.

i) This program should be used primarily for changing location path values. You can also delete a file using this program. After loading the file information by entering the name and extension press the F4 key. The program will verify that you want to delete the file. Be sure you want to do this since the program also deletes the file from the disk.

j) If you enter a new file using this program it will not be created automatically. If you wish to do that, use the program **Initialize Data File** in the DATABASE menu.

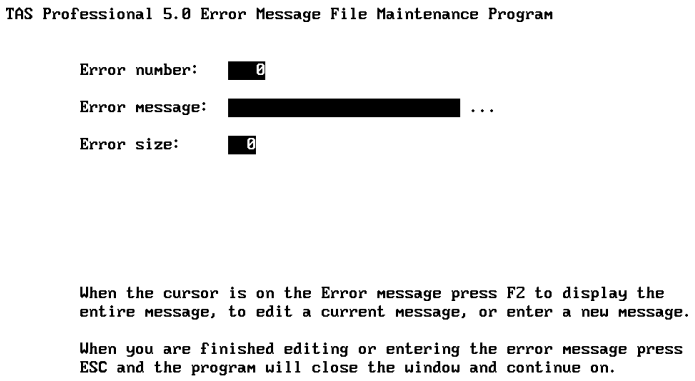
k) To return to the Main Menu press the ESC key.

err/Help Msg maint

You reach this option by placing the cursor on the **err/Help Msg maint** option line and pressing the ENTER key or by pressing the **H** key.

Program name: TASERMSG.RUN

This program will allow you to maintain the ERRMSG.B file. This contains all the error messages for TAS Pro 5.1 (both the runtime and the compiler). You may also use this file yourself for help or error messages. Please start with a number larger than 1000.



err/Help Msg maint

Operations

a) Enter the Error number. If it already exists a portion of the error message will be displayed.

b) To edit the error message move the cursor to the error message entry field and press the F2 key. A TASEdit edit window will be opened with the error message. You can now make any changes you wish using the standard TASEdit options (for more information see the beginning of this chapter). Press the ESC or RETURN key when you are finished making changes. (You may not put a cr/lf into the message; it must be continuous).

c) The program will display the net size of the message and will ask if you want to save the record. Enter **Y** if you do.

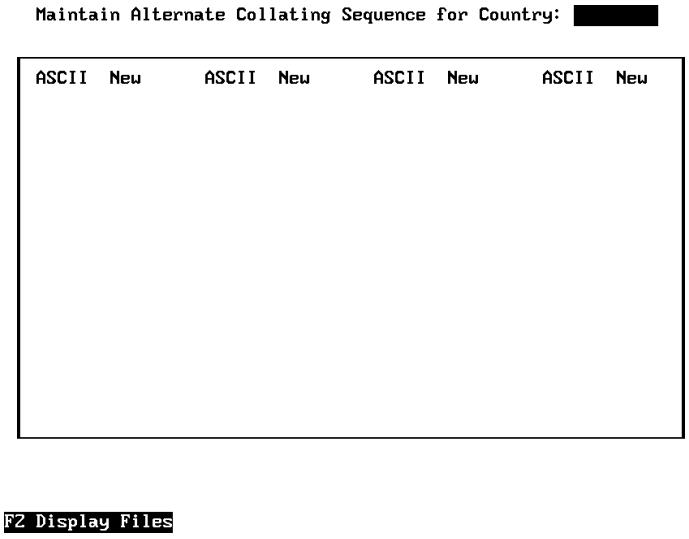
d) To return to the Main Menu press the ESC key while the cursor is in the error number field.

ACS Maint

Use this program to maintain or add to the Alternate Collating Sequence files for TAS Pro 5.1. These are used to redefine sort sequences for non-American alphabets.

You reach this option by placing the cursor on the **ACS maint** option line and pressing the ENTER key or by pressing the **A** key.

Program name: TASACS.RUN



You can press the F2 key to get a list of available ACS files. To add a new one enter the name of the country and press the ENTER key.

ACS Maint

Operations

a) Eight columns of numbers and characters will be displayed. These are actually 4 sets of 2 columns each. The first is the standard ASCII collating sequence. The second is what the sequence should be for your country. To make changes you need to enter the ASCII code for each character in the sequence it should be sorted ascending (lowest character to highest). There are 4 pages of columns (0-63, 64-127, 128-191, 192-255). Use the PAGE DOWN and PAGE UP key to go from page to page.

b) When you are finished press the ESC key and you will be asked if you want to save the information.

NOTE: You must make these changes before initializing any file that is to use this sequence. If you add sequence information, or make changes to it after the file is created, you will need to Reindex or Restructure the file to make sure this information is included in the file structure.

NOTE: No ACS file will be effective until you have entered the appropriate name in TAS50.OVL using the **Set Configuration** option in this **Utilities** submenu.

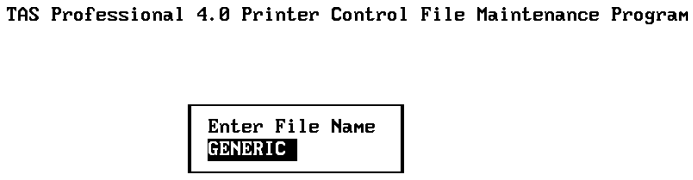
For more information about this feature please contact CAS support.

Printer Drvr maint

Use this program to maintain or add to the printer driver files for TAS Pro 5.1. These are used to determine the appropriate printer control code sequences when changing print sizes and may be used for other printer features.

You reach this option by placing the cursor on the **Printer Drvr maint** option line and pressing the ENTER key or by pressing the **P** key.

Program name: TASPRTR.RUN

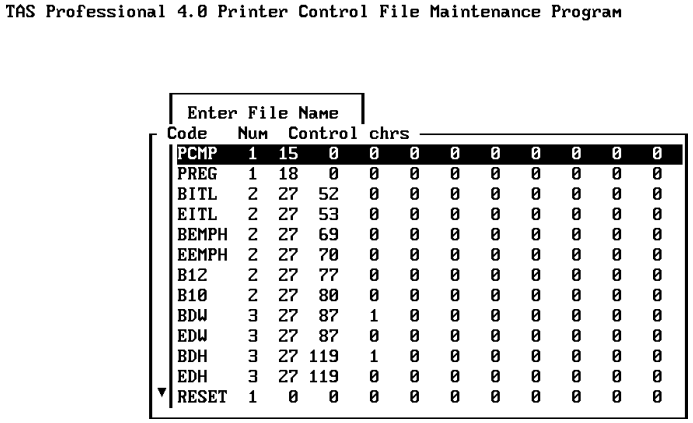


You can press the F2 key to get a list of available printer driver files. To add a new one enter the printer name or a pseudonym and press the ENTER key.

Printer Drvr maint

Operations

a) Enter the name of the printer or an 8 character pseudonym. You can press the F2 key to get a list of available printer driver files.

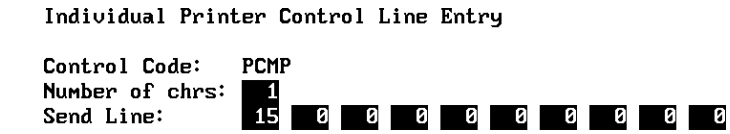


b) Each printer control code has its own 5 character alpha name and 10 characters to be sent to the printer when this option is used in the **PCHR** (printer control character) command. You also need to tell the option how many characters are in the control line.

c) The list provided is a standard entry list. You move up and down the list with Up and Down Arrow keys. You may also go to a specific line by pressing the first character of the name. To change a value move the cursor to it and press the ENTER key. To enter a new item move to the bottom of the list (END key) and press the Down Arrow key to display a blank line, then

press the ENTER key. You may also insert a line in the list by moving the cursor to the proper location and pressing the Ins key. Please note that the location of a control line within the file makes no difference to TAS Pro 5.1.

To delete a line item move the cursor bar to the appropriate line and press the DELETE key. The program will verify that you want to delete the control code line.



d) After you choose the line and press the ENTER key (or add/insert a new line) the screen above will be displayed.

e) If appropriate enter the control code name. This is what you will use in your programs when referring to this line. In general, the only codes used in any TAS program are PCMP (change to compressed format) and PREG (change to regular - 10 pitch - format).

f) Enter the number of characters in the control code.

g) Then enter the actual control characters themselves. These should all be in decimal form.

h) After you have entered the appropriate number of characters the program will automatically ask if everything was entered correctly. If you enter **Y** the line will be added/updated in the list.

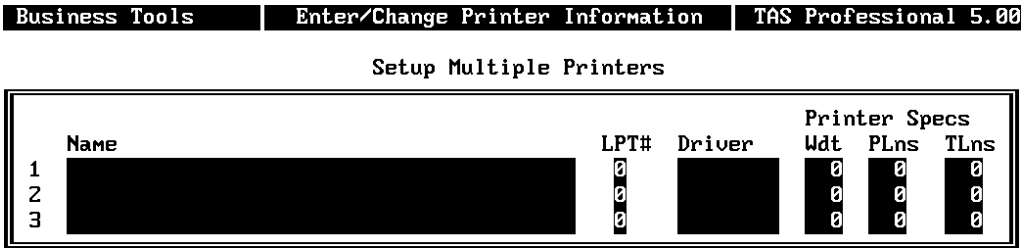
i) To exit from the list press the ESC key. The program will ask if you wish to save the driver back to disk. Enter **Y** to update the current information.

mUlti Prt Maint

A major feature in TAS Professional 5.1 is the ability to direct printer output to multiple printers, each with their own printer driver. You set up the particular information for each printer here.

You reach this option by placing the cursor on the **mUlti Prt Maint** option line and pressing the ENTER key or by pressing the **U** key.

Program name: TASMPRT.RUN



To setup multiple printers:

1. Enter the name you wish to call the printer. This can be anything.

2. Enter the printer number (1,2 or 3) corresponding to the appropriate LPTx value.

3. Enter the matching driver (.CTL file name). You can get a list of drivers by pressing the F2 key.

4. Enter the Printer width, printable lines and total lines for this particular printer.

mUlti Prt Maint

Operations

a) When you choose this option the screen above is displayed. You should enter the printer information in the same order that you want it to appear during program operation. The first printer on the list will be the default value for all programs.

b) You may enter a printer name of up to 25 characters. You must enter some name or the program will assume that you are finished. Next comes the LPT number; this must be from 1 through 3. Then enter the printer driver name. You can also specify different printing specs for each printer. These include the width of the printer (in characters), the number of printable lines (generally 60), and the maximum or total number of lines (generally 66).

c) If you have only one entry here the program will not display the list during runtime.

d) The program will ask if you wish to save the entries after you have entered the last printer line or if you press the ENTER key when the Name field is blank.

TAS Merge

This program gives you a way to change clients' dictionaries quickly and easily.

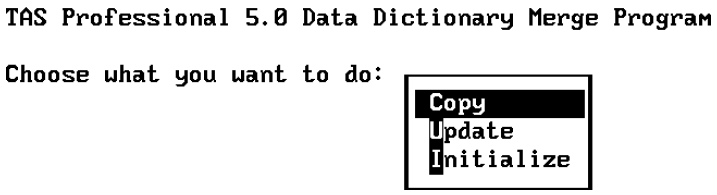
You reach this option by placing the cursor on the **TAS Merge** option line and pressing the ENTER key or by pressing the **T** key.

Program name: TASMERGE.RUN

This program was created to solve a problem that occurs when developing or modifying applications for customers. Since you cannot keep Restructure at the client site unless you purchase a copy of TAS Professional 5.1 for them, you have no way to change the client's dictionaries quickly and easily. TAS Merge helps to address that problem. It will allow you to make changes to file structures at your site and then take just those FDs that have changed to the customer site, and 'merge' them back into the customer's dictionary, at the same time restructuring the appropriate files. There is no limit to the number of files that may be copied to the .XFR files and subsequently updated at the customer site.

The program creates two files: FILEDICT.XFR and FILEKEY.XFR. These may be copied to the customer site along with the TASMERGE.RUN program itself.

When you choose this option from the Utility menu the following screen will appear.



If the entries for the .XFR files don't exist in your TASFLOC file, the program will ask if you wish to add them the first time it is run. Always answer **Y**.

TAS Merge

Operations

Copy

This will move an FD and Key info from the current dictionary to the .XFR files.

a) Enter the FD name you wish to copy. You may press the F2 key to get a list of FDs.

b) The program will copy the appropriate information and then return to the menu of choices.

c) If the FD has been previously copied to the .XFR file it will be deleted first before it is recopied.

Update

Use this option to update another dictionary and data files from the FDs in the .XFR files. This is an automatic procedure and will update all appropriate files.

> **NOTE:** Make sure you make a backup of all affected files before running this procedure (including dictionary files). If for any reason the restructure routine should fail the data will be unrecoverable.

Initialize

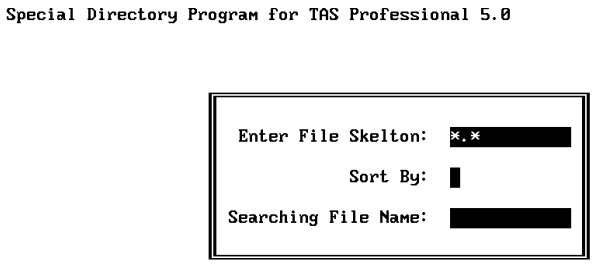
This option will remove all FDs from the .XFR files. You should do this after the Update procedure is complete so that you don't have any extra FDs the next time you run TASMERGE.

File Directory

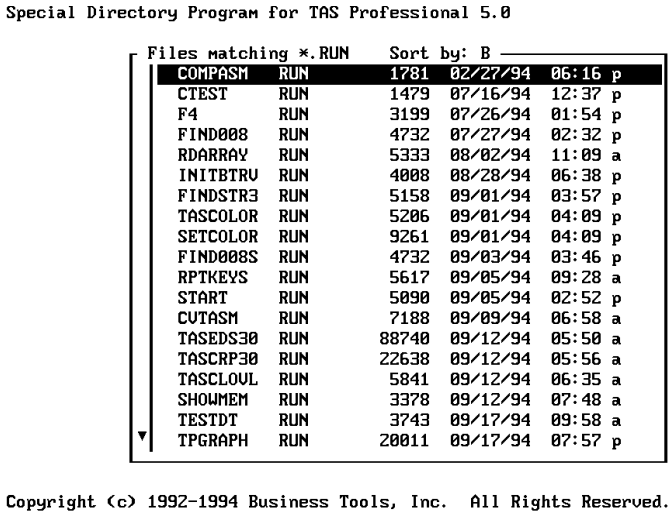
This program will allow you to quickly and easily get a listing of files within your current sub-directory.

You reach this option by placing the cursor on the **File Directory** option line and pressing the ENTER key or by pressing the **F** key.

Program name: DIR.RUN



When you choose this option a screen is displayed that will allow you to enter the files to load (wildcards allowed) and how to sort the list. Sort options are N - sort by name; E - sort by extension; D - sort by file date; T - sort by file time; B - sort by both file date and time; and S - sort by file size. The files will be displayed as they are found and then a standard list will be put on the screen. To choose a file for some action move the cursor to the appropriate line and press the ENTER key. A * will be displayed to the left of any file chosen.



Once you have chosen all appropriate files you may press ^D (CTRL+D) and delete them; ^C (CTRL+C) create COMPILE.BAT, a file with compile commands that can be executed from the DOS prompt; ^P (CTRL+P) and print them; or ^U (CTRL+U) and update the chosen files from another sub-

directory. If you press ^U the program will ask for the sub-directory to check. It will then compare the files chosen against the files with the same name in the sub-directory entered. If the file in the other sub-directory is newer it will be copied to the current sub-directory, replacing the file chosen. You can also ask for a list to be created of all files updated in this manner. The name of the file created is CPY.LST.

set system Date/time

Entering the date and time here has the same effect on your computer as if you entered it at the DOS prompt.

You reach this option by placing the cursor on the **set system Date/time** option line and pressing the ENTER key or by pressing the **D** key.

The values you enter here will be used to change the current date and time on your computer system. However, on a network it will only effect your computer.

change Company code

Use this option to change the default extension when opening TAS Pro 5.1 files within your program. You must have the appropriate files already created or the programs won't be able to open them.

You reach this option by placing the cursor on the **change Company Code** option line and pressing the ENTER key or by pressing the **C** key.

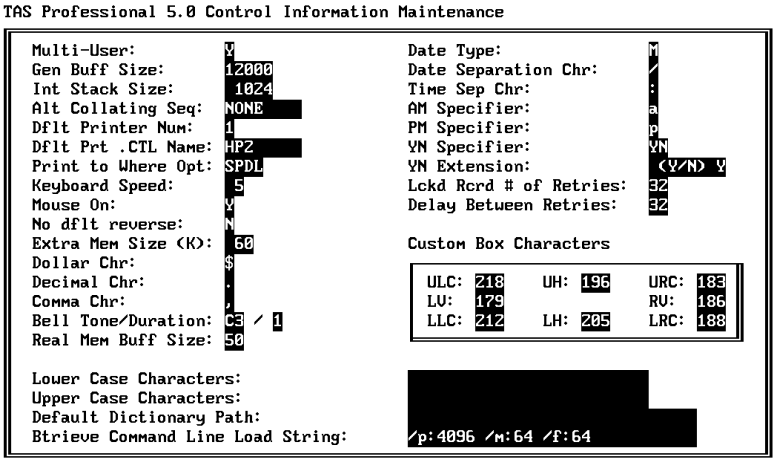
This option is a part of the Main Menu.

Set configuration

This program will allow you to maintain the TAS50.OVL file. This contains all the basic controlling information for TAS Pro 5.1.

You reach this option by placing the cursor on the **Set configuration** option line and pressing the ENTER key or by pressing the **S** key. The screen shown below will appear.

Program name: TASCINFO.RUN



Multi-User - Set to **Y** if you are running on a network with multiple systems accessing the application files at the same time. If not, leave it **N**.

Gen Buff Size - TAS Professional 5.1 creates a buffer in memory to hold certain information about each program. Each unit in this number represents 16 bytes of memory. So, 12000 is actually 192,000 bytes. This should be enough to handle the largest program. If you get an error message stating that this must be change you MUST exit TAS (return to DOS); Run TAS again, choosing this option. Change this value to a larger number. Exit TAS again (return to DOS) and start back up, again checking the program you were attempting to run. If this value is not large enough you will be informed of that at the time your are trying to start the program. Once the program is loaded everything is fine. NOTE: Do not reduce this value to less than 12000.

Int Stack Size - This is a block of memory that TAS Professional 5.1 uses to keep track of certain things during execution of a program. This is only 1k bytes and should be left at this value.

Alt Collating Seq - Used in countries that have character sets that don't follow the ASCII collating sequence. That is, the internal number used to define the character doesn't match where it actually appears in the alphabet. For example, Ö follows O; however the number for Ö is 153 whereas the number for O is 79. If everything were to be correct then it should be 80. However, that's the code for P. You can see the problem. This A.S.C. name is actually a file that is created that will give the 'new' ASCII sequence for all characters in a particular alphabet. This is used when sorting or indexing.

Dflt Printer Num - This is the number of the LPT port to use as the default printer. You can use 1, 2 or 3.

Dflt Prt .CTL Name - This is the name of the printer driver to be loaded at runtime.

Print to Where Opt - The characters that represent the options **S** (screen), **P** (printer) or **D** (disk).

Keyboard Speed - The lower this number the faster your program will appear to run. This is due to TAS Professional 5.1 'speeding up' your keyboard. The higher the number the slower the keyboard becomes. If you wish to turn it off then set this value to 32.

Mouse On - If you want the mouse to be turned on automatically upon entering TAS Professional 5.1 and to remain on when chaining from program to program then enter Y here. If you enter N here and still want to use the mouse in certain programs you will need to use the MOUSE_ON command in those programs. NOTE: If you have a mouse attached to your computer and you're used to working with it we recommend that you set this value to Y.

No dflt reverse - When a screen is displayed in TAS Professional 5.1 the entry fields are normally highlighted in reverse video (or whatever color you have chosen). This is the default process. If you DON'T want this to happen, i.e., fields are displayed in whatever the background color is, then enter Y here. NOTE: Even if you enter Y here the fields are still displayed in reverse video (the ENTER color) when they are actually being entered.

Extra Mem Size (k) - TAS Professional 5.1 allows you to create 4 'extra memory areas' that can be used for storing data during a program or when moving from program to program. This value will set the size for each of the 4 areas. Each area will have the same size. NOTE: There are several methods for getting information from one program to another, however, this option is here to maintain compatibility with the method used in a previous versions of TAS Professional.

Dollar Chr - The dollar sign character.

Decimal Chr - The decimal character.

Comma Chr - The comma character.

Bell Tone - This is the tone to be used when sounding the bell during program operation. The tone C3 is a deep sound. You may use the tones A through G and the octaves 1 through 7. We recommend that you start with this tone.

Bell Duration - The number of clock cycles the tone should last. Set this to 1 to make the bell as short as possible. You should never set this to more than 3.

Real Mem Buff Size - TAS Professional 5.1 uses extended memory for the program code and almost all of its data. However, Btrieve requires the data passed to it be in real memory (the lower 640k). You can set the amount of real memory used here. This value (in k bytes) should be the size of your largest record+500 bytes rounded up to the next 1k. For example, if your largest record is less than 4000 bytes (this is typical and is true of all CAS applications) then you would set this value to 5. This means that TAS Professional 5.1 will use approximately 25k in real memory (5k here + 20k in all cases) leaving you the rest for other programs that must run in that memory space.

Date Type - How the date appears. **M** - Month/Day/Year; **D** - Day/Month/Year; **Y** - Year/Month/Day.

Date Separation Character - The character to use between the different parts of the date value when displayed.

Time Sep Chr - The character to use between the different parts of the time value when displayed.

AM Specifier - What character to use when displaying time in AM/PM fashion to designate AM time.

PM Specifier - What character to use when displaying time in AM/PM fashion to designate PM time.

YN Specifier - The two characters that specify **Y** (yes) and **N** (no).

YN Extension - This is appended to the internal Y/N questions.

Lckd Rcrd # of Retries - The number of times to try to lock a record before an error message is displayed.

Delay Between Retries - The number of cycles to wait before attempting another retry. Each cycle is 65,535 CPU cycles. This time is completely dependent on your computer and clock speed.

Custom Box Characters - These are the characters that make up the box when you specify the C option in the BOX command. As you enter the characters the box specified will be redisplayed surrounding the characters entered.

Lower Case Chrs/Upper Case Chrs - As with the Alternate Collating Sequence, the ASCII character set was not set up to offer an easy method of determining the upper case or lower case equivalent of certain non-American language characters. By putting the lower and upper case characters (lower above the appropriate upper case) in these two fields the program can do the work that has to be done outside the normal method.

Default Dictionary Path - This should be the path where the Dictionary files (FILE*.B) are located. By putting the path value in here you will be able to access these files no matter where you are on the computer.

Btrieve Command Line Load String - TAS Professional 5.1 uses the Btrieve™ Database Manager as its file handler. This needs to be loaded whenever TAS Professional is run. This line tells Btrieve how to load.

After saving the information you must exit to DOS and reload TAS Professional 5.1 before any of the changes made take effect.

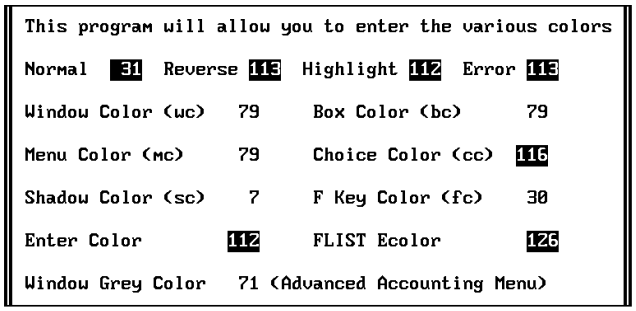
> **NOTE:** Each user in a multi-user system may have their own copy of TAS50.OVL by having them log into the proper path that contains their individual file and then executing TAS. As long as there is an entry in your path statement (in AUTOEXEC.BAT) pointing to the TAS Pro 5.1 subdirectory and the option to SET TAS50= you will need just one copy of the .EXE files and programs. The user will automatically execute the proper program. TAS Pro 5.1 automatically looks for the TAS50.OVL file in the default (user's) subdirectory, not in the TAS50 subdirectory. However, if it is not found in the user's subdirectory, the program will use the one in the TAS50 subdirectory.

set coLor

This program will allow you to maintain the TASCOLOR.OVL file. This contains all the basic color information used in TAS Professional 5.1 programs.

You reach this option by placing the cursor on the **set coLor** option line and pressing the ENTER key or by pressing the **L** key.

Program name: SETCOLOR.RUN



Normal - This is the normal background color.

Reverse - Default field display color.

Highlight - Menu bar color.

Error - The error (and general) message color.

Window Color - The color used for windows.

Box Color - The color used for the boxes that surround windows.

Menu Color - The color used for menus.

Choice Color - The color used for the list cursor bar.

Shadow Color - The color used for shadows behind windows.

F key Color - The color used for displaying key options at the bottom of the screen.

Enter Color - The color used when entering fields.

FLIST Ecolor - This is the color used to highlight characters entered in the LISTF command if the option is used.

Window Grey Color - This is used in the Advanced Accounting Main Menu to set off the inactive menu.

You may press the F1 key to get a listing of all colors.

In TAS Professional 5.1 the system looks for a copy of TASCOLOR.OVL in the user's current path. If one is not found it then looks for the default copy in the TAS Pro path. There must be one on the system or the program will not load. By allowing a user to log into his/her own sub-directory you can

give them their own color file and let them choose their own colors.

If you save the record you must exit to DOS and then reload TAS Professional 5.1.

Edit ascii file

You reach this option by placing the cursor on the **Edit ascii file** option line and pressing the ENTER key or by pressing the **E** key.

This option is a part of the Main Menu and chains to TASEdit™.

You may enter the name of the file to edit or press the F2 key to get a list of files. When you press ENTER, the Main Menu chains to the TASEdit™ program and you will be able to edit the file you have chosen. When you press ESC you will be returned to the Main Menu and will be able to save the file back to disk, if desired. This option is provided here as a simple method of accessing small text files and is not meant to be a full-blown editor. However, you will find this very useful when you need to make quick changes to any ASCII file.

For more information about TASEdit please refer to the beginning of this chapter.

TAS
PROFESSIONAL
VERSION 5.1

Chapter 4

Compiler Information

INTENTIONALLY BLANK

INTRODUCTION

Every TAS Professional 5.1 program must be compiled before it can be executed. When a program is compiled several things occur. Each field used in any command or expression is checked to make sure that it has been defined or is part of a record defined in the data dictionary being used. Each command is checked to make sure it exists and that the options are legal; problems with these requirements cause general syntax errors. And finally, the resulting program, with all its bits and pieces, is saved. This entire process can take as few as 2 seconds or several seconds, depending on the computer and the size of the program. The compiler makes just one pass through the source file. All adjustments are made after the compilation is finished. If errors occur, the program will give the programmer the line number, the offending value, and what the error was.

Once the compiler is complete the program will display the sizes of different parts of the finished program. This information is really just for your enlightenment. However, it will keep you informed of program sizes so that you can keep track of memory usage.

The code produced by the compiler is not just tokenized. It is truly compiled, just not to the proper form for direct execution on an Intel processor.

DEFAULT COMPILER BUFFERS

The following are the buffers that are predefined in the TAS Professional 5.1 compiler. Please see Command Line Changes below for more information. Particular settings specific to a single program may also be entered into the CMPDFLT.B file using the COMPINFO.RUN program. For more information about this program please refer to **Chapter 3, Main Menu - Program**. There is a record in the CMPDFLT.B file with the title DEFAULT. This record of default buffer sizes is used by the compiler when it cannot find a record set up specifically for the file being compiled. If you are running out of room when you are trying to compile a program, you might want to reduce the values in this record or even delete it completely. Here's more information on the various buffers:

Buffer	**CL code**	**What does it do**
Run code	R	Each line that is compiled requires 10 bytes. This buffer, at actual size, is loaded during runtime.
Constant	C	Each constant used in the program is saved in this buffer, along with compiled screen/report formats. This is the most likely buffer that will have to be changed during compilation. This buffer, at actual size, is loaded during runtime.
Spec line	S	Each command has a certain number of bytes that defines it. Those bytes are saved here. This buffer, at actual size, is loaded during runtime.
Screen/Report fmt compile buffer	B	When a screen/report format is compiled this is the temporary buffer that is used to store the compiled code. This value will generally be more than large enough. Once the format has been compiled, it is stored in the constant buffer.
Internal fields (num)	I	The program uses these internal fields while resolving expressions. This should be enough. However, if you

		get an error at runtime saying that there are not enough internal fields, then this value needs to be increased and the program recompiled. Each field requires 32 bytes in the field list, and a maximum of 2000 internal and regular fields may be specified. The actual number of fields specified is loaded at runtime. Please also refer to the Compiler Directive below about Temporary Data Space.
Regular fields (num)	F	Each field specified, both those created with the **DEFINE** command and those from the data dictionary (part of a file) require one slot in the field list. Arrays only require one slot (unless the element specifier is a constant, then it is a separate field). Each field requires 32 bytes in the list and a maximum of 2000 internal and regular fields may be specified. The actual number of fields specified is loaded at runtime.
Line labels (num)	L	Each label, during compilation, requires 16 bytes. In the final program each label takes 2 bytes. The actual final buffer size is loaded at runtime.

Please note that larger buffers require more memory during compilation and affect your ability to compile while other programs are loaded. These buffers, most of the time, will be more than large enough. When they aren't it is very easy to change them. The compiler will alert you that it has run out of room when it comes within 100 bytes of the size of the buffer or within 10 of the number of fields or labels.

COMPILER COMMAND LINE OPTIONS

Compiler buffer sizes, error message control and others can be changed by including control codes on the command line when invoking the compiler. This is accomplished by including the code letter preceded by a minus sign (-) and followed by the appropriate information. For changing buffer sizes the programmer would set the new buffer size in kbytes (1 kbyte is 1024 bytes). Other options require specific characters to be included. You may include multiple codes at one time; however, there must be at least one space between each of them. Also, every control code must be preceded by a '-', immediately followed by the appropriate character and then the required parameters, if any. All control codes must follow the program name to be compiled.

The compiler line codes for the various buffers are listed above (CL code). The other compiler line codes are:

Code Name	**CL code**	**What it does**
Redirect compiler output	P	This code can redirect the output from the compiler to either the printer (-PP) or to the disk (-PF). If the output is to the disk a file will be created with the file name and an extension of .ERR.
Add debug code	D	If this control code is used the compiler will include program name and line number information in the runtime code. This will add approximately 5 bytes per line of code. With this information the runtime will display at the top of the message window the source code file name (even if a program is made up of multiple source code

		files) and the actual line number currently being executed. This is very helpful in determining exactly where in a program an error has occurred.
Allow Translation	T	This code will notify the compiler to open the TASTRNS.B file. If a command or option cannot be found in the normal list, the compiler will check this file for a translated value. If it still can't be found an error will be displayed. For more information about the translation file please refer to **Chapter 3, Main Menu - Program**.
Alternate Dictionary	A	This code will allow you to specify an alternate data dictionary to be used during compilation. Normally the compiler will use the dictionary in the path pointed to by SET TAS50=. For example, if you have SET TAS50=C:\PROF then the data dictionary (all 8 files) would be expected to be in that C:\PROF subdirectory. This is all overridden by the value you have entered, if any, in TAS50.OVL as the default dictionary path.

The following is a example of a compiler command line where the size of the constant section is being increased to 24k, the output is directed to an error file, and you want to use a dictionary in the path C:\ACCTG\SRC:

```
TPC50 -C PRGNAME -C24 -PF -AC:\ACCTG\SRC
```

NOTE: Since the compiler is part of the TAS Professional 5.1 runtime the -C after the TPC50 tells TAS to compile this program instead of running it.

The error file created will be PRGNAME.ERR, the constant buffer will be increased in size to 24k or 24756 bytes, and the alternate dictionary C:\ACCTG\SRC will be used. The use of lower or upper case characters does not matter.

If a buffer size is exceeded the compiler will alert you to that problem. You can then adjust the buffers as needed and compile the program again. In the case of the number of internal fields the problem may not be seen until the program is run. The solution is still to recompile the program with the -I number set to something greater than 30. In a situation of this sort you can generally increase the number of internal fields to 35 or 40 and will have more than enough. In general, if you have an expression so complex that you find the need to increase the number of temporary fields you are better off breaking the expression into two parts.

COMPILER DIRECTIVES

These are instructions that are actually part of a program. The programmer includes these to force the compiler to take certain actions. A compiler directive starts with a # character as the first character on the line. There may be only a single directive on a line. Each directive has its own use and is explained below:

Directive	**What does it do**
ADD_FLDS	During execution of a program the programmer can add fields to the field list thru the use of the **ADD** command. However, for this to be accomplished the programmer must add extra fields to the field list during the

compilation process. This compiler directive will do just that. The example below will add 10 extra field spaces to the field list:

```
#ADD_FLDS 10
```

ALL_LOC

This directive will have the same effect as including the **LOCAL** option in all **DEFINE** commands following the directive. This directive would have to be after the PROC directive to have any effect and will be effective until the next ENDP directive. Example:

```
#ALL_LOC
```

CHK_UP_VLD

Normally, the VALID option in the **ENTER** command is always checked unless the user presses the ESC or Up Arrow key. If you do want to check the VALID option, even if the Up Arrow key is pressed, include this directive. Example:

```
#CHK_UP_VLD
```

ENDP

This directive is the termination for the PROC directive. Once this directive is reached it will terminate the scope for all **LOCAL** defined fields within the routine from the previous PROC directive to this line. If a field with the same name is used outside of the routine bounded by the PROC/ENDP directives it will be treated as a different field. Example:

```
#ENDP
```

EXT_FMT

In TAS Professional 5.1 all screen/report formats are normally included as part of the source code. However, if you wish to have them as separate files on the disk use this compiler directive. If this is set then the program expects all formats (no matter what type) to be separate for this program. You can also use the EXTERN option in the MOUNT command if you want to keep just a few formats separate. This would apply when you have a format you use in several different programs.

```
#EXT_FMT
```

FMT

If you are using 'old type' screen report formats then this option must be set. If the PRO3 compiler directive is also set then the compiler expects the screen/report formats to be in TAS Professional 3.0 form. If PRO3 is not set then you must use the TAS Professional 4.0 method.

```
#FMT
```

INC

You may specify program parts or subroutines not originally part of the source code that you want to be compiled (or included) with it. These parts can be named here. You should note that the included programs will be attached at the end of the original named program. These parts should be subroutines, UDFs, or UDCs to make sure that the execution of the final routine will be as desired or you must overtly transfer control to them through the use of the GOTO or GOSUB command. An example of an include file is:

```
#INC C:\ACCTG\XRTN
```

The extension .SRC need not be a part of the program name. The compiler expects that all parts of the same program are of the same type as the base (or original) source code file.

LIB

You might want to keep a library of commonly used routines for use in any program. This will allow you to maintain only one copy of a set of code and load the pieces required in your program automatically at compile time. To specify the appropriate library file put the name after the LIB directive, i.e.:

```
#LIB TASRTNS
```

The compiler expects the file to reside in the same directory as the compiler unless you specify the appropriate path. With TAS Professional 5.1, you also have the ability to use a SET TASLIB= to set a path in your environment (generally in your AUTOEXEC.BAT file). This is similar to the SET TAS50= in TAS Pro 5.1 and the SET TASPRO= command in 4.0. If you use a SET TASLIB= statement, the compiler will look for all libraries in the path given. This means you can have one set of libraries for all programs you're developing.

A library has been provided to you that includes several useful commands and functions. It is TASRTNS.LIB.

PRO3

This directive tells both the compiler and the runtime that the program is a direct conversion from TAS Professional 3.0. Putting in this directive activates certain features that are different from the native TAS Professional 5.1 process. These include:

a) The #OLD_MATH compiler directive is automatically set. This means that expressions are evaluated from left to right instead of using the operator hierarchy. (For more information, see above.)

b) If you have a **POPS** (Pop Stack) command in your program and have no corresponding GOSUB (i.e., you're at the top of the stack), the command will be ignored rather than giving an error.

c) In the TAS Professional 5.1 version of the GFLD() function (Get Field from Buffer), if you don't provide a file handle, then you must provide a field name. With the #PRO3 directive in place, the program looks for the active non-TAS file number automatically if you don't provide the file handle as the first option in the function body.

d) If you are moving a BCD (new type O) field to an alpha type field using the **EQUAL** (=) command, the program will truncate the BCD field if it is longer than the alpha. This means what was a 5 character BCD field will fit in a 2 character alpha field if the value is less than or equal to 99. However, even if the BCD value is greater than 99, the program will only move the last 2 characters. This only applies to BCD type fields. New I, R, and N fields are moved from left to right without truncation unless you use the appropriate commands or function.

e) In TAS Professional versions 4.0 and 5.1, when you open a non-TAS fixed length record, the program expects you to add 2 bytes to the record size to allow for the CR/LF at the end. In TAS Professional 3.0, this wasn't the case. When this directive is set, the program uses the version 3.0 rules when reading non-TAS fixed length records. This includes not clearing the CR/LF pair when the record is read.

f) In TAS Professional 3.0, the Help (F1), Page Break (PG_BRK) and Related Record Search (RSRCH) traps are all forced to GOSUB if you set it as GOTO. The same applies to TAS Professional 5.1. Please note that the conversion program automatically converts a GOTO for these traps to GOSUB, but even if you force the command to a GOTO, it is interpreted as a GOSUB.

g) When using the ENTER option in a **PMSG** command, the UPCASE flag is always set.

h) When concatenating (adding together) alpha type fields, TAS Professional 3.0 only moves those characters greater than binary 0. This directive activates the same restriction.

i) When displaying or printing, the process will quit when a CR is encountered.

j) The WAIT option in the **PMSG** command ignores the ESC key.

k) The RCN (Get/Set Record Number) command in TAS Professional 3.0 expected a 4 byte alpha field, whereas TAS Professional versions 4.0 and 5.1 use the new R type (4 byte integer) field. If the #PRO3 directive is used, the program will use either type of field.

l) In TAS Professional versions 4.0 and 5.1, if the FOR/NEXT loop runs to completion, the counter is 1 more (or 1 less if the step is -1) than the stop value. In version 3.0, the counter would never be more (or less) than the stop value.

PROC

When you **DEFINE** fields within a program you may specify that they are **LOCAL** to that routine. To 'turn on' the local option you must preface the routine with this directive. Then all defined fields until the next ENDP directive will be local to that routine only. For example, add the following:

```
#PROC
```

For more information about the effect of **LOCAL** on a field please refer to the **DEFINE** command in the reference section, **Chapter 5, Command Reference**.

SFLDS

When the source file is compiled, the program keeps track of the screen formats that are compiled. A buffer of screen fields is created in the runtime that will hold the largest of the screens. The compiler defaults to 10 screen fields even if no screen formats are compiled. If you should want to use the **SAY ... AT** command as part of the program be sure to specify the maximum number of fields to be added to the screen field buffer through the use of this directive. The amount given here will be added to the maximum number already calculated during the compilation process. You should put this directive before the first **MOUNT** command it applies to, in order to be sure that it is carried out correctly. An example is:

```
#SFLDS 20
```

NOTE: Always to be sure to **MOUNT** a screen format before using the **SAY...AT** command. This will make sure that the screen buffers are properly set up with the program before the new fields are added. You may

MOUNT a blank format if there is nothing else you want displayed on the screen.

TDATA

Along with the internal fields a certain amount of data is maintained for internal field use. The default value of this directive is 64K bytes. This should be more than enough for most programs. However, if you receive an error message stating that temporary data has been exhausted, this directive will increase that amount for the specific program. You are given control over this value since a larger buffer just takes up more space. Just specify the number of bytes (characters) to be allocated to this buffer. For example:

```
#TDATA 25000
```

will allocate 25000 bytes to the temporary data buffer.

UDC

You must include this directive if you are going to have User Defined Commands (UDCs) within the program. If you do not, and you try to use a command that doesn't exist normally in TAS Pro 5.1, you will receive an appropriate error message. To activate this option include the following:

```
#UDC
```

No other entry is required.

UDF

You must include this directive if you are going to have User Defined Functions (UDFs) within the program. If you do not, and you try to use a function that doesn't exist normally in TAS Pro 5.1, you will receive an appropriate error message. To activate this option include the following:

```
#UDF
```

No other entry is required.

UDX

This directive allows both UDCs and UDFs.

XLATE

TAS Professional 5.1 offers an option unique in a 4GL. If you don't like the command or option names, create different ones. Thru the use of the translation file you can assign new names that will be the equivalents of the current ones. Then you can use either one within the program. If this directive is specified, the compiler will look for the appropriate command or option within the translation file. To turn this option on set the directive:

```
#XLATE
```

No other entry is required. The translation file (TRANSLTE.B) must be in the same directory as the compiler.

NOTE: This has the same effect as the -T option in the compiler command line.

MISCELLANEOUS INFORMATION

CONTINUATION CHARACTER

The maximum length of any single line is 1024 characters (bytes). However, you may want to keep lines at a reasonable length. For example, you might want no lines to exceed 80 characters so that the program will print easily on narrow paper. Some commands, if fully used, could not fit within those 80 characters. So, we have provided a method for continuing a command on several lines. This is accomplished by making the last character on the line a continuation character, or a backslash (\). Nothing else, including remarks, can be past the \ . The compiler literally concatenates the following lines until no more continuation characters are found. The compiler automatically removes all leading spaces so that:

```
? 'This is \
   a test'
```

will actually end up like this:

```
'This is a test'
```

If you load a program into the Program Editor that contains continuation lines they are automatically removed during the loading process.

> **NOTE:** VERY IMPORTANT!! If you have a continuation character at the end of a line, it will be treated as such even if it is after a remark character. Remark characters include both ';' and '&&'. If you can't figure out why a line isn't being compiled, look for this situation immediately before that line.

In TAS Professional 5.1 the compiler will allow comments or remarks on the same line as the command using the semicolon (";") character. For example:

```
XTRAP SAVE IGNR      ;this will create a buffer to save the traps and the buffer
                     ;will be eliminate automatically when we do XTRAP RSTR
```

MULTIPLE COMMANDS PER LINE

The programmer can put multiple commands on each line as long as they are separated by the character |. Each command will still be treated separately. For example:

```
X=1 | Y=2 | Z = 'ABC'
```

is equivalent to:

```
X=1
Y=2
Z='ABC'
```

NOTE: The Program Editor cannot edit lines with more than one command per line. You must either edit these using the Alt_E option or convert them to multiple line commands.

LOCATION OF SOURCE CODE

You may compile programs that are in a different path. Just append the proper path to the source code file name.

NOTE: All screen/report formats and included files for that particular program must be in the same path as the main source file if they are not a part of the main source file.

INTENTIONALLY BLANK

TAS
PROFESSIONAL
VERSION 5.1

Chapter 5

Command Reference

INTENTIONALLY BLANK

CONVENTIONS USED IN THE COMMAND REFERENCE SECTION

The following format is used in documenting the TAS Professional 5.1 commands.

- The commands are listed in alphabetical order.
- The first paragraph of each command reference section is a brief definition of each command and what it does. This provides a quick insight into the command, and what you might use it for.
- Next comes a list of each part for the command. Parameters and options are separated by semi-colons; refer to the explanations below for more details on typical command elements. This is not necessarily the true syntax for the command but just a listing of options with the appropriate names.
- Each command part is described, along with the purpose, valid values, default value (if applicable), consequences, and whether or not the part is required.

  Your entry portion can consist of the following:

  f/c/e - Field/Constant/Expression. You can use any type of field or constant. This can consist of alpha or numeric constants, any type of field or variable field. This can also be an expression that results in the proper type of field. Some options allow you to include multiple values, e.g.:

  PMSG *val1,val2,...,valx*

  From one to eighty values may be included. Generally, the 1024 character line length rule will be broken before the maximum number of fields can be used. In any situation where different limits apply the documentation so indicates.

  file_expr - A file may be referred to in two different ways. If it was opened using the standard **OPEN** command then it must be referred to by the file name as an alpha constant without the quotes. TEST refers to the file name TEST and not the value ‘saved in’ the field TEST.

  If the file was opened using the **OPENV** (Variable_Open) command then a file number was returned to a field within the command. Use that field name preceded by the "at" sign (@) when referring to the file. @TEST tells the program to use the file number ‘stored in’ TEST when accessing that file.

  flist - In any option, in almost any command that would normally allow a list of field names or expressions, you may use an array of F type pointers instead. In that case you would use the special modifier **flist** and follow it with the name of the F type pointer array field. For example, in the **EXPORT** command you need to specify a list of **from** fields. Instead of actually listing those fields you may set an array of F type pointers to point to each one and then in the command use the following:

  EXPORT *flist fptr_array*

  This will have the same effect within the program as though you had named each one of the fields individually.

  NOTE: You must use an array of F type pointers only. P type pointers cannot be used. The array that you use must be at least one longer than the maximum number of elements you are going to use. To get around the problem of how to access a specific

element in an array field using this process you should note that any single field--array element or not--can be added (the **ADD** command) to the field list in a program. All you need to do is include the array specifier in the field name and a slot will be created for that element. The command will return the fptr value appropriate for that particular element, and you can use it as you would any other fptr.

fn/v - Field name/Variable field. In certain commands/options you must use the actual field name or a variable field. Some options allow multiple fields, e.g.:

TO *fn/v1, fn/v2,..., fn/v3* (in the **RDA** - read array - command)

You may include only one field reference or up to 80. Generally, the 1024 character line length rule will be broken before the maximum number of fields can be used. In any situation where different limits apply the documentation so indicates.

key_expr - You may define the key number to be used in a command in several ways:

number - The actual number of the key, as defined in the data dictionary, may be used preceded by an "at" sign (@). *@3* refers to the third key in the appropriate file (the first key is 1).

key name - The name of the key as assigned in the data dictionary may be used. *dict_overlay* refers to the key name.

no key - If you do not want to use a key but want to access the file through the direct method then use the "no key" symbol, the number 0 preceded by an "at" sign (@). *@0* tells the program to use the direct access method when searching this file. This is the default method when you have opened a non-TAS file.

label - This option is a line label. A line label is the only item on the line, is no more than 14 characters long and must end with a colon (:). A label is a form of a special alpha constant (sac). An example:

```
START:
```

This label is used within a command by using the name but dropping the colon, e.g.:

```
GOTO START
```

If you use a label name that doesn't exist within the program the compiler will give the appropriate error message at the end of the compilation process.

NOTE: In the Program Editor you have the opportunity to receive a list of label names used within the program at the appropriate times.

lexpr - An expression that will resolve to a logical value of .T. or .F. For example:

```
fld1 = 100
```

another example:

```
(fld1 = 100 .a. fld2='ABC') .o. fld3=.F.
```

The logical expression:

fld1=100 .o. 200

is not legal. You cannot compare to more than one value at a time. This expresion should be written:

fld1=100 .o. fld1=200

prt_where - This is part of the **PTO** (print to) option. In commands that produce some sort of output you may choose where to send it. The options include:

S (screen)
P (printer) - current printer (change through **PRT_NUM** - printer num -command)
D (disk) - you can also specify the file name
A (ask) - ask the user at runtime (some commands do not allow this option)

In some commands the D refers to default device. In these cases you will be able to specify only S, P, or D (default). These are explained further in the command itself. The option for these commands is **PTW**. Please note that the List option ("L") is only available in Business Tools' accounting products.

sac - Special Alpha Constant. This is an alpha constant that is not surrounded by quotes. It must start with an alphabetic character (A-Z) but may contain numbers, periods or the underline character. One form of this is a line label. Another is the name of a user defined function or user defined command. Unless otherwise stated, case is ignored and the maximum length will be stated, although 14 characters is generally the maximum.

scope - Use this option, along with the **scope_expr** to limit the number of records accessed. The scope options are:

A (All) - Access all records, this is the default
F (First) - The program will find the first record in the file and then access the next xxx number of records depending on the **scope_expr**.
N (Next) - The program will start with the current record or if no record is active will find the next record in the file. It will then continue with the next xxx number of records depending on the **scope_expr**.
R (Rest) - The program will start with the current record or if no record is active will find the next record in the file. It will then continue until the end of the file.

scope_expr - Use this option to determine how many records to read. The options available for **scope_expr** are: **A** (all), **F** (first), **N** (next), **R** (rest). In the case of F and N you also specify the number of records to be read (e.g., N 100). For example, if you want to restrict a search to the first 100 records the entry for the **scope_expr** would be:

F 100

If the scope is set to N and a record is active the program will start with that record. If a record is not active the program will get the next record before continuing.

udf - Some options require UDFs (User Defined Functions) as the possible entry. What is expected and the required returns are described.

set_option - In this situation your choices are to include the option name or leave it off. The command documentation explains first the default situation (what will happen if you

don't include the option), and then what will happen if you do include the option. Generally, including the option will keep something from happening. For example:

SAVE file_name/@file_num **NOCNF NOCLR**

In the **SAVE** command above, the **NOCNF** option tells the program **not** to ask the user before saving the record. The default case is to ask before saving. The **NOCLR** option tells the program not to clear the record buffer (and record number holder) after saving so that the information just entered into the record is still there, perhaps to be used as default values during the following operations. The default operation is for the program to clear both the record buffer and number holder after saving.

In the command documentation, these **set_options** are shown in italic, capital letters.

In the Program Editor you would choose these options by answering **Y** or **N**. The Y answer will include the option, the N won't.

- Each **OPTION** within the command will be described fully.
- The true **SYNTAX** for the command is described for each of the three different source file types. The command name and options are in **BOLD CAPS**. The parts the user needs to enter are in *italics.*
- In some commands there is a special **USER INTERFACE**. In these situations it is described.
- Special **COMMENTS** or restrictions are given where applicable.
- **SEE ALSO** will list the commands/functions/traps that are related to the command being explained.
- If any **EXPRESSION FUNCTIONS** (not to be confused with the function of the command but a type of command that can be part of an expression) are included in the explanations they will be in bold face with parentheses at the end of the name, e.g., **FUNCTION()**. Even though the function may have arguments that can be included, it will be shown with no arguments within the parentheses. For more information on the function and what arguments may be used, please refer to that function's documentation.

  NOTE: Spaces between multiple entries for an option are for appearance purposes only. For example: *f/c/e1, f/c/e2,..., f/c/ex* need not have any spaces between the commas and the preceding or following fields. However, options within a command need to be separated by at least 1 space.
- **EXAMPLES** may also be given.
- If any other command names are displayed (other than the title) within an explanation they are given in actual syntax in bold format followed by an explanatory title surrounded by parentheses, e.g.,

  ... then you might use **WINACT** (Window Activate) to do ...
- (DOS Only) - If the command does not execute in the Windows environment then this line will be next to the command name at the top of the information block. In this situation the command will be ignored when running in Windows but will not cause an error.

- (Windows Only) - If the command does not execute in the DOS environment then this line will be next to the command name at the top of the information block. In this situation the command will be ignored when running in DOS but will not cause an error.

? (PRINT MESSAGE)

Please see the **PRINT MESSAGE** command.

; or * or && (REMARK chrs)

Please see the **REMARK** command.

{...} (CODE PROCESS CONTROLS)

These single character commands will allow you to group appropriate code together without worrying about program flow. For example, if you have an ENTER command that has a PRE and POST option and you want to have all appropriate code in one section you could write it as follows:

```
ENTER CUST_NAME PRE SetupSearch() POST ClearSearch()
      {
      func SetupSearch
            Trap F2 gosub DoCustSearch
            ret .t.
      func ClearSearch
            trap F2 dflt
            ret .t.
      }
```

When the program is running it will execute the ENTER command but will skip all other commands after the ENTER until the next command after the }. However, the commands within the braces are not invisible. They are still executed properly by the PRE and POST calls and can even be called by other commands before and after this ENTER. The only purpose for these controls is to keep related code together within your program.

ADD

This command will add fields to the internal field list. These fields may be a part of a file that has been opened or just for standalone purposes. This file could have been specified in the original program or could have been opened during the execution of the program. Once the field is added you can access it almost as though it had been named in the original program. An 'F' type pointer is returned and by using the field redirector (&) you can access that field for any purpose, including **ENTER**, **=**, etc.

ADD *field_name* **TYPE** *type* **SIZE** *display_size* **DEC** *decimal_chrs*
ARRAY *num_array_elements* **UP** *up_case* **PICT** *picture*
FILE *filename/@file_number* **KEY** *key_number* **OFST** *offset_in_file* **FPTR** *field_ptr*

field_name - f/c/e - Required - The name of the field being added. Must conform to standard field name requirements.

type - f/c/e - Required - The type of the field being added. Must conform to standard field type requirements.

display_size - f/c/e - Optional - The display size of the field being added. Must conform to standard field display size requirements. If this is not included then the default values for the type of field is used.

decimal_chrs - f/c/e - Optional - If the field is of type N you can specify the number of decimal characters. If this isn't included the default value is 0. This could affect the displayed value of the field or the ability to **ENTER** the correct value. Because this is true, you should take great care to make sure the proper value is used.

num_array_elements - f/c/e - Optional - The number of array elements for this field.

up_case - f/c/e - Optional - If this is a type 'A' field you may specify whether all characters entered to it are to be automatically forced into upper case. To accomplish this, make this value 'Y'.

picture - f/c/e - Optional - If this is a type 'A' field, you can use this option to enter a field into a space that is shorter than the defined length of the field. This is called a slider field. A slider is a field that shows fewer characters on the screen than actually exist in the field. The user can see the other characters by pressing the LEFT and RIGHT ARROW keys and the characters will appear to 'slide' back and forth across the available space. This is very useful when you have several large fields and want them all to fit on the same screen. A slider field is specified with an alpha constant of "Sxx", where xx is the number of columns to display.

filename/@file_number - file_expr - Optional - The file name or number for the field being added if it is part of a record.

key_number - f/c/e - Optional - If this field is a key you can specify the key number here. Then if this field is used in the **ENTER** command the user will be able to search for records using the file search keys (F5-F9).

NOTE: If the **SRCH** (Set Search File) command has been executed, setting a specific search file and key, it will override any other key fields.

offset_in_file - f/c/e - Optional - If this field is part of a record then this is the starting location within the record. The offset value for the first field in the record is 0.

NOTE: In the TAS Professional 5.1 data dictionary the offsets for the fields are properly set for this option; i.e., the first field in the record has an offset value of 0.

field_ptr - fn/v - Required - The field that will receive the pointer to the field being added. Since you cannot access the field by name (it isn't known before being added), this pointer is returned and will allow you to execute all required commands through the use of the field redirector. For example:

ADD ... **FPTR** *fldptr[cntr]* ...

where fldptr is of type F and an array. *Cntr* is an integer (type I) field that is set to the appropriate value. The field can be used later in an **ENTER** command:

ENTER *&fldptr[cntr]* ....

NOTE: This field can also be added to the screen fields through the use of the **SAY** command. If the field is not added to the screen fields, the **ENTER** command above must include the **AT** option (column,row). Please see the **ENTER** and **SAY** commands for more information.

COMMENTS

A basic concept in every TAS Pro 5.1 program is that all fields have been specified when the program is compiled. This restricts the possibilities of what might be done. To get around this limitation this command has been added. If the field being added is NOT from a file the program automatically allocates data space for it from that remaining available. This DOES NOT take space from the temporary data space used when resolving expressions.

NOTE: You need to specify, within the source file, how many 'extra' fields can be added at runtime. This is done through the use of the #ADD_FLDS compiler directive. For more information please see the section in **Chapter 4, Compiler Information** on Compiler Directives.

PROGRAM EDITOR

Field -> Create/Chg -> Add

SAMPLE PROGRAMS

ADDTEST

ALLOCATE FIELD

Use this command to change a field previously defined within a program. This command will work only with **DEFINE**'d fields. The command will allow you to change any part of the field: size, type, etc., even the number of array elements. It is generally used in situations where you don't know how large a field you might need and don't want to take up space before it is needed. Also, you can **DEALLOC** (Array Deallocate) an allocated field during runtime. This capability is useful when you know you need to chain to other programs and might not have enough room if the field is at a set size.

ALLOC *field_name* **TYPE** *type* **SIZE** *display_size* **DEC** *decimal_chrs*
ARRAY *num_array_elements* **UP** *up_case* **PICT** *picture*

field_name - fn/v - Required - The name of the field being allocated. This field must be previously defined within the program.

type - f/c/e - Required - The type of the field being allocated. Must conform to standard field type requirements.

display_size - f/c/e - Optional - The display size of the field being allocated. Must conform to standard field display size requirements. If this is not included then the default values for the type of field are used.

decimal_chrs - f/c/e - Optional - If the field is of type N you can specify the number of decimal characters. If this isn't included the default value is 0. This could affect the displayed value of the field or the ability to **ENTER** the correct value. Because this is true, you should take great care to make sure the proper value is used.

num_array_elements - f/c/e - Optional - The number of array elements for this field.

up_case - f/c/e - Optional - If this is a type 'A' field you may specify whether all characters entered to it are to be automatically forced into upper case. To accomplish this, make this value 'Y'.

picture - f/c/e - Optional - If this is a type 'A' field, you can use this option to enter a field into a space that is shorter than the defined length of the field. This is called a slider field. A slider is a field that shows fewer characters on the screen than actually exist in the field. The user can see the other characters by pressing the LEFT and RIGHT ARROW keys and the characters will appear to 'slide' back and forth across the available space. This is very useful when you have several large fields and want them all to fit on the same screen. A slider field is specified with an alpha constant of "Sxx", where xx is the number of columns to display.

COMMENTS

Typically you would include a group of single character alpha fields in a program. At time of need, these fields could be changed using this command. To see if the command succeeded, check both the size of the field and the number of array elements allocated using the appropriate functions available. These fields are allocated into available memory and DO NOT take any space away from temporary data space used while resolving expressions. You want to start with as small a field as possible since the space originally used by that field will be unusable.

PROGRAM EDITOR

Field -> Create/Chg -> aLlocate

SEE ALSO

REMVA, ALOCARY()

ARRAY DEALLOCATE

Please see the **REMOVE ARRAY** command.

ARRAY FIELD DISPLAY

Please see the **DISPLAY ARRAY FIELDS** command.

ARRAY READ

Please see the **READ ARRAY** command.

ARRAY SORT

Please see the **SORT ARRAY** command.

ARRAY UPDATE

Please see the **UPDATE ARRAY** command.

ARRAY WRITE

Please see the **WRITE ARRAY** command.

ASK

This command provides you with a method of getting the user's response to Y/N questions using a standardized process.

ASK *question* **DFLT** *default_response*

question - f/c/e - Required - You may use this in two ways. It can be an A type field and the program will display the field as the message. It can also be a numeric value (any appropriate type), and the program will read the appropriate record in the ERRMSG.B file. The value you set must relate to the ERROR_NUM field in the ERRMSG record. For more information please see **Chapter 3, Main Menu - Utilities**.

default_response - f/c/e - Optional - The normal default response to this command is .T.. This would be the case if the user pressed just the RETURN or ENTER key without entering a value. If you would like the default response to be N (.F.) then this value must be set to .F.

COMMENTS

The **ASK** message will be displayed at the same location as the error message. The message will be word wrapped properly and the program will back up the cursor one character. To use this feature you should make sure that the last character in the message is a 'Y' or 'N' as appropriate. You can test the user's response to the message through the **ASK()** function. If the **ASK()** function returns .T. the user answered **Y** to the question; if .F., **N**. If the user presses the ESC key at the question, the **ASK()** function returns .F. Also, you can test if the user pressed the ESC key through the use of the **ESC()** function. It will return .T. if the user pressed the ESC key.

PROGRAM EDITOR User interface -> Messages -> Ask

SEE ALSO ASK()

AUTO DECREMENT LIST

This is used in connection with the **LISTM** (List Array) and **LISTF** (List File) commands. If you are executing one of those commands and are in a routine 'outside' of the command (e.g., an enter process), you can use this command to automatically force the cursor to go back to the previous line in the list when you return to it.

AUTODEC

No options

PROGRAM EDITOR User interface -> Lists -> autoDec

SEE ALSO AUTOINC, LISTF, LISTM, AUTOENTER

AUTO ENTER LIST

This is used in connection with the **LISTM** (List Array) and **LISTF** (List File) commands either prior to the command being executed the first time or upon return from an ENTER UDF. This command will cause the program to automatically execute the ENTER UDF again without waiting for the user to press the ENTER key. Generally this command is used in conjunction with the **AUTOINC** or **AUTODEC** command.

AUTOENTER

No options

PROGRAM EDITOR fiLe -> Write

SEE ALSO AUTODEC, AUTOINC, LISTF, LISTM

AUTO INCREMENT LIST

This is used in connection with the **LISTM** (List Array) and **LISTF** (List File) commands. If you are executing one of those commands and are in a routine 'outside' of the command (e.g., an enter process), you can use this command to automatically force the cursor to go to the next line in the list when you return to it.

AUTOINC

No options

PROGRAM EDITOR User interface -> Lists -> aUtoinc

SEE ALSO AUTODEC, LISTF, LISTM, AUTOENTER

AUTO RUN (*DOS only*)

This command can be used to record characters from the keyboard or play back those recorded earlier.

AUTO_RUN *ARN field* **DELAY** *f/c/e RECORD*

ARN field - fn/v - Required - The field that contains the characters to be used as input. This field should contain the values in the .ARN file created either by the **RECORD** option or with the Convert .CHR to .ARN program (see **Chapter 3, Main Menu - Convert .CHR to .ARN**)

DELAY - f/c/e - Optional - If you are playing back characters this is the number of 64k loops you want the system to do between each character. The higher the number the slower the program will run. You would set this value only if you wanted the user to be able to watch the progress of the play back. Otherwise, do not set this value and the program will run as fast as possible.

RECORD - Optional - If you are recording characters entered from the keyboard then you would set this value.

COMMENTS

For an example of this process see AUTOTEST.SRC in the SAMPLE\ sub-directory. Also the support files, AUTOTEST.CHR and AUTOTEST.ARN

PROGRAM EDITOR System -> Auto_run

BACKGROUND (*DOS only*)

This command will change the screen output color to the value set as the **highlight** color in the **COLOR** command.

BKG

No options

PROGRAM EDITOR User interface -> Color Control -> Background

SEE ALSO COLOR, CC(), CCE(), CCF(), CCH(), CCR()

BELL

This command will cause the speaker to beep.

BELL

No options

PROGRAM EDITOR System -> Bell

BUTTON (*Windows only*)

Use this command to put buttons on the screen. Users will be able to click on these buttons with their mouse. Each time they click the button it will be just as though they had pressed the key that you setup as the related value.

BUTTON AT *col,row* **LEN** *length* **WDT** *width* **COLOR** *main_color* **TEXT_COLOR** *caption_color*
CAPTION *caption_text* **KEY** *related_key* **SAVE_TO** *save_to_field*
RSTR_FROM *restore_from_field* **USING** *using_holder_field OFF ON REMOVE*

col - f/c/e - Required - The column value for the upper left corner of the button. The first column on the screen (far left position) is number 1. This value is always based on the largest window, generally the window that is created when the program is run. Even if the active window is smaller the column and row values are for the base window. This value is required only when creating the button initially.

row - f/c/e - Required - The row value for the upper left corner of the button. The first row on the screen (top row) is number 1. This value is always based on the largest window, generally the window that is created when the program is run. Even if the active window is smaller the column and row values are for the base window. This value is required only when creating the button initially.

length - f/c/e - Required - The length or height of the button in rows. We recommend that a button be at least 2 rows in height. This is required only when creating the button initially.

width - f/c/e - Required - The width of the button in columns. This is required only when creating the button initially.

main_color - f/c/e - Optional - The background color of the button. If no value is entered here then the program will use the default color provided by Windows. For more information on Windows colors please see **Chapter 11 - Windows Programming**. This value can be specified only when creating the button initially.

caption_color - f/c/e - Optional - The button text color. If no value is entered here then the program will use the default color provided by Windows. For more information on Windows colors please see **Chapter 11 - Windows Programming**. This value can be specified only when creating the button initially.

caption - f/c/e - Optional - The caption or text that will appear on the face of the button. If you don't specify a value here the button will be blank. This value can be specified only when creating the button initially.

key - f/c/e - Optional - The keyboard key this button will emulate when it is clicked. This is required only when creating the button initially. If you don't specify a key value, or one doesn't match those available, the program will automatically treat the button as the ENTER key. The following are the acceptable values:

F1 through F10
SF1 through SF10 (Shift F1 etc.)
^F1 through ^F10 (Ctrl F1 etc.)
@F1 through @F10 (Alt F1 etc.)
^A through ^Z (Ctrl A etc.)
@A through @Z (Alt A etc.)
ESC, UP (up arrow), DOWN (down arrow), LTA (left arrow), RTA (right arrow), HOME, END, PGUP, ^PGUP, PGDN, ^PGDN, INSERT, DELETE, WDLT (ctrl left arrow), WDRT (ctrl right arrow), TAB and BCKTAB.

An example of this option would be: ... Key 'ESC' ...

save_to_field - fn/v - Optional - If you want to save the current button list so that you can temporarily create a different list you would save the current list pointer or handle to this field. The field must by of type R for this option. When this value is specified all other options, other than **ON** or **OFF** are ignored.

restore_from_field - fn/v - Optional - If you have saved a button list pointer or handle previously with the **SAVE_TO** option you would restore it with this option. When this value is specified all other options are ignored other than **ON** and **OFF**. The field for this option must be of type R.

using_holder_field - fn/v - Optional - If you have previously saved a button list with the SAVE_TO option and now want to turn those buttons on or off without disturbing the current button list you would use this option. This will allow you to manipulate more than one button list on the screen at a time. The only options available with this option are **ON** and **OFF**. If you want to remove the buttons permanently you should **RSTR_FROM** first and then **REMOVE**. The field in this option must be of type R.

OFF - Optional - Turn the current button list off. This means make them disappear from the screen. This will also work with the **SAVE_TO**, **RSTR_FROM** and **USING** options.

ON - Optional - Turn the current button list on. This means to make them appear on the screen after they have been turned off previously. This will also work with the **SAVE_TO**, **RSTR_FROM** and **USING** options.

REMOVE - Optional - Delete the current button list. This will also remove the buttons from the screen.

COMMENTS

Each time you create a new button it is automatically added to the current button list. Its position on the list doesn't have any effect on where it appears on the screen, it's just where it is internally in the program. You can keep multiple lists active at the same time by using the **SAVE_TO**, **RSTR_FROM** and **USING** options. Even though the button may not be part of the current active list, if it's on the screen and the user clicks it, it's just as though the user pressed the appropriate key. Another thing to remember is that buttons are always active. This means if the button is on the screen (visible) the user can always click on it, regardless of what else is active. Even if just a little bit of the button is showing under another window, or menu, the user can still click on that and it's just as though they pressed the corresponding key. Since this could have unexpected effects on your program you need to be sure that you've turned the buttons off before bringing up a list or menu, etc.

For example, let's say you have buttons on the screen but you want to display a list with different buttons. You also want to redisplay those original buttons when you are finished with the list. To accomplish this you would do something like the following:

```
Button save_to field off      ;this would save the current
                              ;button list and turn the
                              ;current buttons off
Button at x,y ....            ;create new button or buttons.
Saves                         ;save current screen before
                              ;displaying the window.
                              ;NOTE:  SAVES and REDSP have
                              ;no effect on buttons.
Window at x1,y1 ...           ;create the appropriate
                              ;window for the list
Listf ....                    ;do the listf
Redsp                         ;get rid of the window
Button rstr_from field on     ;now restore the original
                              ;buttons.
```

This is a very simple example but it gives you an idea of how the process would work.

You also don't have to worry about removing buttons created during a program. That will be done automatically for you.

PROGRAM EDITOR

Win Commands-> Button

SAMPLE PROGRAM

WINTEST

SEE ALSO

HOT_SPOT, PICTURE

CAPTION (*Windows only*)

This will put a text line in the caption bar of the window created for your application. After this command executes the text will stay the same until the next time the command is executed.

CAPTION caption_text

caption_text - f/c/e - Required - The text you want to appear in the caption bar at the top of the window.

PROGRAM EDITOR Win Commands -> Caption

CASE

This is a process control command, i.e., it will control whether a command, or group of commands will be executed. This is one part of a complete command structure.

CASE option_number

option_number - c - Required - Must be an integer value. If the value corresponds to the result in the **SELECT** command, the command lines between the **CASE** command and the next **CASE**, **OTHERWISE** or **ENDC** (Endcase) command will be executed.

COMMENTS

For more information on the different structured programming commands, and the **CASE** command in particular, please see **Chapter 7, Structured Programming Commands.**

If you are missing an **ENDC** (End Case) command in a **SELECT/CASE** structure you will get the If without Endif error message during the compile process.

PROGRAM EDITOR prg Control -> Case -> Case

SEE ALSO SELECT, OTHERWISE, ENDC

CHAIN

Use this command to execute another TAS Professional 5.1 program.

CHAIN *program_name* **WITH** *with NOBASEWIND*

program_name - f/c/e - Required - The name of the program to run. This needs to include the path also, if any.

with - fn/v1,fn/v2,...,fn/vx - Optional - A list of field values to be passed to the program. The receiving program can test for these with the **PARAM** (Parameters) command.

NOTE: Do not use constants or expressions in this command since TAS Pro 5.1 clears that information out before chaining to the next program. You can, however, pass pointers (P type only) to the receiving program so that you can access a value

directly in the previous program. Also, see the **RESET** option in the **DEFINE** command about another way of doing this.

Windows Note: In TAS Professional for Windows you actually can use constants or expressions in the CHAIN command. However, if you're writing for both DOS and Windows you should stick with the restriction above.

NOBASEWIND - ***Windows Only*** - Optional - When a TAS program is run in Windows TAS Professional for Windows creates a new base window the size of the base form. This covers all windows under it. If you want to display a smaller window in the chained to program then include this option. When the new program runs a base window will be automatically created the first time you use the **WINDOW** or **WINDEF** command. You should note that the base window will be only the size of the window you create so if you want to put buttons at the bottom of the base form you won't be able to since the base window for the new program will be smaller than the base form. Where this option is most helpful is when you are chaining to a program that has no screen effect. Each time you chain to that program it would look like you were clearing the screen when you aren't. By using this option that effect would disappear. For more information on creating windows in Windows please see **Chapter 11 - Windows Programming**.

COMMENTS

The original program, upon returning from the "chained-to" program, will continue execution at the next line. All data files previously opened will still be so, and all data will be available (except for that data appearing strictly in the chained-to program).

The only limit to the depth of chaining is available memory. Each program when executed uses only that memory which is absolutely required to run. This can be as little as 5 or 10k or as much as the remaining memory in the system. Each program, when executed, is absolutely independent from its "calling" program unless you desire to pass data or files. Each program can have the maximum number of fields, line labels, etc. However, the limitation on the maximum number of files that may be opened is system-wide, and does not depend on how many or how few programs are executed.

When the chained-to program returns to the original "calling" program, the memory used by the chained-to program is released.

PROGRAM EDITOR System -> Other programs -> Chain

SAMPLE PROGRAMS CHAINTST, CHNTST2, MCHNTST, MCHNTST1

SEE ALSO PARAM

CHAIN RUN ANYTIME PROGRAM

This program will **CHAIN** to a program already specified as a RAP program.

CHAINR *rap_number*

rap_number - f/c/e - Required - The **RAP** buffer number. This is the same value as the **NUMBER** option as set in the **RAP** (Run Anytime Program) command.

COMMENT

This command is similar to the regular **CHAIN** command in that when the **RAP** program finishes executing it will return to the next line in the program that executed the command. This is different than the normal return from a **RAP** program in that the program would normally return to the same line.

PROGRAM EDITOR System -> Other programs -> chain Rap

SEE ALSO RAP

CHANGE DICTIONARY PATH

This command will change the default dictionary path to a different location.

CDPATH *location*

Command parts:

location - f/c/e - Required - The new path for the dictionary. The value should be terminated with a backslash, e.g.:

C:\NEWPRGS\

COMMENTS

This command will NOT affect the dictionary used during compilation. That value defaults to the path specified in the TAS50.OVL file or to the value set in the command line.

PROGRAM EDITOR System -> Programming -> chG dict path

CLEAR FILE BUFFER

This command will clear the internal record buffer for a particular file. It can also be used to clear the record number holder.

CLR *filename/@file_number BUFF/REC*

filename/@number - file_expr - Required - The name or number of the file to be cleared.

BUFF/REC - Optional - What to clear. If the option is *REC* the program will clear just the record number holder, effectively setting the record internally as not active. If you saved the record in that situation, it would be treated as though it were a new record. If the option is *BUFF*, both the record number holder and the record buffer are cleared. The default value is *BUFF*.

COMMENTS

If a record is active in memory, there is data in both the buffer and record number holder. If this record is saved to disk it will update the record currently at that record number. If the record number holder is cleared, the program will save a new record. Through the use of this command, if you clear just the

record number holder, you can set up a process where some or all of the previous record can be used as default values for the new record and still save a new record when the **SAVE** command is executed.

NOTE: When a record buffer is cleared, either during this command or **OPEN, OPENV** (Open Variable), **SAVE** or **DELETE**, the buffer is set to binary 0s.

USER INTERFACE

The equivalent action (**CLR ... BUFF**) occurs if the user presses the **F3** key and no **TRAP**'s are set for that key. The program will clear all open files. If none are opened, nothing will happen. If a screen is mounted and there are fields from any of these files on that screen, the values will be cleared.

PROGRAM EDITOR fiLe -> Clear

CLEAR LINE

This command will clear a line on the screen. You can start at a certain location, clear the entire line or a certain number of characters, and control the color displayed.

CLRLNE AT *starting_column,row* **NCHR** *number_of_characters* **COLOR** *color*
NOCOLOR ABS

starting_column - f/c/e - Optional - The number of the column at which to start the display. The first column on the screen (far left position) is number 1. The default value, if none is provided, is the current column value as set by the last display or enter to the screen. This value can be easily determined through the use of the **COL()** function.

row - f/c/e - Optional - The row or line number to clear. The first row on the screen (top) is number 1. The default value, if none is provided, is the current row value as set by the last display or enter to the screen. This value can be easily determined through the use of the **ROW()** function.

number_of_characters - f/c/e - Optional - The number of characters to clear. If this option is not set, the command will clear the remaining line.

color - f/c/e - Optional - If you do not set the *NOCOLOR* option described below, you can specify the color value to be used during this operation. If this option is not set, the program will use the regular color as currently set.

NOCOLOR - Optional - If this option is set the command will use whatever color is presently at that position on the screen.

ABS - Optional - If this option is set the program will use absolute coordinates when determining where to clear the line. If it is not set and a window is active, the upper left hand corner of the window is column 1, row 1. By including this option you will achieve the same result as using the **FORCE** command except that you don't have to turn it off. It is automatically turned off after the command is finished.

PROGRAM EDITOR User interface -> Screen control -> clear Line

SAMPLE PROGRAM CLRLTEST

CLEAR PROGRAM ERROR

This command will clear the system program error number holder. This is set anytime the program encounters an error while executing a program or if you execute the **ERR** (Error) command.

CLRPE

No options

COMMENTS

You would use this command when making calls to a routine that may or may not fail. This is the only way to clear the error number before calling the routine again. Also, the program error holder is not reset when the program returns to the calling (previous) program. This allows you to check if an error occurred in the chained program. Use the **ERR()** function to check for an error value.

PROGRAM EDITOR User interface -> Messages -> Clear prg error

SEE ALSO ERR, ERR()

CLEAR SCREEN

You use this command to clear the currently active window (may be entire screen).

CLRSCR

No options

COMMENTS

You can change the color in the current window by changing the normal color value in the **COLOR** command and then performing **CLRSCR** (Clear Screen).

NOTE: If you have a screen mounted and then do a **CLRSCR**, the text and all the fields will be cleared. However, the next time you get a record or redisplay the entire screen, the fields that were part of that screen will be redisplayed. To remove them you must use the **CLRSF** (Clear Screen Fields) command.

PROGRAM EDITOR User interface -> Screen control -> Clear screen

CLEAR SCREEN FIELDS

This command will remove the active screen fields added either through the **MOUNT** or **SAY** command.

CLRSF

No options

COMMENTS

Each time a record is found or the screen is refreshed the program redisplays all fields (if any) in the screen field buffer. If you want to eliminate this redisplay, then you must use this command. Once the

screen field buffer has been cleared, the only way the fields can be restored is through the reuse of the **MOUNT** and/or **SAY** commands.

NOTE: The **SAVES** (Save Screen) / **REDSP** (Redisplay Screen) commands save the current screen field buffer and restore it. If you have used the **SAVES** command, and then **MOUNT** a screen, when you use the **REDSP** command those current screen fields will be automatically replaced with the previous set. For more information please refer to the **SAVES** or **REDSP** command.

PROGRAM EDITOR User interface -> Screen control -> clear scrn Flds

SEE ALSO MOUNT, SAY, SAVES, REDSP

CLIK_SRCH_LIMIT (*Windows only*)

This command will limit how far the field search process will go in your program before it stops.

CLIK_SRCH_LIMIT

No options

COMMENTS

For a detailed explanation of the field search process please see **Chapter 11 - Windows Programming**.

PROGRAM EDITOR Win Commands -> cLik Srch Limit

SEE ALSO UPAR

CLOCK

Use this command to display a real-time clock on the screen.

CLOCK *ON/OFF* **AT** *starting_column,row MIL*

ON/OFF - Optional - If you set this option to *ON* the clock will be displayed on the screen, *OFF* and it will be removed.

starting_column - f/c/e - Optional - The number of the column at which to start the display. The first column on the screen (far left position) is number 1. The default value, if none is provided, is column 40.

row - f/c/e - Optional - The row or line number to use in displaying the clock. The first row on the screen (top) is number 1. The default value, if none is provided, is row 1.

MIL - Optional - If this option is included, the clock will display in 24 hour time instead of am/pm.

COMMENTS

The clock should be turned off before executing non-TAS programs.

PROGRAM EDITOR System -> Date/time -> Clock

CLOSE FILE

This command is used to close both TAS and non-TAS files that have been previously opened using the **OPEN** or **OPENV** (Open Variable) command.

CLOSE *filename/@file_number DEL*

filename/@file_number - file_expr - Required - The name or number of the file to be closed.

DEL - Optional - If the file being closed is non-TAS, you can include this option and the program will delete the file from the disk.

COMMENTS

If you don't close the files before the program exits, it will be done automatically. Once a file is closed, its 'position' can be used again; i.e., if one is closed another can be opened.

PROGRAM EDITOR fiLe -> Open/close -> Close

SEE ALSO OPEN, OPENV

CLOSE NON-TAS FILE

This is a TAS Professional 3.0 command here for compatibility. The preferred method is to use the **OPENV/CLOSE** commands.

CLSO *delete_yn*

delete_yn - Optional - If you want to delete the file after closing it then put a Y here.

COMMENTS

This is the equivalent of the TAS Professional 3.0 command Close Non-TAS File. It is the close command that would be used for files opened with the **OPNO** command.

PROGRAM EDITOR 3.0 Commands -> A - Clse non-TAS

CLOSE PRINT TO DISK FILE

This command will close the file used for printer output during a specific program operation. This will allow you to print to the disk and then load that file during the same program.

CLSPF

No options

COMMENTS

Normally, the print to disk file is closed when you exit a program, or specify output to another file.

PROGRAM EDITOR Reports -> close ptD file

SEE ALSO PON

COLOR (*DOS only*)

Set the three color values for the program.

COLOR NORM *normal* **HIGH** *highlight* **REV** *reverse* **ERR** *error*
ENTER *enter_color* **ARRAY** *color_array_start*

normal - *f/c/e* - Optional - The normal or regular color. This is the color used as the normal display color. Unless otherwise overridden, all display to the screen is in this color.

highlight - *f/c/e* - Optional - This color is used for choice bars in menus.

reverse - *f/c/e* - Optional - Used for displaying fields that are a part of a mounted screen.

error - *f/c/e* - Optional - The color used when displaying an error or help message at the bottom of the screen.

enter_color - *f/c/e* - Optional - The color to use when the ENTER command is executed.

color_array_start - *fn/v* - Optional - If this field is included then the program will get the colors from an array of color values with this field being color 1. Normally a color is specified in a program as a numeric value from 1 through 255. By using this option the programmer can include an array of color values and the program will translate the color number to an array number, i.e., color 1 would be array element 1, color 10 array element 10, etc. This will allow the programmer to specify a 'color value bucket' and then by changing the values in those 'buckets' change the colors to be displayed. This is the process that is used in all TAS Professional 5.1 programs. The color information is kept in TASCOLOR.OVL. Since each user can have their own copy of the file, they can have their own colors without having to create separate programs for each.

NOTE: Even though the color fields are considered part of an array they do not necessarily have to be a true array. So long as all the fields are of the same type as the initial field, they can be individual fields and will still be treated as an array by this routine.

PROGRAM EDITOR User interface -> Color control -> Color

SEE ALSO BKG, FRG, CC(), CCE(), CCF(), CCH(), CCR()

COMMAND

This will define the beginning of a User Defined Command (UDC).

CMD *command_name command_parameters*

command_name - sac - Required - This is a special alpha constant. It is the name you give to this command. It may be up to 14 characters long. It must not be the same as any line label used anywhere else in the program.

command_parameters - *fn/v1,fn/v2,...,fn/vx* - Optional - The values being passed to the UDC. A maximum of 20 fields may be specified. The field names are separated with commas.

COMMENTS

You must include the compiler directive **#UDC** or **#UDX** before the first user defined command is referenced in the program.

PROGRAM EDITOR

System -> Programming -> udC

SAMPLE PROGRAM

UDCTEST

COMPANY CODE

This command will get or set the three letter extension on files to be used as company codes (e.g., for use with Business Tools' accounting products).

CO *co_code GET/SET*

co_code - *f/c/e* - Required - The new three letter extension. If this is a *GET* operation then this must be a fieldname or variable field (fn/v).

GET/SET - Optional - If *SET* then the program will set the company code default value to that in **co_code**. If *GET* then the program will get the current company code value and put it into the **co_code** field.

COMMENTS

Accounting software products from Business Tools can handle multiple companies, and use a three character extension on files to indicate different companies' data. For example, the master inventory file for company #57 might be BKICMSTR.B57.

Once you set this value it is used as the extension for all TAS files opened unless otherwise overridden.

You should have a set of records in FILELOC.B with the proper company code value for each company you have active. For more information about this topic, please refer to **Chapter 3, Main Menu - Utilities**.

PROGRAM EDITOR

System -> Company code

SEE ALSO

CO()

COMPILE PROGRAM (*DOS only*)

Use this command to compile a TAS Pro 5.1 source code file from within another TAS Pro 5.1 program.

COMPRG *source_code_file* **PTO** *messages_where* **ERR** *error_field* *DEBUG*
DICT *dictionary*

source_code_file - f/c/e - Required - The name of the source file to be compiled. If the file isn't in the current path, the proper path must be part of the file name. For example:

COMPRG '*D:\TAS50\CUSTCODE*'.....

where the program being compiled is *CUSTCODE.SRC*

NOTE: If you are compiling an EDT file you must include the extension or the program will think it is an SRC type source file. In the example above it would be:

COMPRG *'D:\TAS50\CUSTCODE.EDT'*

messages_where - pd - Optional - This will tell the compiler to print the results somewhere other than the screen. If you choose **D** (disk) the program will create a file with the same name as the program being compiled, using the same path, with an extension of .ERR. If the *INT* option is set the file name will be set to INTERNAL. If you choose **P** (printer) the information will be sent to the default printer as set in TAS50.OVL.

error_field - fn/v - Optional - The program will return the number of errors that occur during the compile process into this field. Must be of I type.

DEBUG - Optional - If this option is part of the command, the compiler will include information so that finished program can display the program name and actual line number during any error or help message.

dictionary - f/c/e - Optional - You may specify an alternate dictionary path here if desired. It will override all others, including that in the TAS50.OVL file.

COMMENTS

Each program must be compiled before it can be executed by the TAS Pro 5.1 runtime. By being able to compile a 'field' and return it to another 'field' you can create 'one time use' programs on the fly. Through the ability to read and write non-TAS files using the standard **OPENV** (Open Variable) command, etc., you can create programs and still save them to disk, if desired.

PROGRAM EDITOR

System -> Programming -> compile Prog

SAMPLE PROGRAMS

COMPTST, COMPTST2

COMPILER DIRECTIVE

This command will give the compiler instructions that are necessary only during compilation of a program.

#type_value

type_value - sac - Required - This is another form of the special alpha constant. It may or may not also have a numeric constant (**value**) as a part. The options are:

ADD_FLDS - Adds the number of field slots specified to the internal field list. To be used during operation to add fields on the fly.

ALL_LOC - Specify that all fields defined are *LOCAL*. Still requires the **PROC** and **ENDP** compiler directives before this directive takes effect. Once the **ENDP** directive is encountered the **ALL_LOC** is turned off and must be specified again if you wish to make another set of **DEFINE** fields *LOCAL* to a routine.

CHK_UP_VLD - Normally, the **VALID** option in the **ENTER** command is always checked unless the user presses the ESC or Up Arrow keys. If you want to check the valid when the Up Arrow key is pressed also then include this directive.

ENDP - Specifies the end of a Procedure that is started with the **PROC** directive.

EXT_FMT - In TAS Professional 5.1 all screen/report formats are normally included as part of the source code. However, if you wish to have them as separate files on the disk use this compiler directive. If this is set then the program expects all formats (no matter what type) to be separate for this program. You can also use the **EXTERN** option in the **MOUNT** command if you want to keep just a few formast separate. This would apply when you have a format you use in several different programs.

FMT - If you are using 'old type' screen/report formats then this option must be set. If the **PRO3** compiler directive is also set then the compiler expects the screen/report formats to be in TAS Professional 3.0 form. If **PRO3** is not set then you must use the TAS Professional 4.0 method.

INC - A named source file to be compiled and included in the run program.

LIB - Specify a library of commonly used routines. At the end of the compilation process if the compiler is still missing pieces it will look here. If an extension is not given for the library file name it is assumed to be '.LIB'. If no path is given it is assumed to be in the user's current directory. If the name is surrounded by angle brackets, e.g., <TASRTNS.LIB>, it is to be found in the TAS50 path. With TAS Professional 5.1, you also have the ability to use a SET TASLIB= to set a path in your environment (generally in your AUTOEXEC.BAT file). If you use a SET TASLIB= statement, the compiler will look for all libraries in the path given. This means you can have one set of libraries for all programs you're developing.

OLD_MATH - Use the TAS Pro 3.0 method of resolving expressions, left to right, regardless of operator precedence.

PRO3 - Specifies to both the compiler and the runtime that the program is a direct conversion from TAS Professional 3.0. A more complete listing of the effects of this directive is in **Chapter 4, Compiler Information**.

PROC - Alerts the compiler that all **DEFINE** fields from this point until the corresponding **ENDP** directive that have the *LOCAL* option set are available to this routine only. The routine is made up of the commands between the **PROC** and **ENDP** compiler directives. Without the **PROC** directive the *LOCAL* option will be ignored.

SFLDS - An additional number of screen fields slots to be added to each screen buffer so that you can use the **SAY** command to add screen fields on the fly.

TDATA - Amount of temporary data to allocate for this program in bytes.

XLATE - Open and use the TRANSLTE.B file so that if a command or option can't be found in the normal list it might be found here.

UDC - User defined commands are allowed.

UDF - User defined functions are allowed.

UDX - Both UDCs and UDFs are allowed.

XLATE - Turn on the command/option translation capability.

COMMENTS

For more information on Compiler Directives please refer to **Chapter 4, Compiler Info**.

PROGRAM EDITOR System -> Program -> comp Directive

CURSOR

Use this command to change the size of the cursor on the screen and to turn it on and off.

CURSOR *ON/OFF* **SIZE** *start_line,stop_line WAIT DFLT*

ON/OFF - Optional - If *ON* the cursor will be turned on, if it is off. This is the default value. If *OFF* it will be turned off or removed from the screen.

start_line - f/c/e - Optional - The beginning scan line for the cursor block. This may be from 0 through 7 for color monitors and 0 through 13 for monochrome.

stop_line - f/c/e - Optional - The ending scan line for the cursor block. This may be from 0 through 7 for color monitors and 0 through 13 for monochrome.

WAIT - ***Windows Only*** - Optional - Will change the standard arrow cursor to an hourglass or whatever is the equivalent in Windows 95.

DFLT - ***Windows Only*** - Options - Will change the cursor back to the standard arrow. Used after setting the **WAIT** option when whatever process that was taking so long is over.

COMMENTS

The **stop_line** value should be the larger number. The default value in TAS Pro 5.1 is 6,7 for color and 11,12 for mono displays.

NOTE: The **ON/OFF** and **SIZE** options have no effect in Windows.

PROGRAM EDITOR User interface -> Screen control -> cursOr

DATE

This command will get or set the computer system date.

DATE *date_field GET/SET*

date_field - *fn/v* - Required - The date field that either contains the new date value or will be the holder.

GET/SET - Optional - If *SET* then the program will set the system date to that value in **date_field**. If *GET* then the program will get the system date value and put it into the **date_field** field.

COMMENTS

This command will affect just the workstation it is running on and no others if you are running on a network. This has the same effect on the computer as if you typed in DATE at the DOS prompt and entered the value.

PROGRAM EDITOR System -> Date/time -> Date

SEE ALSO DATE(), DTOC(), CTOD(), DMY(), MDY(), DOM(), YEAR(), CMNTH(), MNTH(), CDOW()

DEALLOCATE FIELD

This command will deallocate (remove from memory and allow reuse of previously allocated space) a field previously allocated using any of the allocation commands or functions.

DEALOC *fieldname*

fieldname - fn/v - Required - The name of the field being deallocated. The field must have been allocated using one of the appropriate commands or functions.

COMMENTS

Once this command has been used the field being deallocated will not be able to be used in a program until it is allocated again.

PROGRAM EDITOR Field -> Create/chg -> dEallocate

SEE ALSO ALLOC, ALOCARY()

DECREMENT

Decrement an I or R type field.

DEC *fieldname*

fieldname - fn/v - Required - The name of the field being decremented. Must be of I or R type.

COMMENTS

This is the same as subtracting 1 from the field. For example:

DEC *CNTR* is equivalent to:

CNTR = CNTR - 1

This command provides a more succinct method to subtract 1 from a field. This command also runs a little faster than the standard subtraction routine.

PROGRAM EDITOR

Field -> Decrement

SEE ALSO

INC

DEFINE FIELD

This command is used to 'create' a field that will be used only within the current program (or subsequent programs that are called from this program). This is not a field that is part of a record in a standard TAS Professional 5.1 file.

DEFINE *field_name_list* **TYPE** *type* **SIZE** *display_size* **DEC** *decimal_chrs*
ARRAY *num_array_elements* **PICT** *picture* **DUP** *dup_field UP RESET LOCAL* **INIT** *init_val*

field_name_list - sac1,sac2,...sacx - Required - The name or names of the field(s) being defined. Must conform to standard field name requirements. You may define a maximum of 10 fields in one **DEFINE** command. Each of the fields will have the same specifications.

type - sac - Optional - The type of the field being added. Must conform to standard field type requirements. If you don't specify a field type, the default value is N. The field name types are:

	Name	Internal size
A	Alphanumeric	maximum 4 gbytes
N	Numeric	8 bytes
D	Date	4 bytes
T	Time	4 bytes
I	Integer	2 bytes
B	Byte	1 byte
F	F type pointer	5 bytes
P	P type pointer	14 bytes
R	Record number	4 bytes

L	Logical	1 byte
O	Pro 3.0 BCD	maximum 10 bytes
V	Overlay field	

display_size - numeric constant - Optional - The display size of the field being added. Must conform to standard field display size requirements. If this is not included, then the default size value for the type of field is used.

NOTE: If the field is type A or O you must supply a value. V (overlay) type fields are treated internally as A type fields so you must supply a size for these also. Internally, the overlay field will start with the next field in the list and continue for the number of characters specified. See further information about the overlay field in COMMENTS below.

decimal_chrs - numeric constant - Optional - If the field is of type N you can specify the number of decimal characters. If this isn't included the default value is 0. This could affect the displayed value of the field or the ability to **ENTER** the correct value. Therefore you should take great care to make sure the proper value is used. The available range is 0 through 8.

num_array_elements - numeric constant - Optional - The number of array elements for this field. Default value is 0.

picture - f/c/e - Optional - If this is a type 'A' field, you can use this option to enter a field into a space that is shorter than the defined length of the field. This is called a slider field. A slider is a field that shows fewer characters on the screen than actually exist in the field. The user can see the other characters by pressing the LEFT and RIGHT ARROW keys and the characters will appear to 'slide' back and forth across the available space. This is very useful when you have several large fields and want them all to fit on the same screen. A slider field is specified with an alpha constant of "Sxx", where xx is the number of columns to display.

dup_field - fn/v - Optional - This option will force all of the fields in this particular DEFINE command to be of the same type, size, decimal characters, and array elements as the one specified from the File Field Data Dictionary. The benefit to this option is that if you are using defined fields in a program and those fields are supposed to mimic fields in a record, you no longer have to worry about making them match. Simply recompile the programs if you make any changes, and the defined fields will always match their DUP fields in the data dictionary.

UP - Optional - If this is a type A field you may specify whether all characters entered to it are to be automatically forced into upper case. To accomplish this include this option.

RESET - Optional - If this option is part of this command the program will try to match the fieldname up with the identical name in the previous program, if any. If it matches it will be reset to the specifications for the field in the previous program. This means that you can use that data in the new program just as though it was there normally. When you're done, and you return to the previous program any changes made to that data are still there. This is an alternative to using the **WITH** option in the **CHAIN** command and the **PARAM** command in the subsequent program. If the field cannot be found in the previous program, or there is no previous program, the field is treated like a normal field within that program. It is initialized and can be used, if desired.

This feature allows you to base certain options in subsequent programs on whether or not the data is there to use it and still not have to worry about what it's going to do if you come at the program a different way. This option is used in all TAS Pro 5.1 programs to pass color values from program to program so that the TASCOLOR.OVL file needs to be read only once.

If you know you're not going to use a field unless it is 'reset' to a field in the previous program you can define it as any type and size you want. For example, if you want to access a very large array in a previous program and won't use the field at all if it isn't there you should define the field as an A type size 1. If it is blank (or SIZE(fld_name,'d')=1) you will know that you didn't reach this program from a previous one, or at least from the one that was passing the data. Of course, there are other ways to find that out, including passing info about the previous field with the **PARAM** command. Using this approach, you can avoid taking data space in the current program that will go unused since you are accessing data in the previous program instead. It doesn't matter what type or size you specify for the subsequent field. It will always be changed to the specs of the original field if the **RESET** option is active.

LOCAL - Optional - If this option is included in the command then the program adds the procedure number to the field information. Through the use of this option and the **PROC/ENDP** compiler directives you can have multiple routines with the same field names and not worry about whether or not you overwriting some other data belonging to another field. The scope of a *LOCAL* field is limited to the boundaries of the routine in which it appears.

The key to this whole process is to make sure the routine this field applies to has the **PROC** directive at the beginning (before the field is defined) and the **ENDP** directive at the very end (after the last **RET** command that applies to this routine).

NOTE: If you want to have the same effect as *LOCAL* but don't wish to use this process, or you need to protect a file field (can't be defined as *LOCAL*), use the **PUSHF** (Push Field) and **POPF** (Pop Field) commands.

init_val - c1,c2,c3,. . ., cx - Optional - This option will initialize the defined field to these values. Specify on the command line what values you want the defined field to be set to when the program is run. The values must be constants; other fields and expressions are not allowed. You can initialize A, N, B, I, R, D, and T type fields. Please note that O (BCD), P, and F type fields cannot be initialized using this option. IF YOU USE THIS OPTION, IT SHOULD BE THE LAST THING ON THE COMMAND LINE. The option must appear after the type, size, and number of array elements have been specified. If you are defining an array field, you may initialize each element separately. Do this by using a list of constants, with each constant being the correct type. The program will initialize the array field starting with element 1 and going until it runs out of values. If you put more values on the line than array elements, you will get an error during compilation. These values are actually set at the time the program is started, so you can put your **DEFINE**s anywhere in your program, and the values will not be reset just by running over the original **DEFINE** command.

NOTE: 'A' type constants should be specified without surrounding quotes. 'D' and 'T' type constants do need quotes. For example:

```
;The following shows a non-array A type field with an INIT value
DEFINE TEST type A size 10 init ABCDEFG
```

```
; This is a D type array field with an INIT value
DEFINE DATE_FLD type D size 8 array 2 init '01/01/94' , '02/01/94'
```

No other field types need surrounding quotes.

COMMENTS

Generally, if multiple fields are defined using the same command the fields will follow sequentially in memory. However, this is not always the case. When the field is encountered in the program the first time it is put in the program's field list. If you are using defined field overlays and you want to make sure that all the fields in the block are together you should define them at the beginning of the program before any reference is made to any of the fields in the block.

If a field is not a part of the regular file dictionary or the define field dictionary it must be explicitly defined using this command somewhere in the program.

PROGRAM EDITOR

Field -> Create/chg -> Define

SEE ALSO

REDEF

DELETE ALL RECORDS

You can use this command to delete all or a group of records from a TAS Professional 5.1 file.

DALL *filename/@file_number* **KEY** *keyname/@key_number* **START** *start_value* **SCOPE** *scope*
scope_value **FOR** *for_filter_expression*
WHILE *while_filter_expression* **CNTR** *counter_field DISP*

filename/@file_number - file_expr - Optional - The name or number of the file to be used. If you do not include this option, the program will look for a default search file set in the **SRCH** (Search File) command. If that hasn't been previously set, the program will report an error and the command will be skipped. This entry will override any default value set in the **SRCH** command.

keyname/@key_number - key_expr - Optional - Set this to the appropriate value to delete the records in the file in the correct order. If you do not include this option, the program will look for a default key set in the **SRCH** command. If a default key hasn't been previously set the program will report an error and the command will be skipped. This entry will override any default value set in the **SRCH** command.

start_value - f/c/e1,f/c/e2,...,f/c/ex - Optional - The value to use as the beginning record. This is similar to doing a **FIND** before the command. If the index you're searching on has multiple segments you may list the proper value for each segment, separating them with commas. This will allow you to specify the exact type for each of the segments. For example, suppose you're searching on an index that has two segments: a 5 character alpha and an I type field. Each record you want has the same alpha but the I field will change, and is in order. In the first record the I type field has a value of 0. The following would find the first record for that group, if it exists:

```
start 'ABCDE',0!
```

Notice that we separate the values with a comma. The only requirement is that the values be of the same type (and size) as the segments in the key. In this case we put the I type constant specifier (!) after the 0 to make sure that it is passed as an I type field.

scope - Optional - This option may help determine how many records are deleted. For more information about the **scope** specifier please see the general information at the beginning of this chapter.

scope_value - f/c/e - Optional - If **scope** is set to **N** or **F** this is the number of records to delete.

for_filter_expression - lexpr - Optional - You would use this option to restrict the records deleted in this command. If the expression did not resolve to .T. the record would not be deleted. In this option, the search continues until the program reaches the end of file and then will quit.

while_filter_expression - lexpr - Optional - This option also restricts the records deleted. However, the first time the expression resolves to .F., the program stops deleting records and continues to the next command. Here's the benefit of this option: if the program is deleting records in a certain order (by a specific key) and the expression returns .F. then the program knows that all the appropriate records have been deleted and there is no need to continue. For example: You want to delete all the invoice records for a specific customer. The first record for that customer is found in the invoice file either by using the **start_value** option within this command, or through the **FIND** command before executing this command. Then the **while_filter_expression** would be:

INV_CUST_CODE = CUSTOMER_CODE

(This assumes that there is a field called INV_CUST_CODE in the invoice file and a like field in the customer file called CUSTOMER_CODE. The **keyname/ @key_number** option must be set to the proper value so that the records are deleted in CUSTOMER_CODE order, or you must have set the **SRCH** command previous to this command.) When the first invoice record for the next customer is read the program will stop deleting records.

NOTE: If there is no **start_value** you must set the **scope** value to *R* or *N xxx*. If you do not do this, the program will find the first record in the file and the **while_filter_expression** option will probably fail the first time. However, if the **start_value** option is used you can ignore this requirement.

counter_field - fn/v - Optional - This is an I type field that will be used for passing the number of records deleted.

DISP - Optional - Each time a record is read the program will redisplay the screen fields, if you specify this option. This will slow down the operation of this command, depending on how many fields are displayed on the screen. The net effect will probably be negligible unless you are reading through a very large file.

COMMENTS

If the **while_filter_expression** option is used you must watch out for the proper setting of the **scope**, and/or **start_value** options also. If everything is supposedly set properly and yet no records are being deleted, make sure that the proper first record is in memory prior to the execution of this command or that correct use has been made of the **start_value** option. If a record is not active when this command is executed nothing will be deleted. You can use the **start_value** option to make sure that a record is active.

NOTE: This command will work on Btrieve (TAS Pro 5.1 as well as others) files only.

PROGRAM EDITOR fiLe -> Mult rec cmds -> Delete all

SEE ALSO DEL

DELETE CHARACTERS

Use this command to delete a group of characters from an alpha field.

DELC *fieldname* **START** *start* **NCHR** *number_of_characters* **MEM_AREA** *mem_area#*

fieldname - fn/v - Required - The name of the alpha field.

start - f/c/e - Optional - The starting position within the field. The first character in the field is at position 1. If this value isn't specified the program will set **start** to the first character.

number_of_characters - f/c/e - Required - The number of characters to delete from the field.

mem_area# - f/c/e - Optional - If you are actually deleting the characters from a memory area then this is the number (from 1-4). NOTE: Even if you want to use a memory area you must still specify a **fieldname**, although it will be ignored during execution of the command.

COMMENTS

This command has the same effect as though you cut out a piece of the field and put the remaining two pieces back together, filling the end of the field with spaces.

PROGRAM EDITOR Field -> alpha Fld cmds -> Delete chrs

DELETE FILE

This command will delete a file from the disk.

DELF *filename*

filename - f/c/e - Required - The name of the file to be deleted.

COMMENT

No error is reported if the file is not found and nothing is returned if it is.

PROGRAM EDITOR System -> File commands -> Delete file

DELETE RECORD

Delete an active record for a specific TAS Pro 5.1 file.

DEL *filename/@file_number NOCNF* **GOTO** *goto_if_no_del* **ERR** *error_label*

filename/@file_number - f/c/e - Required - The name or number of the file that contains the record to be deleted.

NOCNF - Optional - If this option is included in the command, the record is deleted without asking for confirmation from the user. The default action is to ask before deleting the record. This would be useful when you want to ask more than just "Delete? Y or N," or if you're deleting records from multiple files and want to ask the Delete question only once.

goto_if_no_del - label - Optional - If the user answers **N** to the Delete question (*NOCNF* is not included), you can put a line label here signifying where the program should transfer control. If this is not provided the program will continue with the next line.

error_label - label - Optional - If you specify a label here the program will transfer control to the appropriate line if an error occurs during the command. If you don't put a label here and an error occurs it will be displayed on the screen for the user. If you don't want an error to be displayed then specify NO_ERR as the **error_label**.

NOTE: If an error occurs during the DEL command you can test for it with the **FLERR()** function.

COMMENTS

This command will only work with regular TAS Professional 5.1 files. Also, the record to be deleted must be active. This means that it cannot have been cleared either through the **CLR** (Clear Record Buffer) command or **SAVE** command.

PROGRAM EDITOR

fiLe -> Delete

SEE ALSO

DALL

DISPLAY ARRAY FIELDS

This command will display an array on the screen without having to display each element individually. Each element will display as a separate line. Through the use of this command and by capturing key strokes using the **INKEY()** function you can set up a routine where an array of alpha fields too wide to be printed to the screen (a 132 column report printed to disk perhaps) can be 'shifted' left, right, up, or down, depending on the user's input.

DISPF *field_name* **OFST** *starting_character_position* **AT** *starting_column,starting_row* **WDT** *number_of_characters_to_display* **NUM** *number_of_lines_to_display* **COLOR** *color_values*

field_name - fn/v - Required - The name of the array field. This must be an 'A' type field. If you want to start with an element other than the first, the array element number can be included in the field spec. For example:

fld[element_num]

By incrementing or decrementing the element number you can make the information appear to scroll down or up the screen.

starting_character_position - f/c/e - Optional - This is the position number of the first character to be displayed. The first character in the array element is at position 1. Even though this is an optional field, you increment or decrement this value to move right or left across the screen. The default value is the first character in the field, or 1.

starting_column - f/c/e - Optional - The number of the column at which to start the display. The first column on the screen (far left position) is number 1. The default value, if none is provided, is the current column value as set by the last display or enter to the screen. This value can be easily determined through the use of the **COL()** function.

starting_row - f/c/e - Optional - The number of the row at which to start the display. The first row on the screen (top) is number 1. The default value, if none is provided, is the current row value as set by the last display or enter to the screen. This value can be easily determined through the use of the **ROW()** function.

number_of_characters_to_display - f/c/e - Optional - The number of characters to display on a single line, from a single array element. If this is not included the program will use the display size of the field or the width of the current screen window, whichever is less.

number_of_lines_to_display - f/c/e - Optional - The number of lines (array elements) to display at one time. If this is not included the program will use the maximum number of array elements in the field or the number of rows in the current screen window, whichever is less.

color_values - NY - Optional - If **Y** then this array is actually color values and should be displayed appropriately.

NOTE: All commands that specify a column/row value do so in the form **AT** *col,row* where the column and row values are separated with a comma.

COMMENTS

The array being displayed must be of type A. By manipulating the array element value within the array field spec and the **starting_character_position** value, you can give the impression of a window being moved across a much larger data block.

PROGRAM EDITOR

Field -> Array -> Disp Array

SAMPLE PROGRAM

DISPARY

SEE ALSO

RDA

DISPLAY MEMORY AREA (*DOS only*)

This is a TAS Professional 3.0 command here for compatibility. There is no preferred TAS Professional 5.1 command/function.

DISPM *mem_area#* **START** *start_val* **OCCURS** *size_of_element* **OFST** *offset_in_element* **AT** *col_row_loc* **WDT** *#_chrs_to_prt* **NLNES** *#_lines_to_prt*

mem_area# - f/c/e - Required - This is the memory area to display. This must resolve to a value of 1 through 4.

start_val - f/c/e - Required - The starting character in the memory area. The first character is at position 1.

size_of_element - f/c/e - Required - This is the total width in characters of a single element. This command will be displaying a part or all of a single element.

offset_in_element - f/c/e - Required - Where do the characters to be displayed begin within the element. The first character position in each element is 1.

col_row_loc - f/c/e,f/c/e - Required - The beginning column/row location separated with a comma. For example, if the beginning location is column 10, row 10 then this would be:

AT 10,10

#_chrs_to_prt - f/c/e - Required - How many characters do you want to print from the element.

#_lines_to_print - f/c/e - Required - How many lines (elements) do you want to print. The program will start with the position specified in START and print the number of lines specified here.

COMMENTS

This is the equivalent of the TAS Professional 3.0 command Display Memory.

PROGRAM EDITOR

3.0 Commands -> Disp mem

ELSE

This is a process control command, i.e., it will control whether a command, or group of commands will be executed. This is one part of a complete command structure.

ELSE

No options

COMMENTS

For more information on the different structured programming commands, and the **ELSE** command in particular, please see **Chapter 7, Structured Programming Commands.**

PROGRAM EDITOR

Prg Control -> If -> Else

SEE ALSO

IF, ELSE_IF, ENDIF

ELSE_IF

This is a process control command, i.e., it will control whether a command, or group of commands will be executed. This is one part of a complete command structure.

ELSE_IF *else_if_expression*

else_if_expression - lexpr - Required - If the **else_if_expression** resolves to .T., the command lines between the **ELSE_IF** command and the next **ELSE_IF**, **ELSE** or **ENDIF** command will be executed.

COMMENTS

For more information on the different structured programming commands, and the **ELSE_IF** command in particular, please see **Chapter 7, Structured Programming Commands.**

PROGRAM EDITOR

Prg control -> If -> eLse_if

SEE ALSO

IF, ELSE, ENDIF

ENDCASE

This is the end point of the process control command **SELECT/CASE/ENDC**.

ENDC

No options

COMMENTS

For more information on the different structured programming commands, and the **ENDC** command in particular, please see **Chapter 7, Structured Programming Commands.**

If you are missing an **ENDC** (End Case) command in a **SELECT/CASE** structure you will get the If without Endif error message during the compile process.

PROGRAM EDITOR

Prg control -> Case -> Case

SEE ALSO

SELECT, ENDC, OTHERWISE

ENDIF

This is the end point of the process control command **IF/ENDIF**.

ENDIF

No options

COMMENTS

For more information on the different structured programming commands, and the **ENDIF** command in particular, please see **Chapter 7, Structured Programming Commands.**

PROGRAM EDITOR Prg control -> If -> eNdif

SEE ALSO IF, ELSE, ELSE_IF

ENDSCAN

This is the end point of the process control command **SCAN/ENDS**.

ENDS

No options

COMMENTS

For more information on the different structured programming commands, and the **ENDS** command in particular, please see **Chapter 7, Structured Programming Commands.**

PROGRAM EDITOR fiLe -> Mult rec cmds -> Scan -> Ends

SEE ALSO SCAN, SLOOP, SLOOP_IF, SEXIT, SEXIT_IF

ENDWHILE

This is the end point of the process control structure **WHILE/ENDW**.

ENDW

No options

COMMENTS

For more information on the different structured programming commands, and the **ENDW** command in particular, please see **Chapter 7, Structured Programming Commands.**

PROGRAM EDITOR Prg control -> While -> eNdw

SEE ALSO WHILE, LOOP, LOOP_IF, EXIT, EXIT_IF

ENTER

Use this command to allow the user to enter data into a field.

ENTER *fieldname* **MASK** *mask* **HELP** *help_label/@expression* **AT** *starting_column,row* **COLOR** *color* **PRE** *pre_enter_expression* **POST** *post_enter_expression*
DFLT *default_value* **VLD** *valid_check* **VLDM** *valid_message*
ARRAY **CNTR** *array_counter* **ENUM** *enumerated_field* **DO** *do_udf*
UPAR *up_arrow_goto_label* **GROUP** *group_num*
UP ACR PSWD NOREV AUTO_SRCH NOCLICKON NOCLICKOFF

fieldname - fn/v - Required - The name of the field being entered. This field must have been defined within the program or be part of a file that has been opened. If entering an A type field the maximum number of characters that may be entered is 256. If you have use of the TASEdit (tm) program you should use that program for fields larger than the maximum size.

mask - f/c/e - Optional - In the case of an A type field this would be the allowable entry characters. If this option is set and the user enters a character not included in this field, the program will sound the bell and not allow the character to be entered.

This is not the same as the **PICTURE**. The **PICTURE** allows more control and can be defined for the field within the **DEFINE** command or data dictionary.

help_label/@expression - label or f/c/e - Optional - You may provide a specific help message for this **ENTER** command. It may be of two different types. You may specify a label name. In that case the program passes control to that label and when the program returns, it continues with the ENTER command. You may also specify an expression of some type preceded by the at sign (@). This might be as simple as an alpha constant. For example:

```
@'This field is the customer name.'
```

Or it could be a UDF that passes the number of the record in ERRMSG.B. For example:

```
@help(1009)
```

starting_column - f/c/e - Optional - The number of the column at which to start the entry. The first column on the screen (far left position) is number 1. If the field is not part of a mounted screen format you must supply the column and row where entry is to begin. If you don't provide a column and row, an error message will be displayed and the program will continue with the next command.

row - f/c/e - Optional - The row or line number to use in the entry. The first row on the screen (top) is number 1. If the field is not part of a mounted screen format you must supply the column and row where entry is to begin. If you don't provide a column and row, an error message will be displayed and the program will continue with the next command.

color - f/c/e - Optional - This option can set the color to be used when entering this field. The color will revert back to the display color after the **ENTER** command is completed. The **color** value must resolve to a standard color value.

pre_enter_expression - lexpr - Optional - This is an option that you can use to make sure that all requirements are met before allowing entry to this field. If the expression returns .F., the program will not allow the entry.

NOTE: A very important feature is the ability to use a UDF as the expression. This means you can transfer control within the **ENTER** command to another routine that can be used, not only for making sure you really want to enter this field at this time, but also to set up all appropriate traps and to display the correct information at the bottom of the screen. By using the post_enter_expression to undo what you have done with this option, you can be sure you will never have traps set that shouldn't be or messages that don't relate to what is really going on.

post_enter_expression - lexpr - Optional - If you want to do something after the entry, it can be done here. You can use this to reset traps and other housekeeping chores that need to be done before the user will be able to press another key, or you can also test against whatever criteria you desire. The result of this expression can be checked with the **ENTER()** function. The function will return whatever this expression returned (.T. or .F.).

default_value - f/c/e - Optional - If you want the field to default to a certain value upon entry the value can be set here. If the **ENTER** field is blank (in the case of an A type field), or 0 the **default_value** will be displayed as the **ENTER** field value. The user will be able to clear this data by pressing CTRL-U or by overwriting the value; by pressing the ENTER or RETURN key, the user can accept the value.

valid_check - lexpr - Optional - Through the use of this option you can create an expression that will check if the value the user entered was acceptable. The test can be any function that will return .T. or .F. If the expression returns .F., the program will look for a **valid_message**. If it exists it will be displayed; if not, a standard valid error message will be used. The program will then continue with the entry and will not allow the user to exit without the **valid_check** option returning .T. There is a case where this isn't true and that's when the user presses the ESC key. In this case the program will always exit the **ENTER** command, unless the **ESC TRAP** is set to **ignore**.

A valid_check may be as simple as:

```
NUM_UNITS>0
```

Or it can be a UDF or a complex expression. For example:

```
ROW()>3 .a. (TOTAL_SALES<1000 .o. MONTH(DATE())=3)
```

NOTE: Do not do another ENTER as part of the **valid_check**.

valid_message - f/c/e - Optional - Use this option if the **valid_check** option is used and you want to send a special message if the expression returns .F. The message will be displayed in the standard error message location. You may include the valid_message as an alpha constant with no other requirements, if desired. For example:

"You didn't put in the proper value. Please try again."

ARRAY - Optional - If this option is included in the command then the program will allow you to enter all of the elements in that array field through this one ENTER statement. The user can 'page' through the elements by pressing the Page Dn key for elements beyond the current value and Page Up for previous elements. Changes can be made and

saved. If you use the **array_counter** value you can display the current element number as the user 'pages' through the different values.

NOTE: Do NOT include an array specifier as part of the **fieldname**. If you wish to choose the array element directly use the **array_counter** option and put the appropriate element number in that field.

array_counter - fn/v - Optional - If you have specified that this is an **array_field** you can also specify an **array_counter** field. The program will pass the current element number to this field. If you put this field on the screen, everytime the user presses the PgDn or PgUp key the appropriate element number will be displayed.

You can see this process at work in the Maintain Database program. It puts the field in the proper location within the field name so that it looks like an array specifier. You can also change the **array_counter** directly and control which element is displayed in cases where there are so many array elements that using the standard PgUp or PgDn key movement is too time-consuming.

enumerated_field - f/c/e1, f/c/e2,..., f/c/ex - Optional - You can specify an enumerated list of values that are the only acceptable values for this field. The user can page through the list by using the PgUp/PgDn keys or by pressing the space bar. The first value in the **enumerated_field** list will be displayed as the default field value. When the user presses the RETURN or ENTER key the value currently displayed will be stored in the **ENTER** field.

do_udf - udf - Optional - In this case you don't want to actually enter the field but want to execute this UDF. In general, this would be to edit a large field using the TASEdit™ program, or the **MENU** command, etc. However, you can use this for any purpose. If a value is returned from the UDF, it will be saved into the field. If no value is returned the program assumes that you made any changes desired directly into the appropriate field.

The user can initiate this UDF by pressing the F2 key or Ctrl-Home. You should exit your routine with the standard **RET** command.

up_arrow_goto_label - *label* - Optional - Generally, if the user presses the UP ARROW key, the program will leave the current **ENTER** field and move towards the beginning of the program command line by command line until the next previous **ENTER** or **UPAR** (UP ARROW) command is found. If none is found the bell will sound and the current **ENTER** will be re-executed. If you want to send the program to a specific line label and not allow this 'search' then this option may be used.

In general it is much wiser to use the **UPAR** command and the **pre_enter_expression** option to control the process by which the user moves around on the screen.

group_num - ***Windows Only*** - f/c/e - Optional - When a user selects a field on the screen by clicking on it with the mouse the first thing the field select process does is check the **GROUP** value for the current field. If there is no value set by the programmer the internal value is set to 0. The process will then allow the user to only select fields with the same **GROUP** value. By setting all fields to different **GROUP** values you can easily keep the user from selecting any fields with the mouse.

UP - Optional - If this option is included in the command the program will force to upper case all alphanumeric characters entered. Even if this option isn't set in the **ENTER** command, it will apply if it is set in the field definition.

ACR - Optional - If this option is included in the command the **ENTER** routine will automatically exit upon filling the last character in the field. Normally, the user would have to press the ENTER or RETURN key to finish the **ENTER** command.

PSWD - Optional - If this option is included in the command the characters the user enters are not echoed to the screen. The program will display question marks (?) instead of the entered characters.

NOREV - Optional - Normally the program uses the **reverse** color value when displaying and entering fields. If this option is included in the command, the **ENTER** command will use the **normal** color value.

AUTO_SRCH - Optional - If this option is included in the command and the field is listed as an index in the Data Dictionary, then upon exiting the **ENTER** command the program will attempt to find a matching record. If a record is already active for this file, this option will be ignored.

NOCLICKON - ***Windows Only*** - Optional - If you set this option the user will not be able to select this field with the mouse.

NOCLICKOFF - ***Windows Only*** - Optional - If you set this option the user will not be able to select a different field with the mouse when they are currently on this field.

COMMENTS

In all situations if the user presses a control key or function key and a trap is set for that key this command will act as though s/he pressed the ENTER key first, and the entry will be saved before exiting the command. However, if the ESC key is pressed the data entered is discarded.

PROGRAM EDITOR

User interface -> Enter

SAMPLE PROGRAMS

ENTTEST, MTEST

SEE ALSO

NOVLDMSG, ENTER()

EQUAL (=)

Use this command to set the receiving field to the value of an expression.

receiving_field = expression

receiving_field - fn/v - Required - The field that is going to be changed.

expression - f/c/e - Required - The expression that is providing the data. This can be as simple as a numeric constant, 1, or a very complex expression that uses functions and UDFs.

COMMENTS

If the two field types are not the same the program will attempt to convert the expression to the type of the **receiving_field**.

PROGRAM EDITOR

Field -> Equal

EQUALS DAY

This is a TAS Professional 3.0 command here for compatibility. The preferred method is to use the =DOW() function for numeric day of week, or =CDOW() for alphanumeric day of week.

EQU_DAY *recv_field* **FLD** *date_field*

recv_field - fn/v - Required - The field that is going to receive the day value. If the receiving field is A type then the result is the character day of week. If the receiving field is numeric the result is the day ofweek number.

date_fld - fn/v - Required - The field containing the date value to use for calculating the day.

COMMENTS

This is the equivalent of the TAS Professional 3.0 command Equals Day.

PROGRAM EDITOR 3.0 Commands -> Equ Day

EQUALS MONTH

This is a TAS Professional 3.0 command here for compatibility. The preferred method is to use the =MNTH() function for the month number, or =CMNTH() for alphanumeric month.

EQU_XMT *recv_field* **FLD** *date_field*

recv_field - fn/v - Required - The field that is going to receive the month value. If the receiving field is A type then the result is the character month. If the receiving field is numeric the result is the month number (1-12).

date_fld - fn/v - Required - The field containing the date value to use for calculating the month.

COMMENTS

This is the equivalent of the TAS Professional 3.0 command Equals Month.

PROGRAM EDITOR 3.0 Commands -> Equ Mnth

EQUALS PORTION OF

This is a TAS Professional 3.0 command here for compatibility. The preferred method is to use the =MID() function.

EQU_MID *recv_field* **FLD** *parse_fld* **START** *start_val* **NCHR** *num_chrs* **MEM_AREA** *memory_area#*

recv_field - fn/v - Required - The field that is going to receive the portion of the **parse_fld**.

parse_fld - fn/v - Required - The field that you want a part of.

start_val - f/c/e - Required - The starting character in the parse_fld. The first character in the field is at position 1.

num_chrs - f/c/e - Required - The number of characters to get from the **parse_fld**.

mem_area# - f/c/e - Optional - If you are actually getting the characters from a memory area then this is the number (from 1-4). NOTE: Even if you want to use a memory area you must still specify a **parse_fld**, although it will be ignored during execution of the command.

COMMENTS

This is the equivalent of the TAS Professional 3.0 command Equals Portion Of.

PROGRAM EDITOR 3.0 Commands -> Equ Mid

ERROR

Use this command to display an error message at the standard message location. You can also set the internal System Error Number field.

ERR *message* **NUM** *number*

message - f/c/e - Required - You may use this in two ways. It can be set as an A type field and the program will display the field as the error message. It can also be set as a numeric value (any appropriate type) and the program will read the appropriate record in the ERRMSG.B file. The value set by you must relate to the ERROR_NUM field in the ERRMSG record. For more information please see **Chapter 3, Main Menu**.

number - f/c/e - Optional - If this value is set it will be saved in the internal System Error Number field. You can access this value elsewhere in the program through the **PERR()** function.

COMMENTS

By using the ERRMSG.B file as the storage for error messages you can easily add, change, delete, etc. any appropriate message. You should start any message numbers at 5000 or more. This will assure that no messages will have to change due to interference with records saved by CAS and provided with the system. Obviously, the error number values don't have to be sequential.

PROGRAM EDITOR User interface -> Messages -> Error

SEE ALSO PERR()

EXECUTE PROGRAM

You would use this command to execute a non-TAS Professional 5.1 program.

EXEC *program_name* **WITH** *with*

program_name - f/c/e - Required - This is the name of the program to be run. This must include any path if applicable.

with - f/c/e - Optional - The program will pass this value as the 'tail' of the command line. For example:

EXEC 'EDLIN' WITH 'TEST.SRC'

would be the equivalent to typing the following at the DOS prompt:

EDLIN TEST.SRC

COMMENT

When the non-TAS program has finished executing, control will be passed to the line following the **EXECUTE PROGRAM** command.

NOTE: You can pass a P type pointer (or any other numeric value) to the program you're running by converting it to an alpha field first using the **STR()** function.

Windows Note: You can only execute another Windows program with this command. It will not execute a DOS program or command.

PROGRAM EDITOR

System -> Other programs -> Execute program

SAMPLE PROGRAM

RUNTEST, RUNTEST1

SEE ALSO

EXEC()

EXIT

This is a process control command, i.e., it will exit from a **WHILE** loop. This is one part of a complete command structure.

EXIT

No options

COMMENTS

For more information on the different structured programming commands, and the **EXIT** command in particular, please see **Chapter 7, Structured Programming Commands.**

PROGRAM EDITOR

Prg control -> While -> Exit

SEE ALSO

WHILE, LOOP, LOOP_IF, EXIT_IF, ENDW

EXIT_IF

This is a process control command, i.e., it will control whether to exit from a **WHILE** loop. This is one part of a complete command structure.

EXIT_IF *expression*

expression - lexpr - Required - If the **expression** resolves to .T. then the program will transfer control to the line following the appropriate **ENDW** (Endwhile) command.

COMMENTS

For more information on the different structured programming commands, and the **EXIT_IF** command in particular, please see **Chapter 7, Structured Programming Commands.**

PROGRAM EDITOR

Prg control -> While -> eXit_if

SEE ALSO

WHILE, LOOP, LOOP_IF, EXIT, ENDW

EXPORT

This command will write records to a non-TAS file. Within this single command you can accomplish many tasks that would normally take many more lines of code and time to execute.

EXPORT *from_field_list MEM* **FILE** *filename/@file_number* **KEY** *keyname/@key_number*
START *start_value* **SCOPE** *scope scope_value* **NUM** *number* **CNTR** *counter_field* **FOR** *for_filter_expression* **WHILE** *while_filter_expression* **TO** *to_file*
TYPE *file_type* **DLM** *delimiter_character APND DISP*

from_field_list - fn/v/e1, fn/v/e2,..., fn/v/ex - Required - The fields in the record that are being exported. You may also create expressions that can be exported.

MEM - Optional - If you are exporting from an array in memory instead of a standard file include this option in the command.

filename/@file_number - file_expr - Optional - The name or number of the file to be used. If you do not include this option, the program will look for a default search file set in the **SRCH** (Search File) command. If a Search File hasn't been previously set, the program will report an error and the command will be skipped. This entry will override any default value set in the **SRCH** command.

keyname/@key_number - key_expr - Optional - Set this to the appropriate value to export the records in the file in the correct order. If you do not include this option, the program will look for a default key set in the **SRCH** command. If a default key hasn't been previously set the program will report an error and the command will be skipped. This entry will override any default value set in the **SRCH** command.

start_value - f/c/e1,f/c/e2,...,f/c/ex - Optional - The value to use as the beginning record. This is similar to doing a **FIND** before the command. If the index you're searching on has multiple segments you may list the proper value for each segment, separating them with commas. This will allow you to specify the exact type for each of the segments. For example, suppose you're searching on an index that has two segments: a 5 character alpha and an I type field. Each record you want has the same alpha but the I field will change, and is in order. In the first record the I type field has a value of 0. The following would find the first record for that group, if it exists:

```
start 'ABCDE',0!
```

Notice that we separate the values with a comma. The only requirement is that the values be of the same type (and size) as the segments in the key. In this case we put the I type constant specifier (!) after the 0 to make sure that it is passed as an I type field.

scope - Optional - This option may help determine how many records are exported. For more information about the **scope** specifier, please see the general information at the beginning of this chapter.

scope_value - f/c/e - Optional - If **scope** is set to **N** or **F** this is the number of records to exported.

number - fn/v - If *MEM* is set then this is Required - If *MEM* is set, this is the number of elements in the array to be exported.

counter_field - fn/v - Optional - This is an I type field that will be used for passing the number of records read, and the current array number, to your program. You can also use this to count the number of records read.

for_filter_expression - lexpr - Optional - You would use this option to restrict the records exported in this command. If the expression did not resolve to .T. the record would not be exported. In this option, the search continues until the program reaches the end of file and then will quit. You can also use this to perform the same task if the *MEM* option is set. Use the **counter_field** as the array specifier in the expression.

while_filter_expression - lexpr - Optional - This option also restricts the records exported. However, the first time the expression resolves to .F., the program stops reading records and continues to the next command. Therefore, if the program is reading records in a certain order (by a specific key) and the expression returns .F., then the program knows that all the appropriate records have been read and there is no need to continue reading. For example: You want to export all the invoice records for a specific customer. The first record for that customer is found in the invoice file either by using the **start_value** option within this command, or through the **FIND** command before executing this command. Then the **while_filter_expression** would be:

INV_CUST_CODE = CUSTOMER_CODE

(This assumes that there is a field called INV_CUST_CODE in the invoice file and a similar field in the customer file called CUSTOMER_CODE. The **keyname/ @key_number** option must be set to the proper value so that the records are read in CUSTOMER_CODE order, or you must have executed the **SRCH** command previous to this command.) When the first invoice record for the next customer is read the program will stop reading records.

NOTE: If there is no **start_value** you must set the **scope** value to *R* or *N xxx*. If this is not done, the program will find the first record in the file and the **while_filter_expression** option will probably fail the first time. However, if the **start_value** option is used you can ignore this requirement.

to_file - f/c/e - Required - The name of the file you are exporting to. Must include the entire path, if any, and any extension. The program will use the exact name you enter.

file_type - f/c/e - Required - The type of file you are exporting to. The options are:

D - dBASE III+
L - delimited
F - fixed length/SDF
Y - SYLK
I - DIF
X - text

Don't forget: this option entry needs to be a constant ('L', or 'D', etc.) or a field that contains a single character that matches one of the above.

NOTE: In dBASE III+ format the program will create the header information automatically. You do need to remember that maximum field name sizes in dBase III+ are 10 characters and they are 15 in TAS. When we create the dBase III+ file we truncate the trailing 5 characters. You might want to take that into consideration when you are assigning field names if you are going to export to this format regularly.

delimiter_character - f/c/e - Optional - If you choose the delimited ('L') type of file, you can also choose a delimiter character. If you don't specify this value the program will put commas (,) between each field and surround alpha fields with quotes ("xxx").

APND - 'Y' or 'N' - Optional - Normally the **EXPORT** command will create a new file each time it is run. However, you may choose to add records to an existing file by selecting this option. The full option is APND 'Y' to append records to a current file and APND 'N' to create a new file.

NOTE: The program doesn't check to make sure that the previous records are in the same format.

DISP - Optional - Each time a record is read the program will redisplay the screen fields, if you specify this option. This will slow down the operation of this command, the amount of slow down depending on how many fields are displayed on the screen. The net effect will probably be negligible unless you are reading through a very large file.

COMMENTS

If the **while_filter_expression** option is used you must watch out for the proper setting of the **scope**, and/or **start_value** options also. If you think everything is set properly and yet no records are being exported, make sure that the proper first record is in memory prior to the execution of this command or that correct use has been made of the **start_value** option. The last record read before exiting the command is still in the record buffer after the program leaves this command.

PROGRAM EDITOR

fiLe -> Mult rec cmds -> Export

SEE ALSO

IMPORT

FEXIT

This is a process control command, i.e., it will exit from a **FOR/NEXT** loop. This is one part of a complete command structure.

FEXIT

No options

COMMENTS

For more information on the different structured programming commands, and the **FEXIT** command in particular, please see **Chapter 7, Structured Programming Commands.**

PROGRAM EDITOR Prg control -> For -> Exit

SEE ALSO FOR, FLOOP, FLOOP_IF, FEXIT_IF, NEXT

FEXIT_IF

This is a process control command, i.e., it will control whether to exit from a **FOR/NEXT** loop. This is one part of a complete command structure.

FEXIT_IF *expression*

expression - lexpr - Required - If the **expression** resolves to .T., then the program will transfer control to the line following the appropriate **NEXT** command. The counter value will remain at the last value.

COMMENTS

For more information on the different structured programming commands, and the **FEXIT_IF** command in particular, please see **Chapter 7, Structured Programming Commands.**

PROGRAM EDITOR Prg control -> For -> eXit_if

SEE ALSO FOR, FLOOP, FLOOP_IF, FEXIT, NEXT

FILL

Use this command to put a single character multiple times into a single field or multiple array elements of the same field.

FILL *fieldname TRAIL/LEAD/ALL* **CHR** *character_to_use* **TIMES** *times*

fieldname - fn/v - Required - The field to be filled. May not be a pointer (P or F), but any other type is legal. You can use the redirector and a pointer to point to a legal field.

TRAIL/LEAD/ALL - Optional - If the field being filled is of A type you may set whether to fill the trailing (*TRAIL*) or leading (*LEAD*) characters, or all (*ALL*). The program will put the **character_to_use** in each receiving position starting with the last (and then decrementing) or the first (and then incrementing) until the program finds the first character with an ASCII value greater than a space (ASCII 32). In the case of *ALL* the program will replace all characters in the field.

NOTE: All displayable characters are greater than a space. The default value is *TRAIL*.

character_to_use - f/c/e - Optional - The fill character. This really isn't optional, however, if this value is not set by you the default is binary 0. You may use any type of field or expression, but the program will only use the first character (byte) of the resultant field.

times - f/c/e - Optional - If the field being filled is an array you may stipulate how many elements to fill using this option. You may also specify the beginning array element number, if other than 1, in the fieldname in the normal fashion.

COMMENTS

If the field being filled is not an A type, the program will fill the entire field with **character_to_use**.

If you want to use a binary fill character it can be passed as an I or B type field or through the **CHR()** function.

PROGRAM EDITOR Field -> alpha Fld cmds -> Fill

SEE ALSO TRIM, FILLM, JUST(), FILL()

FILL MEMORY AREA

This is a TAS Professional 3.0 command here for compatibility. There is no preferred TAS Professional 5.1 command/function.

FILLMEM *mem_area#* **START** *start_val* **NCHR** *num_chrs* **CHR** *chr_to_use*

mem_area# - f/c/e - Required - This is the memory area to fill. This must resolve to a value of 1 through 4.

start_val - f/c/e - Required - The starting character in the memory area. The first character is at position 1.

num_chrs - f/c/e - Required - The number of characters to fill.

chr_to_use - f/c/e - Required - What to use as a fill character.

COMMENTS

This is the equivalent of the TAS Professional 3.0 command Fill Memory.

PROGRAM EDITOR 3.0 Commands -> Fillmem

FILTER (*DOS only*)

This command will set an expression as a for_filter to be used during a file search unless overridden in the command. This will affect all searches from the keyboard.

FILTER *expression*

expression - lexpr - Required - This is the expression to test against. It will literally test the **expression** against each record found. If it resolves to .F., the program will search for the next record and will continue until it reaches the end of the file. Unlike a normal search if the program reaches the beginning or end of the file the record will be cleared.

COMMENTS

This is a very powerful feature of TAS Professional 5.1. Through the use of this command you are able to change the 'look' of the file at runtime, on the fly, with no changes to your program. We use this feature in the Maintain Database and Export and Import utilities. Also, any programs you generate from

the Report Writer can be easily set up to have this same feature without any programming!

PROGRAM EDITOR fiLe -> Find -> fiLter

FIND

Use this command to find a record in any TAS file opened with the **OPEN** command.

FIND *find_type* **SRCH** *key_or_file_name* **REL** *related_field* **ERR** *error_label*
FOR *for_filter_expression NLOCK KEYO NOCLR*

find_type - mgflnpr - Required - This is the type of **FIND** you want to execute. The options are:

M - matching - Find the exact record based on the **search_value** option. You can also set the appropriate key fields with the **EQUAL** command and then the **search_value** option need not be used.

G - generic - Same as match however the program will return the exact or next greatest record.

F - first - Find the first record in the file.

L - last - Find the last record in the file.

N - next - Find the next record. This assumes that you have done a search previously that successfully found a record in the file (exact, generic, first, etc.).

P - previous - Find the previous record. This assumes that you have done a search previously that successfully found a record in the file (exact, generic, last, etc.).

R - related - In this type of search, the program will use the value in the **related_field** when searching for the record. **NOTE**: There is a process in TAS Pro 5.1 where you can set relations between files outside of the **FIND** command. It is recommended that you use that method. Then TAS Pro 5.1 will automatically search for related records whenever a record is found using any method.

key_or_file_name - sac - Required - If this is a type **N** or **P** (next or previous) search, then this value is the name of the file being searched in special alpha constant form (i.e., the true name of the file without quotes). Otherwise this is the name of the key being used for the search, also in special alpha constant form.

related_field - fn/v - The name of the field to use as the search value for this command.

NOTE: There is a process in TAS Pro 5.1 where you can set relations between files outside of the **FIND** command. It is recommended that you use that method. Then TAS Pro 5.1 will automatically search for related records whenever a record is found using any method.

error_label - label - Optional - If you specify a label here, the program will transfer control to the appropriate line if an error occurs during the command.

NOTE: We do not recommend using this method to check for an error. You can test for an error with the **FLERR(FNUM())** functions and create a more graceful process to continue operation rather than just a GOTO, which is the effect of this option.

for_filter_expression - lexpr - Optional - You would use this option to restrict the records read in this command. If the **expression** did not resolve to .T. the record would not be found. In this option, the search continues until the program reaches the end of file (EOF = .T.) and then will quit.

NLOCK - Optional - If this option is included in the command the program will NOT lock the record upon finding it as it would normally do in a multi-user situation. The default operation is to place a lock on the record upon reading it.

KEYO - Optional - If this option is included in the command the program will search only for the key. The result is that you can use this method for checking if a key value exists without having to find the entire record. This will preserve the values in your record until you are ready to save it. To test whether or not the record exists use the **FLERR()** function without specifying the file number.

NOCLR - Optional - Normally, during a **M** (match) type **FIND**, if a record is not found the program will clear the record buffer. If you don't want this to happen then include this option. If a record was active before the **FIND** it will still be active. If you had data as part of a new record then that data will still be there.

COMMENTS

Generally, if an error is returned during the **FIND** operation, the record for that file will not be active and will be cleared (no active data). However, this is not the case for the NEXT/PREVIOUS options. If there is an active record in the buffer and the user/program attempts a NEXT or PREVIOUS and reaches the end/beginning of the file, that record will still exist and will still be active.

NOTE: This is the older method of finding records used generally in TAS Professional 3.0 and before. It is recommended that you use the **VARIABLE** commands (**OPENV** (Open Variable), **FINDV** (Find Variable), etc.) since they are more flexible and have more options. The choice, however, is up to you.

PROGRAM EDITOR

fiLe -> Find -> Find

SEE ALSO

OPEN

FIND VARIABLE

Use this command to find a record in any file, either TAS (opened with the **OPENV** (Open Variable) command) or non-TAS.

FINDV *find_type* **FNUM** *file_number* **KEY** *keyname/@key_number*
VAL *search_value* **ERR** *error_label* **FOR** *for_filter_expression*
NLOCK KEYO NOCLR

find_type - mgflnp - Required - This is the type of find you want to execute. The options are:

M - matching - Find the exact record based on the **search_value** option. You can also set the appropriate key fields with the **EQUAL** command and then the **search_value** option need not be used.

G - generic - Same as match, however the program will return the exact or next greatest record.

F - first - Find the first record in the file.

L - last - Find the last record in the file.

N - next - Find the next record. This option assumes that you have done a search previously that successfully found a record in the file (exact, generic, first, etc.).

P - previous - Find the previous record. This option assumes that you have done a search previously that successfully found a record in the file (exact, generic, last, etc.).

file_number - f/c/e - Optional - This is the same value that is returned in the **OPEN VARIABLE** command. If you do not include this option, the program will look for a default **file_number** set in the **SEARCH_KEY** command. If that default hasn't been previously set, the program will report an error and the command will be skipped. This entry will override any default **file_number** value set in the **SEARCH_KEY** command.

keyname/@key_number - key_expr - Optional - Set this to the appropriate value to search the records in the file in the correct order. If you do not include this option, the program will look for a default key set in the **SRCH** command. If a default key hasn't been previously set, the program will report an error and the command will be skipped. This entry will override any default value set in the **SRCH** command.

search_value - f/c/e1,f/c/e2,...,f/c/ex - Optional - This is similar to setting each segment in the key to a value separately. If the index you're searching on has multiple segments you may list the proper value for each segment, separating them with commas. This will allow you to specify the exact type for each of the segments. For example, suppose you're searching on an index that has two segments: a 5 character alpha and an I type field. Each record you want has the same alpha, but the I field will change, and is in order. In the first record the I type field has a value of 0. The following would find the first record for that group, if it exists:

val 'ABCDE',0!

Notice that we separate the values with a comma. The only requirement is that the values be of the same type (and size) as the segments in the key. In this case we put the I type constant specifier (!) after the 0 to make sure that it is passed as an I type field.

error_label - label - Optional - If you specify a label here the program will transfer control to the appropriate line if an error occurs during the command.

NOTE: We do not recommend using this method to check for an error. You can test for an error with the **FLERR()** function and create a more graceful process of continuing operation than just performing a GOTO, which is the effect of this option.

for_filter_expression - lexpr - Optional - You would use this option to restrict the records read in this command. If the **expression** did not resolve to .T., the record would not be found. In this option, the search continues until the program reaches the end of file (EOF = .T.) and then will quit.

NLOCK - Optional - If this option is included in the command the program will NOT lock the record upon finding it as it would normally do in a multi-user situation. The default operation is to place a lock on the record upon reading it.

KEYO - Optional - If this option is included in the command the program will search only for the key. You can use this method to check if a key value exists without having to find the entire record. This will preserve the values in your record until you are ready to save it. To test whether or not the record exists, use the **FLERR()** function without specifying the file number.

NOCLR - Optional - Normally, during a **M** (match) type **FINDV**, if a record is not found the program will clear the record buffer. If you don't want this to happen then include this option. If a record was active before the **FINDV**, it will still be active. If you had data as part of a new record then that data will still be there.

COMMENTS

Generally, if an error is returned during the **FINDV** operation the record for that file will not be active and will be cleared (no active data). However, this is not the case for the NEXT/PREVIOUS options. If there is an active record in the buffer and the user/program attempts a NEXT or PREVIOUS and reaches the end/beginning of the file, that record will still exist and will still be active.

You can use this command to search for a value in a non-TAS file by setting the **KEY** to @0 (or by not including it at all) and putting the value you are searching for as a logical expression in the **FOR** option. For example:

```
FINDV f ..... FOR fld_name='ABCD'
```

Where fld_name is a field in the non-TAS file being searched. The command above will return the first occurence of the value 'ABCD'.

PROGRAM EDITOR

fiLe -> Find -> find Variable

SAMPLE PROGRAM

NONTSRCH

SEE ALSO

OPENV

FLOOP

PURPOSE

This is a process control command, i.e., it will go back to the beginning in a **FOR/NEXT** loop. This is one part of a complete command structure.

FLOOP

No options

COMMENTS

For more information on the different structured programming commands, and the **FLOOP** command in particular, please see **Chapter 7, Structured Programming Commands.**

PROGRAM EDITOR Prg control -> For -> Loop

SEE ALSO FOR, FLOOP_IF, FEXIT, FEXIT_IF, NEXT

FLOOP_IF

This is a process control command, i.e., it will control whether to loop back to the beginning in a **FOR/NEXT** loop. This is one part of a complete command structure.

FLOOP_IF *expression*

expression - lexpr - Required - If the **expression** resolves to .T., then the program will transfer control to the appropriate **FOR** line. This would be the equivalent of executing the **NEXT** command.

COMMENTS

For more information on the different structured programming commands, and the **FLOOP_IF** command in particular, please see **Chapter 7, Structured Programming Commands.**

PROGRAM EDITOR Prg control -> For -> loop_If

SEE ALSO FOR, FLOOP, FLOOP_IF, FEXIT, NEXT

FOR

This is the first part of the **FOR/NEXT** loop. In this part of the command you can specify the counter, start value, stop value, and the amount to add to or subtract from the counter each time the loop is executed.

FOR*(counter;start_value;stop_value;step_value)*

counter- fn - Required - The name of the field used as a loop counter.

start_value - f/c/e - Required - The first value to be used in the loop. The **counter** field is initialized to this value the first time through.

stop_value - f/c/e - Required - When the **counter** reaches this value the program knows to exit the loop at the **NEXT** command.

step_value - f/c/e - Required - How much to add to or subtract from **counter** each time through the loop. If you wish to subtract (or decrement) **counter** in the loop, precede the value with the minus sign. For example, a **step_value** of -1 will reduce the **counter** value by 1 each time the loop is executed.

COMMENTS

All fields/values in the **FOR** command except **counter** must be of I type. The **counter** can be either I or R type. The loop will always be executed at least once so you need to make sure that the actions to be taken are acceptable at all times. The loop will continue through the stop_value.

For more information on the different structured programming commands, and the **FOR** command in particular, please see **Chapter 7, Structured Programming Commands.**

PROGRAM EDITOR Prg control -> For -> For

EXAMPLE

```
FOR(cntr;1;10;1)
   ? x
NEXT
```

The results will be:

```
1
2
3
4
5
6
7
8
9
10
```

SEE ALSO FLOOP, FLOOP_IF, FEXIT, FEXIT_IF, NEXT

FORCE

This can be used to tell the program to ignore any active windows when doing output to the screen.

FORCE *ON/OFF*

ON/OFF - Required - If this option is *ON* the program will ignore any windows when displaying information to the screen. To return to normal mode the option would be *OFF*.

COMMENTS

You can also use the **abs** option in **PMSG** (Print Message), **CLRLNE** (Clear Line), etc. Don't forget to turn this off (**FORCE** *OFF*) when you're done.

PROGRAM EDITOR User interface -> Windows -> Force

FORCE3

This is a TAS Professional 3.0 command here for compatibility. The preferred TAS Professional 5.1 command is **FORCE**.

FORCE3 *Y/N*

Y/N - Required - If this option is *Y* the program will ignore any windows when displaying information to the screen. To return to normal mode the option would be *Y*.

COMMENTS

This is the equivalent of the TAS Professional 3.0 command Print Outside Window.

PROGRAM EDITOR 3.0 Commands -> Force3

FOREGROUND (*DOS only*)

This command will change the screen output color to the value set as the **normal** color in the **COLOR** command.

FRG

No options

PROGRAM EDITOR User interface -> Color control -> Foreground

SEE ALSO BKG, REV, CC(), CCE(), CCF(), CCH(), CCR()

FORMAT

This command will format an alpha or numeric field for a specific look during output.

FORMAT *fieldname* **RECV** *receiving_field* **NEG** *neg_how* **PICT** *picture*
NOCMA NOFD NOZERO OFF

fieldname - *fn/v* - Required - The name of the field to be formatted. May be either a numeric or alpha field.

receiving_field - *fn/v* - Optional - If you want to put the formatted numeric field in another field this is the name of the A type receiving field. This only applies if the format field is of type N.

neg_how - Optional - LTPCA - **L** - Leading negative sign, **T** - Trailing negative sign, **P** - Parentheses, **C** - Trailing CR, **A** - Angle brackets. This is applicable only when the format field is of type N.

picture - *f/c/e* - Required - This is used to create what is called a 'slider field'. Slider fields are useful when you have a field you want to display on the screen, and that field is too wide to fit within the screen or current window. You can specify how many characters to display by preceding that value with an 'S'. For example:

'S30'

will display 30 characters of the field at any time. When the user enters that field it will start at the 'beginning.' As characters are entered the data will appear to 'slide' across the entry box. The user may press the LEFT and RIGHT arrows to go back and forth in the field. Also the HOME key goes to the beginning of the field, and the END key to the current ending character in the field. This allows you to enter large fields without having to use the TASEdit program or an outside editor. This only applies when the format field is of type A.

NOCMA - Optional - If this option is set the program will NOT include comma characters in the resulting formatted field. The default is to include comma characters. This only applies when the format field is of type N.

NOFD - Optional - If this option is set the program will NOT include a floating dollar sign in the resulting formatted field. The default is to include a floating dollar sign. This only applies when the format field is of type N.

NOZERO - Optional - If the field value is 0 don't print anything. This only applies when the format field is of type N, D (date), or T (time). Use this option if you want dates or times to print as blanks.

OFF - Optional - Turn off any current formatting. This only applies when the format field is of type N.

COMMENTS

If the formatted field will not fit within the display size, the command will return the field as is, without any of the formatting characters. This applies only to numeric fields.

When formatting an array you need specify only the major field name. For example, to have all elements of the field AMT_REMAIN which is defined as a N type size 10 dec 2, use the command:

```
FORMAT amt_remain
```

Don't specify the array element. However, if you use an expression for the element specifier, e.g.:

```
? amt_remain[x+1]
```

the program will not be able to print the field properly formatted. In this case you should use a UDF perhaps in the form of:

```
? fmt_fld(amt_remain[x+1])
....

func fmt_fld fmt_fld_val
        define fmt_fld_hldr type a size 10
        format fmt_fld_val recv fmt_fld_hldr neg a
        ret fmt_fld_hldr
```

This would print the field properly formatted.

If you use the **PICT** option in this command you must add a field to your program for each format command that uses that option. Do this with the **#ADD_FLDS** compiler directive.

PROGRAM EDITOR Field -> Create/chg -> Format

FUNCTION

This is the command that defines a User Defined Function, or UDF. A UDF may be used in any situation that would allow a regular (standard) function. Some options in some commands require that a UDF be used.

FUNC *function_name function_parameters*

function_name - sac - Required - This is a special alpha constant. It is the name you give to this function. It may be up to 14 characters long. It must not be the same as any line label used anywhere else in the program.

function_parameters - *fn/v1,fn/v2,...,fn/vx* - Optional - The values being passed to the UDF. A maximum of 20 fields may be specified. The field names are separated with commas.

COMMENTS

You must include the compiler directive **UDF** before the first user defined function is referenced in the program. All functions end with the command **RET** (Return). You may return a single value of any type, including an expression. Fields within the UDF are not automatically local. However, you may use the **PUSHF** (Push Field)/**POPF** (Pop Field) commands for saving/restoring all appropriate field values. Through this process the UDF may be made reentrant.

Any legal TAS Professional 5.1 command may be used in conjunction with a UDF.

Any UDF can be called as though it were a subroutine instead of a function. For example, a UDF with the name of CALC_EXP can be included as part of an expression:

```
TOTAL_EXP=CALC_EXP()
```

You can also GOSUB CALC_EXP. Note that in this case, you don't include the parentheses after the name, and you cannot pass data to the expression. So: you have two different ways to access any function.

If you do a GOSUB to a function, there is no built-in method of finding out what the returned value was. A function called RETVAL() allows you to do just that and must be the next executed line after the GOSUB.

Suppose you have a subroutine you want to be able to access two different ways: both as a true UDF and as a regular subroutine. However, you want to check for successful execution by returning .T. or .F. Structured as a standard UDF, that's very simple. For example:

```
IF DO_SUB() xxxx
```

will do or not do whatever depending on whether DO_SUB() returns .T. or .F. If you instead GOSUB DO_SUB and place this:

```
IF RETVAL() xxxx
```

on the next line after the GOSUB, the effect would be the same. RETVAL can be used only for testing the returned value from a UDF. You cannot just RET a value from a standard subroutine; the value will be ignored.

This process is used extensively in the Report/2 format report writer. Header blocks are accessed both as standard subroutines and as UDFs. The routines are actually called using GOSUBL, since it's easy to obtain a line number for the UDF just as though it were a standard line label. You can see this in action by creating a report using this utility and generating the source code as appropriate.

PROGRAM EDITOR

System -> Programming -> udF

SAMPLE PROGRAM

UDFTEST.SRC

EXAMPLE

```
#udf

define x type n

clrscr
x = user_func(1,3,'A')
? x
quit

define y,z,a type n
define do type a size 1
func user_func y,z,do
    if do = 'A'
        a = y+z
    else_if do = 'S'
        a = y-z
    else_if do = 'M'
        a = y*z
    else_if do = 'D'
        a = y/z
    endif
    ret a
```

The above is a complete program. The result would be:

```
4.00
```

GET LABEL LINE NUMBER

This command will return the line number of a label for future use in the **GOTOL** (Goto Line) or **GOSUBL** (Gosub Line) command.

GETLBL *label_name* **FLD** *receiving_field*

label_name - label - Required - The label name you want to convert to a line number.

receiving_field - fn/v - Required - The field that will receive the line number. Must be of type I.

PROGRAM EDITOR

Prg control -> get label lN#

GOSUB

This command will temporarily transfer control to a subroutine. The first line of that routine must be a standard line label.

GOSUB *gosub_label*

gosub_label - label - Required - Transfer program control to the line label.

COMMENTS

A subroutine 'called' through this approach must use the **RET** (Return) command when finished. Unlike a function, no data may be returned. The program will continue with the next line after the original **GOSUB** (once the **RET** command has been executed from the subroutine).

PROGRAM EDITOR

Prg control -> Goto/gosub -> goSub

GOSUB LINE

This command will temporarily transfer control to a subroutine. In this case the program transfers control directly to a line number.

GOSUBL *gosub_line_number*

gosub_line_number - label - Required - Transfer program control to the line as specified by the number value.

COMMENTS

A subroutine 'called' through this approach must use the **RET** (Return) command when finished. Unlike a function, no data may be returned. The program will continue with the next line after the original **GOSUBL** (once the **RET** command has been executed from the subroutine).

NOTE: It is up to you to be sure that you pass a legal line number. If you do not the results are uncertain and may stop the program entirely.

PROGRAM EDITOR

Prg control -> Goto/gosub -> gosub linE

SEE ALSO

GETLBL

GOTO

This command will 'permanently' transfer control to a different part of the program. The first line of that routine must be a standard line label.

GOTO *goto_label*

goto_label - label - Required - Transfer control to the line label.

COMMENTS

This command, as opposed to the **GOSUB**, has no automatic way to return. You must use another **GOTO** to return to the next command, if desired. Generally, this command is used only in situations where you do not want to execute the lines following this command.

This command is not part of the regular 'structured' commands that a programmer would use in creating a program. Much effort has been made in TAS Professional 5.1 so that this command need not be used at all. However, it remains here for those times you find it the expedient choice.

PROGRAM EDITOR

Prg control -> Goto/gosub -> Goto

GOTO LINE

This command will 'permanently' transfer control to a different part of the program. In this case the program transfers control directly to a line number.

GOTOL *goto_line_number*

goto_line_number - label - Required - Transfer program control to the line as specified by the number value.

COMMENTS

This command, as opposed to the **GOSUB**, has no automatic way to return. You must use another **GOTO** (or **GOTOL**) to return to the next command, if desired. Generally, this command is used only in situations where you do not want to execute the lines following this command.

This command is not part of the regular 'structured' commands that a programmer would use in creating a program. Much effort has been made in TAS Professional 5.1 so that this command need not be used at all. However, it remains here for those times you find it the expedient choice.

NOTE: It is up to you to be sure that you pass a legal line number. If you do not the results are uncertain and may stop the program entirely.

PROGRAM EDITOR Prg control -> Goto/gosub -> goto Line

SEE ALSO GETLBL

GRAY (*Windows only*)

Use this command to automatically gray out inactive windows.

GRAY *ON/OFF*

ON/OFF - Required - If you set this option to *ON* any windows that become inactive after the command will be "grayed out." This means the type will be displayed in 'gray' format. What this is depends on the version of Windows you are running and what colors you have set for your screen. If you set this option to OFF then the gray process does not occur.

COMMENTS

It does take some extra time to gray out inactive windows and if you set this to **ON** it will slow your programs down to some extent. Again, this is very dependent on your computer, amount of memory, etc. You can also set this value in TP5WIN.INI. For more information on this please refer to **Chapter 11 - Windows Programming**.

PROGRAM EDITOR Win Commands -> Gray

HOT SPOT (*Windows only*)

Use this command to mark a spot on the screen where the user can click with their mouse. Since the Hot Spot itself is clear you would generally put this over another picture, or an area on the screen. Each time the user clicks the Hot Spot it will be just as though they had pressed the key that you setup as the **KEY** value.

HOT_SPOT AT *col,row* **LEN** *length* **WDT** *width* **KEY** *related_key* **NUM** *hot_spot_handle* *REMOVE*

col - f/c/e - Required - The column value for the upper left corner of the hot spot. The first column on the screen (far left position) is number 1. This value is always based on the largest window, generally the window that is created when the program is run. Even if the active window is smaller the column and row values are for the base window. This value is required only when creating the hot spot initially.

row - f/c/e - Required - The row value for the upper left corner of the hot spot. The first row on the screen (top row) is number 1. This value is always based on the largest window, generally the window that is created when the program is run. Even if the active window is smaller the column and row values are for the base window. This value is required only when creating the hot spot initially.

length - f/c/e - Required - The length or height of the hot spot in rows. This is required only when creating the hot spot initially.

width - f/c/e - Required - The width of the hot spot in columns. This is required only when creating the hot spot initially.

key - f/c/e - Optional - The keyboard key this hot spot will emulate when it is clicked. This is required only when creating the hot spot initially. If you don't specify a key value, or one doesn't match those available, the program will ignore any user clicks on the hot spot. The following are the acceptable key values:

F1 through F10
SF1 through SF10 (Shift F1 etc.)
^F1 through ^F10 (Ctrl F1 etc.)
@F1 through @F10 (Alt F1 etc.)
^A through ^Z (Ctrl A etc.)
@A through @Z (Alt A etc.)
ESC, UP (up arrow), DOWN (down arrow), LTA (left arrow), RTA (right arrow), HOME, END, PGUP, ^PGUP, PGDN, ^PGDN, INSERT, DELETE, WDLT (ctrl left arrow), WDRT (ctrl right arrow), TAB and BCKTAB.

An example of this option would be: ... Key 'ESC' ...

hot_spot_handle - fn/v - Required - When you initially create the hot spot you must supply a field that will hold the hot spot number or handle. This is how you will refer to this hot spot if you want to remove it from the screen. This must be an I type field.

REMOVE - Optional - Remove the hot spot from the screen. If the NUM (or handle) value is 0 then all hot spots will be removed from the screen.

PROGRAM EDITOR Win Commands-> Hot Spot

SEE ALSO BUTTON, PICTURE

IF

This is a process control command, i.e., it will control whether a command, or group of commands, will be executed. This is one part of a complete command structure.

IF *if_expression what_to_do goto_gosub_label*

if_expression - lexpr - Required - If the **if_expression** resolves to .T., the **what_to_do** action will be taken. If it is .F. and the **what_to_do** action is not *THEN*, the program will look for the next **ELSE_IF** or **ELSE** command. If one is not found the program will transfer control to the line following the appropriate **ENDIF** command.

what_to_do - Required - The action to take if the expression resolves to .T. Possible actions include:

DO - Do the following commands until the program reaches an **ELSE**, **ELSE_IF** or **ENDIF** command. This is the default value and doesn't have to be specified.

THEN - Execute the following command. The command to execute in this case must follow on the same physical line.

GOTO - Goto the line label specified in the command.

GOSUB - Gosub the line label specified in the command.

REENT - Reenter the last ENTER command.

RET - Return.

goto_gosub_label - label - Required - If the **what_to_do** option is *GOTO* or *GOSUB* then you must specify the label name here.

COMMENTS

For more information on the different structured programming commands, and the **IF** command in particular, please see **Chapter 7, Structured Programming Commands.**

PROGRAM EDITOR

Prg control -> If -> If

SEE ALSO

ELSE, ELSE_IF, ENDIF

IF DUPLICATE RECORD

This is a process control command, i.e., it will control whether a command, or group of commands, will be executed. This is one part of a complete command structure.

IFDUP *keyname what_to_do goto_gosub_label*

keyname - sac - Required - The program checks the file to see if a record exists with the value in the key field(s). If it does, the **what_to_do** action will be taken. If it does not and the **what_to_do** action is not *THEN*, the program will look for the next **ELSE_IF** or **ELSE** command. If one is not found, the program will transfer control to the line following the appropriate **ENDIF** command.

what_to_do - Required - The action to take if a record is found with the specified key value. Possible actions include:

DO - Do the following commands until the program reaches an **ELSE**, **ELSE_IF** or **ENDIF** command. This is the default value and doesn't have to be specified.

THEN - Execute the following command. The command to execute in this case must follow on the same physical line.

GOTO - Goto the line label specified in the command.

GOSUB - Gosub the line label specified in the command.

REENT - Reenter the last ENTER command.

RET - Return.

goto_gosub_label - label - Required - If the **what_to_do** option is *GOTO* or *GOSUB* then you must specify the label name here.

COMMENTS

Please note that the IFDUP command cannot tell if the duplicate record is active. Therefore you should not allow the user to re-enter (change) the relevant key field if the record is active. For example, you might use the following code:

```
Enter key_fld pre IFNA(file_handle)
```

For more information on the different structured programming commands, and the **IF** command in particular, please see **Chapter 7, Structured Programming Commands.**

PROGRAM EDITOR

Prg control -> If -> if Dup

SEE ALSO

ELSE, ELSE_IF, ENDIF

IF RECORD NOT ACTIVE

This is a process control command, i.e., it will control whether a command, or group of commands, will be executed. This is one part of a complete command structure.

IFNA *filename/@file_number what_to_do goto_gosub_label*

filename/@file_number - file_expr - Required - The name or number of the file to be checked.

what_to_do - Required - The action to take if the record is not active. Possible actions include:

DO - Do the following commands until the program reaches an **ELSE**, **ELSE_IF** or **ENDIF** command. This is the default value and doesn't have to be specified.

THEN - Execute the following command. The command to execute in this case must follow on the same physical line.

GOTO - Goto the line label specified in the command.

GOSUB - Gosub the line label specified in the command.

REENT - Reenter the last ENTER command.

RET - Return.

goto_gosub_label - label - Required - If the **what_to_do** option is *GOTO* or *GOSUB* then you must specify the label name here.

COMMENTS

For more information on the different structured programming commands, and the **IFNA** command in particular, please see **Chapter 7, Structured Programming Commands.**

PROGRAM EDITOR

Prg control -> If -> ifnA

SEE ALSO

ELSE, ELSE_IF, ENDIF

IMPORT

This command will import records from a non-TAS file and save them to a TAS Pro 5.1 file. Within this single command you can accomplish many tasks that would normally take many more lines of code and more time to execute.

IMPORT *to_field_list* **FROM** *from_file* **TYPE** *file_type*
DLM *delimiter_character* *MEM* **MAXA** *number* **CNTR** *counter_field*
TO *to_filename/@file_number* **SCOPE** *scope scope_value*
FOR *for_filter_expression* *DISP*

to_field_list - fn/v1, fn/v2,..., fn/vx - Required - The fields in the record that are being imported to. These must be real fields and cannot be expressions.

from_file - f/c/e - Required - The name of the file you are importing from. Must include the entire path, if any, and any extension. The program will use the exact name you enter.

file_type - f/c/e - Required - The type of file you are importing from. The options are:

D - dBASE III+
L - delimited
F - fixed length/SDF
Y - SYLK
I - DIF
X - text

Don't forget: this type code needs to be a constant ('L', or 'D', etc.) or a field that contains a single character that matches one of the above.

delimiter_character - f/c/e - Optional - If you choose the delimited ('L') type of file you can also choose a delimiter character. If you don't specify this value the program will assume commas (,) between each field and alpha fields surrounded with quotes ("xxx").

MEM - Optional - If you are importing to an array in memory instead of a standard file include this option in the command.

number - fn/v - If the *MEM* option is included, then this is Required - If *MEM* is included in the command, **number** is the maximum number of elements in the array to be imported to.

counter_field - fn/v - Optional - This is an I type field that will be used for passing the number of records read, and the current array number, to your program. You can also use this to count the number of records read.

to_filename/@file_number - file_expr - Required - The name or number of the file to be imported to.

scope - Optional - This option may help determine how many records are imported. For more information about the **scope** specifier, please see the general information at the beginning of this chapter.

scope_value - f/c/e - Optional - If **scope** is set to **N** or **F** this is the number of records to be imported.

for_filter_expression - lexpr - Optional - You would use this option to restrict the records imported in this command. If the expression did not resolve to .T., the record would not be imported. In this option, the search continues until the program reaches the end of file and then will quit. You can also use this to perform the same task if *MEM* is specified. Use the **counter_field** as the array specifier in the expression.

DISP - Optional - Each time a record is read the program will redisplay the screen fields, if you specify this option. This will slow down the operation of this command, with the amount of slow-down depending on how many fields are displayed on the screen. The net effect will probably be negligible unless you are reading through a very large file.

PROGRAM EDITOR

fiLe -> Mult rec cmds -> Import

SEE ALSO

EXPORT

INCREMENT

Increment an I or R type field.

INC *fieldname*

fieldname - fn/v - Required - The name of the field being incremented. Must be of I or R type.

COMMENTS

This is the same as adding 1 to the field. That is:

INC *CNTR*

is equivalent to:

CNTR = CNTR + 1

This command provides a more succinct method of adding 1 to a field. This command also runs a little faster than the standard addition routine.

PROGRAM EDITOR Field -> Increment

SEE ALSO DEC

INITIALIZE TAS FILE

This command is used to create (initialize) TAS Professional 5.1 files.

INIFLE *filename* **SPECS** *specifications CNF*

filename - f/c/e - Required - The name of the file to be created. The file name must include the standard file name, the extension, and the file path, if any.

specifications - fn/v - Required - This is the field containing the information Btrieve[tm] will use to create the data file.

CNF - Optional - If this option is included in the command the program will confirm that the user wants to create the file before trying to create it. However, if the file already exists, the program will alert the user that the file exists and confirm that they wish to initialize it irregardless of this option.

COMMENTS

You must have a full copy of the developer's version of Btrieve[tm] available from Novell before attempting to use this command. The requirements for properly setting up the **specifications** buffer are beyond the scope of this document.

PROGRAM EDITOR System -> File commands -> Init File

INSERT

Use this command to replace temporary place-keeping symbols in an A type field with other A type fields. The general use of this command will be in inserting names, addresses, amounts, etc. into stock paragraphs, letters, etc. that can then be printed with this 'personal' information in it.

INSERT *field_list* **INTO** *into_field*

field_list - f/c/e1, f/c/e2,..., f/c/ex - Required - The fields/expressions that will be used when replacing the place-keeper values. The first field will replace place-keeper 1, the second field place-keeper 2, etc. Each **INSERT** value must be of A type. The place-keeper value is '*'+number, i.e., *1 or *2, etc. Each **INSERT** field may be used multiple times within the **into_field**.

into_field - fn/v - Required - The name of the field that contains the place-keeping values that need to be replaced. Must be an A type field.

COMMENTS

The process is to remove the place-keeping characters and then insert the appropriate **INSERT** field. If the total size of the 'new' value is beyond the size of the field the overflow characters at the end will be deleted.

PROGRAM EDITOR

Field -> alpha Fld cmds -> Insert

EXAMPLE

Assume that the value of field X is:

Now is the *3 for all good *2 *1 come to the aid of their party. Now is the *3 for all good *2 *1 come to the aid of their party. Now is the *3 for all good *2 *1 come to the aid of their party.

After the execution of the command:

INSERT 'to','men','time' INTO X

The value of field X will now be:

Now is the time for all good men to come to the aid of their party. Now is the time for all good men to come to the aid of their party. Now is the time for all good men to come to the aid of their party.

SAMPLE PROGRAM

INSERT

SEE ALSO

DELC

INTERRUPT (*DOS only*)

Use this command to call any DOS, BIOS, etc. interrupt directly from TAS Professional 5.1.

INT *int_num* **REGS** *register_field*

int_num - f/c/e - Required - This is the decimal form of the interrupt number. Make sure you're using the decimal and not hex representation.

register_field - fn/v - Required - The first field in the register list.

COMMENTS

TAS Professional 5.1 can do a lot, but it can't do everything! Using this command you can do almost all the rest. This is like writing in assembly language without having to worry about all the problems you typically encounter when writing in assembly. You can access many things in Netware, or LANTASTIC, etc. without having to write outside programs.

A word of warning: if you don't understand everything about the interrupt you're using, don't try this. You can easily shut down the system using this feature. It is provided so that skilled programmers can more easily and quickly access features they couldn't before.

We recommend you use the following register list just as we list it below:

```
define reg_al,reg_ah,reg_bl,reg_bh,reg_cl,reg_ch,reg_dl,reg_dh type b
define reg_si,reg_di,reg_ds,reg_es type i
define carry,zero type l
```

You can name the registers anything you want; however, they must be in the order listed above. Using the **REDEFINE** command you can create 16 bit values that overlay two 8 bit regs. For example:

```
define reg_dx type i
redefine reg_dx loc reg_dl
```

The following program achieves the same result as the GETDOSV program shown in the documentation for the **LOAD** command. However, this program doesn't need an outside assembler program to get the results. Compare the differences:

```
;GETDOSV in TAS using the INT command.
;
;Returns DOS version
;
;define the register list
;
define reg_al,reg_ah,reg_bl,reg_bh,reg_cl,reg_ch,reg_dl,reg_dh type b
define reg_si,reg_di,reg_ds,reg_es type i
define carry,zero type l
;
;now do the DOS call
;
reg_ah=48
int 33 regs reg_al          && int 33 is 21h
;
;the version is returned in reg_al and reg_ah
;
clrscr
? 'DOS VERSION: ',trim(str(reg_al),'l'),'.',trim(str(reg_ah),'l') wait
;
;that's all there is to it!
```

PROGRAM EDITOR System -> Programming -> Interrupt

SAMPLE PROGRAM INTTEST

SEE ALSO OFST(), TEST()

JUSTIFY FIELD

This will move the characters in an A type field, starting with the first non-space character, to the beginning of the field, to the end or will center the characters.

JUST*fieldname LEFT/RIGHT/CENTER*

fieldname - fn/v - Required - The name of the field to be justified.

LEFT/RIGHT/CENTER - Optional - Move the characters to the **LEFT** in the field, to the **RIGHT**, or **CENTER** them.

COMMENTS

When you use the function **JUST()** the program puts the field in the temporary data space. If you are trying to **JUST** a field that is larger than that space you will get the message that the program is out of room in the temporary data area. This command is provided to you so that you can **JUST** a field larger than the temporary data area within its own space. Using this command you will not run out of memory.

PROGRAM EDITOR Field -> alpha Fld cmds -> Justify

SEE ALSO TRIM()

KEYBOARD UPPER CASE

This can be used to turn on or off the shiftlock on the keyboard.

KBDUP *ON/OFF*

ON/OFF - Required - If this option is *ON* the program will force all characters entered through the keyboard to be in upper case. To return to normal mode the option would be *OFF*.

PROGRAM EDITOR System -> Keybd up case

LABEL

Specify a label name.

*label_name***:**

label_name - label - Required - The name of the label. Cannot have been used as a label anywhere else in the program.

COMMENTS

The label must be the first characters on the line, must be no more than 14 characters long, and must be terminated with a colon (:). Also, there can be no other commands on the same line.

PROGRAM EDITOR Prg control -> Line label

LIST (*DOS only*)

This command will list a file, or a group of array fields, to the screen, printer, or a disk file.

LIST *field_list* **TTL** *title_list MEM* **NUM** *number* **CNTR** *counter_field*
FILE *filename/@file_number* **KEY** *keyname/@key_number*
START *start_value* **SCOPE** *scope scope_value* **FOR** *for_filter_expression*
WHILE *while_filter_expression* **PTO** *print_to* **PFN** *print_to_file NOFF*

field_list - f/c/e1, f/c/e2,..., f/c/ex - Required - This is the list of fields to be used in this command.

If the *MEM* option is set then the program assumes that each field is an array field. Through the use of the **counter_field** you need not use just regular array fields. However, if just standard array fields are to be used then you need not include any array spec. For example, a normal array field would be used as: FIELD[ARRAY_ELEMENT]. When using a standard array field in this option, you would specify just the FIELD portion, and no array element. The program will automatically increment the array element number. You can also use the **counter_field** within the array expression. For example, if the **counter_field** is *CNTR* then the current array element number can be accessed by using *CNTR* as the array element field, i.e., FIELD[CNTR]. You can also specify the beginning array element number by setting this value in the **counter_field**.

A maximum of 80 fields/expressions/constants or any combination thereof may be included in a single **field_list** group.

title_list - f/c/e1, f/c/e2,..., f/c/ex - Optional - This list will be displayed after any automatic internal top of form and as the first line printed by the command. The information printed is not automatically lined up with the data so you need to be sure that it displays properly.

MEM - Optional - If you are listing from an array in memory instead of a standard file include this option in the command.

number - fn/v - If the *MEM* option is set, then this is Required - If *MEM* is specified then this is the number of elements in the array to be listed.

counter_field - fn/v - Optional - This is an I type field that will be used for passing the number of records read, and the current array number, to your program. You can also use this to count the number of records listed.

filename/@file_number - file_expr - Optional - The name or number of the file to be used. If you do not include this option, the program will look for a default search file set in the **SRCH** (Search File) command. If a default search file hasn't been previously set, the program will report an error and the command will be skipped. This entry will override any default value set in the **SRCH** command.

keyname/@key_number - key_expr - Optional - Set this to the appropriate value to list the records in the file in the correct order. If you do not include this option, the program will look for a default key set in the **SRCH** (Search File) command. If a default key hasn't been previously set the program will report an error and the command will be skipped. This entry will override any default value set in the **SRCH** command.

start_value - f/c/e1,f/c/e2,...,f/c/ex - Optional - The value to use as the beginning record. This is similar to doing a **FIND** before the command. If the index you're searching on has multiple segments you may list the proper values for each of the segments, separating

them with commas. This will allow you to specify the exact type for each of the segments. For example, suppose you're searching on an index that has two segments: a 5 character alpha and an I type field. Each record you want has the same alpha but the I field will change, and is in order. In the first record the I type field has a value of 0. The following would find the first record for that group, if it exists:

start 'ABCDE',0!

Notice that we separate the values with a comma. The only requirement is that the values be of the same type (and size) as the segments in the key. In this case we put the I type constant specifier (!) after the 0 to make sure that it is passed as an I type field.

scope - Optional - This option may help determine how many records are listed. For more information about the **scope** specifier, please see the general information at the beginning of this chapter.

scope_value - f/c/e - Optional - If **scope** is set to **N** or **F** this is the number of records to listed.

for_filter_expression - lexpr - Optional - You would use this option to restrict the records listed in this command. If the expression did not resolve to .T. the record would not be listed. In this option, the search continues until the program reaches the end of file and then will quit. You can also use this to perform the same task if *MEM* is specified. Use the **counter_field** as the array specifier in the expression.

while_filter_expression - lexpr - Optional - This option also restricts the records listed. However, the first time the expression resolves to .F., the program stops reading records and continues to the next command. Therefore, if the program is reading records in a certain order (by a specific key) and the expression returns .F., then the program knows that all the appropriate records have been read and there is no need to continue reading. For example: Suppose you want to list all the invoice records for a specific customer. The first record for that customer is found in the invoice file either by using the **start_value** option within this command, or through the **FIND** command before executing this command. Then the **while_filter_expression** would be:

INV_CUST_CODE = CUSTOMER_CODE

(This assumes that there is a field called INV_CUST_CODE in the invoice file and a similar field in the customer file called CUSTOMER_CODE. The **keyname/ @key_number** option must be set to the proper value so that the records are read in CUSTOMER_CODE order, or you must have set a default key using the **SRCH** command previous to this command.) When the first invoice record for the next customer is read the program will stop reading records.

NOTE: If there is no **start_value** you must set the **scope** value to *R* or *N xxx*. If this is not done, the program will find the first record in the file and the **while_filter_expression** option will probably fail the first time. However, if the **start_value** option is used you can ignore this requirement.

print_to - prt_where - Optional - Use this option to direct the output of this command. The options are S,P,D, or A. (For more details on these options, please see the information at the beginning of this chapter.) If **A** is set the program will ask the user at runtime where to print. If no **print_to** option is set the program will use the default value. This will be either the screen or the last value set with the **PRINT TO** command.

print_to_file - *f/c/e* - Optional - If you set the **print_to** option to **D**, the name of the file to use may be specified with this option. If the file name is not specified the program will ask for one at runtime.

NOFF - Optional - If this is included in the command the program will not do a top of form operation at the end of the command. Irregardless of the setting of this option, normal form feeds will be executed during the operation of this command if the report would take more than one page.

COMMENTS

If the **while_filter_expression** option is used you must be careful to make the proper setting of the **scope**, and/or **start_value** options also. If you think everything is set properly and yet no records are being listed, make sure that the proper first record is in memory prior to the execution of this command or that correct use has been made of the **start_value** option. The last record read before exiting the command is still in the record buffer after the program leaves this command.

This command provides you with an easy and quick method to display a group of fields from a file or from an array in memory. Any file, TAS or non-TAS, may be displayed.

PROGRAM EDITOR

fiLe -> Mult rec cmds -> List records

SAMPLE PROGRAMS

DLTEST, DLTEST2, LISTTEST

LIST ARRAY

This command will generate a list of array values. The user can scroll through the list using the UP and DOWN ARROW keys, PgUp and PgDn, or enter a single character, and depending on your options, find a value in the list.

All of the lookup windows in the TAS Professional 5.1 utilities use either this command or the **LISTF** (List File) command.

LISTM *field_list* **ENTER** *enter_udf* **SRCH** *search_udf* **CHSE** *chse_expr* **HELP** *help_udf* **OTH** *other_udf* **ACTV** *number_of_active_elements* **MAXA** *max_number_of_elements* **CNTR** *counter_field* **ON_MOVE** *move_udf* **CCOLOR** *choice_color* **STYP** *search_type* **FLINE** *first_line_fields* **CBF** *chrs_btwn_fields* **BLNES** *num_of_blank_lines* **ETXT_COLOR** *enter_text_color* **CTXT_COLOR** *choice_text_color RND NOWAIT MENU NOADD NOSHIFT USE_TRAPS INS_AT_END USE_COLORS*

field_list - f/c/e1,f/c/e2,...,f/c/ex - Required - The array fields, constants or expressions to be used in the list.

NOTE: You can separate the fields on the list by putting spaces between the fields. For example:

field1,' ',field2,' ',field3

enter_udf - udf - Optional - If you specify this option the program will execute the UDF each time the user presses the ENTER or RETURN key (Change or Add a record), the DEL key (Delete a record) or INS (Insert a record). The type of entry and the location of the data are passed to the routine through the **ETYP()** function and the value in **counter_field**.

search_udf - udf - Optional - If this is specified the program will execute this UDF if the user presses the F2 key. You can then reset the **counter_field** value in the routine and return .T. if you want to reset the display to a specific element, or .F. if you want to leave everything as it is.

chse_expr - expression - Optional - Through the use of this option you can specify the field to be used as the choice field for this command. The program will use the first character of the field returned by this expression. In general you would use this to point to a different field to be checked when the user enters a character from the keyboard. For example, you may want to choose one of two fields depending on other values. You could set an F or P type pointer to point to field one. Then if the user presses the F3 key, for example, your udf could change the pointer so that it points to field two. In both cases this option would look like:

CHSE *pointer_fld*

where *pointer_fld* is the F or P type pointer. Now, if the user presses a key (to choose a specific line in the list) the program will use the field being pointed to instead of just using the first field in the list.

help_udf - udf - Optional - This is the UDF to execute if the user presses the F1 key.

NOTE: The **counter_field** value is constantly updated by the program. This way you can use that value in a help message if desired and tailor the response to the particular element number currently being processed.

other_udf - udf - Optional - This UDF is called if the user presses the F3 key. If you return .T. from this UDF, the program will assume that major changes have been made to the array and will restart the list command. This will allow you to resort the array into a different order and list it again without having to completely exit from the command.

number_of_active_elements - f/c/e - Required - The number of elements in the array(s) that have live data. Or, the total number of lines in the list.

max_number_of_elements - f/c/e - Optional - The maximum number of elements allowed in the smallest array. This is necessary only if you are using the **enter_udf** option.

NOTE: This must be at least 1 more than the **number_of_active_elements** or the program will not execute the **enter_udf**.

counter_field - fn/v - Required - This is an I type field that will be used for passing the current array number to your program. If the *MENU* option is specified then this is the field that will be used for returning the line (element) number when the user presses the RETURN or ENTER key.

NOTE: If the user presses the ESC key this field is set to 0.

move_udf - udf - Optional - This UDF is called each time the user presses any key while in the **LISTM** command. The **counter_field** is set to the proper value before this UDF is executed. The program expects no return and this is for your use only.

choice_color - f/c/e - Optional - The color to use to mark the ‘active’ line. If this isn’t specified the program will use the reverse color value.

search_type - f/c/e - Optional - If you are using the FAST SEARCH[tm] process and the value being searched for is not an A type field then you need to specify the field type with this option.

first_line_fields - f/c/e,f/c/e,...,f/c/e - Optional - Each of these fields contains a title line previously set up with the **SETLINE** command.

chrs_btwn_fields - f/c/e - Optional - This character string will be inserted at the end of each field in the list and before the next. If you allow the user to scroll left and right, you should use this option instead of just inserting a constant between each field yourself. By using this option, the program will know that these are additional fields and will not stop at them when shifting to the left or right.

num_of_blank_lines - f/c/e - Optional - The number of blank lines to keep between the top of the window box and the first line in the list.

enter_text_color - ***Windows Only*** - f/c/e - Optional - When a user enters characters to search for a specific entry the matching characters are generally displayed in an offsetting color. This is the text color for those characters. If this value is not specified the program will use black. For more information on colors in Windows programs please refer to **Chapter 11 - Windows Programming**.

choice_text_color - ***Windows Only*** - f/c/e - Optional - This is the text color for the choice (or highlight) bar. If this value is not specified the program will use black. For more information on colors in Windows programs please refer to **Chapter 11 - Windows Programming**.

RND - Optional - If this is included in the command the program will know that the elements are not in alphabetic order and will loop through the entire list when searching for a specific value.

NOWAIT - Optional - If this is included in the command the program will display the lines but will immediately continue with the next command. This is useful when you want to display the choices available but you want to have the user start with a different process.

MENU - Optional - If this is included in the command the program will exit the routine after the user presses the RETURN or ENTER key. The current element number will be in the **counter_field**. This allows you to use this command as though it were a very large menu. If the user presses the ESC key the **counter_field** will be set to 0.

NOADD - Optional - If this is included in the command the user will not be able to add new values to the array but will still be able to change or delete them. This is effective only if the **enter_udf** option has been included.

NOSHIFT - Optional - Use this option to prevent the user from shifting or scrolling right or left. This might be the case where you are keeping more data than you want the user to access. The default is to allow the user to scroll right and left.

USE_TRAPS - Optional - If this option is set the program will look to 'outside' traps that are set to something other than Default or Ignore before checking for operations within this command.

INS_AT_END - Optional - If this option is set then the program will treat all insert operations at though the user had gone to the end of the list, moved to a new line, and pressed the ENTER key.

COMMENTS

You need to specify a window before this command or the routine will take the entire screen. Also, the command uses 2 columns of that window to provide up and down arrow indicators and the line between the arrows and the list data.

All traps, other than F1, F2 and F3 are still available to you and can be set before executing this command. The **counter_field** value is not cleared if one of these traps is executed. You can safely use a **TRAP** to exit from the routine since all housekeeping is done before the **TRAP** is run. If you wish to return to this routine from a trap routine do a **GOSUB** in the **TRAP** with a standard **RET**.

If you have included an **enter_udf**, the following applies:

1) When returning from the UDF, return .T. if you wish to save the changes made to the array element; return .F. if you don't.

2) Do not save the array element yourself in the UDF. The command will save it for you. This also applies to inserting a new element.

3) If you are allowing the user to delete an element and wish to do so, return from the appropriate UDF with a .T.; otherwise return with a .F. and nothing will change. Again, don't delete the element in the UDF; the command will do so for you.

4) If you have other arrays/files that are also being accessed along with the main array you do have to make any appropriate changes to those arrays/files that are necessary. The command will only affect the main array/file.

5) You can also have the program affect other array fields that are not displayed by including them in the list. Even if they can't be seen in the window the program will affect them all during an insert, delete, etc.

You must return to the command from any UDF called with a **RET .x.** command. If you wish to exit the command upon return, execute the **LIST_EXIT** command and the program will continue automatically to the next command upon return to the **LISTM**.

USER INTERFACE

To add a new line the user would move the cursor to the end of the list (END key, DN ARROW, PgDn, etc.) and the press the DOWN ARROW key once more so that the cursor is on a blank line. If the user presses the RETURN or ENTER key now the program will allow entry of a new set of values. As a shortcut, you can also press the ^END key (ctrl+END) to jump immediately to the entry of a new record. A line can be inserted at the cursor location by pressing the INS key. For this to happen you must have specified an **enter_udf** and have more elements available than active.

If the mouse is active, the user will be able to use the mouse to move through the list. Please see the detailed explanation under the **MOUSE CONTROL** in the Introduction of **Chapter 3, Main Menu Programs**.

PROGRAM EDITOR

User interface -> Lists -> Array list

SAMPLE PROGRAM

ARAYLIST

SEE ALSO

LISTF, LIST_EXIT, AUTOINC, NOREDSP, NORSTRT

LIST EXIT

This is used in connection with the **LISTM** (List Array) and **LISTF** (List File) commands. If you are executing one of those commands and are in a routine 'outside' of the command (e.g., an enter process), you can use this command to terminate the **LISTF** or **LISTM** command when you return to it.

LIST_EXIT

No options

PROGRAM EDITOR User Interface -> Lists -> Exit list

SEE ALSO LISTF, LISTM, AUTOINC, NOREDSP, NORSTRT

LIST FILE

This command will generate a list of records directly from a file. The user can scroll through the list using the UP and DOWN ARROW keys, PgUp and PgDn, or enter a single character, and depending on your options, find a value in the list. You can also display lines wider than the window; the user shifts the display right or left using the RIGHT or LEFT ARROW keys.

All of the lookup windows in the TAS Professional 5.1 utilities use either this command or the **LISTM** (List Array) command.

LISTF *field_list* **ENTER** *enter_udf* **SRCH** *search_udf* **CHSE** *chse_expr* **HELP** *help_udf*
OTH *other_udf* **FILE** *filename/@file_number* **KEY** *keyname/@key_number*
START *start_value* **FOR** *for_filter_expression* **WHILE** *while_filter_expression*
ON_MOVE *move_udf* **ECOLOR** *enter_color* **CCOLOR** *choice_color* **STYP** *search_type*
FLINE *first_line_fields* **CBF** *chrs_btwn_fields* **BLNES** *num_of_blank_lines*
ETXT_COLOR *enter_text_color* **CTXT_COLOR** *choice_text_color RND NOWAIT MENU NOADD UP NOSHIFT END USE_TRAPS INS_AT_END USE_COLORS*

field_list - f/c/e1,f/c/e2,...,f/c/ex - Required - The file fields, constants or expressions to be used in the list. The fields can be from other files (generally related).

NOTE: You can separate the fields on the list by putting spaces between the fields. For example:

field1,' ',field2,' ',field3

enter_udf - udf - Optional - If you specify this option the program will execute the UDF each time the user presses the ENTER or RETURN key (Change or Add a record), the DEL key (Delete a record) or INS (Insert a record). The type of entry is passed to the routine through the **ETYP()** function. The record is active in the normal record buffer and can be accessed using the standard field names.

search_udf - udf - Optional - If this is specified the program will execute this UDF if the user presses the F2 key. If you want to reset the display to a specific record then leave that record active and return a .T. value; return .F. if you want to leave everything as it is.

chse_expr - expression - Optional - Using this option you can specify the field to be used as the choice field for this command. The program will use the first character of the field

returned by this expression. In general you would use this to point to a different field to be checked when the user enters a character from the keyboard. For example, you may want to choose one of two fields depending on other values. You could set an F or P type pointer to point to field one. Then if the user presses the F3 key, for example, your udf could change the pointer so that it points to field two. In both cases this option would look like:

CHSE *pointer_fld*

where *pointer_fld* is the F or P type pointer. Now, if the user presses a key (to choose a specific line in the list), the program will use the field being pointed to instead of just using the first field in the list.

If an index (key) is being used this option is ignored since the program uses multiple characters entered to search for a specific record.

help_udf - udf - Optional - This is the UDF to execute if the user presses the F1 key.

other_udf - udf - Optional - This UDF is called if the user presses the F3 key. If you return .T. from this UDF the program will assume that changes have been made to one of the options in the command (such as the key specifier) and will restart the list command. This will allow you to change the key being used and list the data again without having to completely exit from the command. Through the use of the **flist** (see the information at the beginning of this chapter) option and an array of F pointers, you could even change the fields being listed or the order in which they appear.

filename/@file_number - file_expr - Optional - The name or number of the file to be used. If you do not include this option, the program will look for a default search file set in the **SRCH** (Search File) command. If a default file hasn't been previously set the program will report an error, and the command will be skipped. This entry will override any default value set in the **SRCH** command.

keyname/@key_number - key_expr - Optional - Set this to the appropriate value to read the records in some specific order. If you do not include this option, the program will look for a default key set in the **SRCH** (Search File) command. If a default key hasn't been previously set the program will report an error and the command will be skipped. This entry will override any default value set in the **SRCH** command.

start_value - f/c/e1,f/c/e2,...,f/c/ex - Optional - The value to use as the beginning record. This is similar to doing a **FIND** before the command. If the index you're searching on has multiple segments you may list the proper values for the segments, separating them with commas. This will allow you to specify the exact type for each of the segments. For example, suppose you're searching on an index that has two segments: a 5 character alpha and an I type field. Each record you want has the same alpha but the I field will change, and is in order. In the first record the I type field has a value of 0. The following would find the first record for that group, if it exists:

start 'ABCDE',0!

Notice that we separate the values with a comma. The only requirement is that the values be of the same type (and size) as the segments in the key. In this case we put the I type constant specifier (!) after the 0 to make sure that it is passed as an I type field.

for_filter_expression - lexpr - Optional - You would use this option to restrict the records listed in this command. If the expression did not resolve to .T., the record would not

be listed. In this option, the search continues until the program reaches the end of file and then will quit.

while_filter_expression - lexpr - Optional - This option also restricts the records listed. However, the first time the expression resolves to .F., the program stops reading records and continues to the next command. Therefore, if the program is reading records in a certain order (by a specific key) and the expression returns .F., then the program knows that all the appropriate records have been read and there is no need to continue reading. For example: Suppose you want to export all the invoice records for a specific customer. The first record for that customer is found in the invoice file either by using the **start_value** option within this command, or through the **FIND** command before executing this command. Then the **while_filter_expression** would be:

INV_CUST_CODE = CUSTOMER_CODE

(This assumes that there is a field called INV_CUST_CODE in the invoice file and a similar field in the customer file called CUSTOMER_CODE. The **keyname/ @key_number** option must be set to the proper value so that the records are read in CUSTOMER_CODE order, or you must have executed the **SRCH** command previous to this command.) When the first invoice record for the next customer is read the program will stop reading records.

NOTE: If you use this option you must use the **start_value**. If you do not, the program will find the first record in the file and the **while_filter_expression** option will probably fail the first time. Also, the program won't display the lines properly when the user reaches the end or beginning of the list.

move_udf - udf - Optional - This UDF is called each time the user presses any key while in the **LISTF** command. The current record is in the normal buffer before this UDF is executed. The program expects no return and this is for your use only.

An example of the use of this option can be seen in the Maintain Data Dictionary program. When you move the cursor up or down the list the appropriate record is shown in full in the window above the list. We used this option to call a UDF that redisplays the appropriate information.

enter_color - f/c/e - Optional - If you specify a value here, the program will display the characters entered so far by the user in searching for a specific record. The color you choose will be used to highlight the entered characters and make searching easier.

choice_color - f/c/e - Optional - The color to use to mark the 'active' line. If this isn't specified the program will use the reverse color value.

search_type - f/c/e - Optional - If you are using the FAST SEARCH[tm] process and the value being searched for is not an A type field, then you need to specify the field type with this option.

first_line_fields - f/c/e,f/c/e,...,f/c/e - Optional - Each of these fields contains a title line previously set up with the **SETLINE** command.

chrs_btwn_fields - f/c/e - Optional - This character string will be inserted at the end of each field in the list and before the next. If you allow the user to scroll left and right, you should use this option instead of just inserting a constant between each field yourself. By using this option, the program will know that these are additional fields and will not stop at them when shifting to the left or right.

num_of_blank_lines - f/c/e - Optional - The number of blank lines to keep between the top of the window box and the first line in the list.

enter_text_color - ***Windows Only*** - f/c/e - Optional - When a user enters characters to search for a specific entry the matching characters are generally displayed in an offsetting color. This is the text color for those characters. If this value is not specified the program will use black. For more information on colors in Windows programs please refer to **Chapter 11 - Windows Programming**.

choice_text_color - ***Windows Only*** - f/c/e - Optional - This is the text color for the choice (or highlight) bar. If this value is not specified the program will use black. For more information on colors in Windows programs please refer to **Chapter 11 - Windows Programming**.

RND - Optional - If this is included in the command, the program will know that the records are not in alphabetic order and will loop through the entire list when searching for a specific value. Do not set this option if you want the user to be able to find a record by entering the appropriate characters. See the COMMENTS below.

NOWAIT - Optional - If this is included in the command the program will display the lines but will immediately continue with the next command. This is useful when you want to display the choices available but you want to have the user start with a different process.

MENU - Optional - If this is included in the command the program will exit the routine after the user presses the RETURN or ENTER key. The current record will be active. This allows you to use this command as though it were a very large menu. If the user presses the ESC key the file buffer will be cleared before the command exits back to the program.

NOADD - Optional - If this is included in the command the user will not be able to add new values to the array but will still be able to change or delete them. This is effective only if the **enter_udf** option has been included.

UP - Optional - If this is included in the command then all characters the user enters from the keyboard during searches will be set to upper case.

NOSHIFT - Optional - Use this option to prevent the user from shifting or scrolling right or left. This might be the case where you are keeping more data than you want the user to access. The default is to allow the user to scroll right and left.

END - Optional - Use this option when you want to start at the end of the file instead of the beginning. If there are enough records in the file to fill the list the last record will be on the last line with previous records above it. The cursor bar will be positioned on the last record. If there aren't enough records then all records in the file will be displayed with the cursor bar still placed on the last one. The key used will be the one specified in the KEY option or by the SRCH command.

USE_TRAPS - Optional - If this option is set the program will look to 'outside' traps that are set to something other than Default or Ignore before checking for operations within this command.

INS_AT_END - Optional - If this option is set then the program will treat all insert operations at though the user had gone to the end of the list, moved to a new line, and pressed the ENTER key.

COMMENTS

You need to specify a window before this command or the routine will take the entire screen. Also, the **LISTF** command uses 2 columns of that window to provide up and down arrow indicators and the line between the arrows and the list data.

All traps, other than F1, F2 and F3 are still available to you and can be set before executing this command. You can safely use a **TRAP** to exit from the routine since all housekeeping is done before the **TRAP** is run. If you wish to return to this routine from a trap routine do a **GOSUB** in the **TRAP** with a standard **RET**.

If you have included an **enter_udf**, the following applies:

1) When returning from the UDF return .T. if you wish to save the changes made to the record; return .F. if you don't.

2) Do not save the record yourself in the UDF. The command will save it for you. This also applies to inserting a new record.

3) If you are allowing the user to delete a record and wish to do so, return from the appropriate UDF with a .T.; otherwise return with a .F. and nothing will change. Again: don't delete the record in the UDF; the command will do so for you.

4) If you have other files that are also being accessed along with the main file you do have to make any appropriate changes to those files that are necessary. The command will only affect the main file.

You must return to the command from any UDF called with a **RET .x.** command. If you wish to exit the command upon return execute the **LIST_EXIT** command and the program will continue automatically to the next command upon return to the **LISTF**.

You may not access Text type files as part of a **LISTF**. However, you can access Fixed length record type files and dBASE III+ files.

If you are using record locking it will be temporarily turned off for any files that are a part of this command. This includes any related files. However, if the *MENU* flag is set, or a record is active for any other reason when exiting this command, it will be locked before the routine is terminated. This should also affect any related records. Effectively, you will not have to worry about records being locked while getting a listing, however. If you choose a record that is already locked, the appropriate error message will be displayed depending on the **RLCK TRAP**. Also, if you execute a trap other than F2 or F3, the current record, which would automatically be activated, will be locked, if appropriate. This only applies to normal TAS files. There is no record locking on non-TAS files.

USER INTERFACE

To add a new line the user would move the cursor to the end of the list (END key, DN ARROW, PgDn, etc.) and the press the DOWN ARROW key once more so that the cursor is on a blank line. If the user presses the RETURN or ENTER key now the program will allow entry of a new record. As a shortcut, you can also press the ^END key (ctrl+END) to jump immediately to the entry of a new record. A line can be inserted at the cursor location by pressing the INS key. For this to happen you must have specified an **enter_udf.**

If the file being listed is a standard TAS Pro 5.1 (or other Btrieve) file and a key has been specified (not @0) the user may search for a record in the list by entering the appropriate characters. As each character is entered the program will find the record that matches those characters. At the first non-match the program will sound the bell and the searching will stop. If the user presses the BACKSPACE key, the program will find the previous record that matched those characters. The user can then enter a new

character or continue to press the BACKSPACE key until the first record in the list is again reached. Also, they can press the ^U key and the program will return to the first record in the list irregardless of the current location or the characters entered.

If the mouse is active, the user will be able to use the mouse to move through the list. Please see the detailed explanation under the **MOUSE CONTROL** in the Introduction of **Chapter 3, Main Menu Programs**.

PROGRAM EDITOR User interface -> Lists -> File list

SAMPLE PROGRAM FILELIST, LIST_TST.SLT/.SRC

SEE ALSO LISTM, LIST_EXIT, AUTOINC, AUTODEC, AUTOENTER, NOREDSP, NORSTRT

LOAD PRINTER DRIVER *(DOS Only)*

This command will allow you to change the current .CTL file from within a program.

LD_PDRV *ctl_file_name*

ctl_file_name - f/c/e - Required - This is the name of the driver file. It must be in the same directory as TPC32.EXE and have the extension .CTL. All you need to provide is up to 8 characters of file name.

COMMENTS

This command, and the **PRT_NUM** (Printer Number) command, give you great flexibility in choosing which printer and what type of printer you are going to use for any type of output. Any changes made with this command are not permanent. If the user is part of a network the driver for just the one work-station is changed, not the entire network.

PROGRAM EDITOR Reports -> Printer -> Driver

LOOP

This is a process control command, i.e., it will loop back to the beginning in a **WHILE/ENDW** loop. This is one part of a complete command structure.

LOOP

No options

COMMENTS

For more information on the different structured programming commands, and the **LOOP** command in particular, please see **Chapter 7, Structured Programming Commands.**

PROGRAM EDITOR Prg control -> While -> Loop

SEE ALSO WHILE, LOOP_IF, EXIT, EXIT_IF, ENDW

LOOP_IF

This is a process control command, i.e., it will control whether to loop back to the beginning in a **WHILE/ENDW** loop. This is one part of a complete command structure.

LOOP_IF *expression*

expression - lexpr - Required - If the expression resolves to .T. then the program will transfer control to the appropriate **WHILE** line. This would be the equivalent of executing the **ENDW** command.

COMMENTS

For more information on the different structured programming commands, and the **LOOP_IF** command in particular, please see **Chapter 7, Structured Programming Commands.**

PROGRAM EDITOR Prg control -> While -> loop_If

SEE ALSO WHILE, LOOP, EXIT, EXIT_IF, ENDW

MEMORY POINTER UPDATE (*DOS only*)

This is a TAS Professional 3.0 command here for compatibility. There is no preferred TAS Professional 5.1 command/function.

MEM_PTR *mem_area#* **START** *start_val* **OCCURS** *#of_chrs_before_next* **SIZE** *size_of_ptr* **VAL** *chg_value* **TIMES** *#_ptrs_to_chg* **DO***what_to_do*

mem_area# - f/c/e - Required - This is the memory area to use. This must resolve to a value of 1 through 4.

start_val - f/c/e - Required - The character position of the first pointer to be changed. The first character in the memory area is at position 1.

#_of_chrs_before_next - f/c/e - Required - The total number of characters before the next pointer. If there are a total of 10 characters per element in the memory area and the pointer is part of that element then this would be 10.

size_of_ptr - f/c/e - Required - This is the internal or csize of the pointer itself. If the pointer is O type (old form BCD) and has a display size of 5 then this would be 3 (5/2+rounding).

chg_value - f/c/e - Required - The amount to add to or subtract from each pointer. NOTE: The size and type of this field must be the same as the pointer.

#_ptrs_to_chg - f/c/e - Required - How many pointers do you want to change.

what_to_do - f/c/e - Required - This must resolve to either A (add) or S (subtract).

COMMENTS

This is the equivalent of the TAS Professional 3.0 command Memory Pointer Update.

PROGRAM EDITOR 3.0 Commands -> Mem ptr

MEMORY SPACE UPDATE (*DOS only*)

This is a TAS Professional 3.0 command here for compatibility. There is no preferred TAS Professional 5.1 command/function.

MEM_SPC *mem_area#* **START** *start_val* **STOP** *stop_val* **NCHR** *#_of_chrs* **DO***what_to_do*

mem_area# - f/c/e - Required - This is the memory area to use. This must resolve to a value of 1 through 4.

start_val - f/c/e - Required - The character position of the beginning of the block of characters. The first character in the memory area is at position 1.

stop_val - f/c/e - Required - The character position of the end of the block of characters.

num_of_chrs - f/c/e - Required - Number of characters to add or subtract.

what_to_do - f/c/e - Required - This must resolve to either A (add) or S (subtract).

COMMENTS

This is the equivalent of the TAS Professional 3.0 command Memory Space Update.

PROGRAM EDITOR 3.0 Commands -> Mem space

MENU

Display a bar-type menu and prompt for a valid choice. The **NMENU** command is an alternative for accomplishing the same thing.

MENU AT *starting_column,starting_row* **LEN** *length* **WDT** *width*
CPC *choices_per_column* **FLD** *menu_lines_field* **CNTR** *counter_field*
NCH *maximum_number_of_choices* **MCW** *choice_print_width*
ESC *esc_key_label* **HELP** *help_label* **TTL** *menu_title HOLD*

starting_column - f/c/e - Required - Upper left corner of the menu, column value.

starting_row - f/c/e - Required - Upper left corner of the menu, row value.

length - f/c/e - Required - Number of rows long, including box around the menu. If there are two lines within the menu the **length** is 4.

width - f/c/e - Required - Number of columns wide, including box around the menu. If the choices are 10 columns wide within the menu the total **width** is 12.

choices_per_column - f/c/e - Required - If there are multiple columns in the menu this is the number of choices within a single column.

menu_lines_field - fn/v - Required - The array field that contains the menu lines.

Each menu line is made up of 2 parts. The first is the displayed portion; this is what your user will see. The second is the control information. This consists of 5 characters. The first two characters indicate the starting column value for this line within the

menu, the next two indicate the row value for this line within the menu, and the last character is the choice character. If the user presses a key that matches this character the command will act as though the user moved the cursor to that particular line and pressed the ENTER or RETURN key.

Regarding the column and row specifiers: these are not the same values as those set for the menu as a whole. Imagine that the menu is actually a blank slate and you can put options any where you want on that slate. That is basically how the process works. This allows you to leave spaces between options or to make multiple columns. Each time the user presses the DOWN ARROW key the next element is chosen and the routine goes to the correct location within the menu. Pressing UP ARROW causes the previous element to be used.

counter_field - fn/v - Required - The field that will contain the line number the user chooses. Must be of type I.

maximum_number_of_choices - f/c/e - Required - The total number of values that could be chosen in the menu.

choice_print_width - f/c/e - Required - The width in number of characters of the displayable portion of the menu choice line. All lines need to be of the same width.

esc_key_label - label - Optional - Where to go if the user presses the ESC key. If this option is not set the program will exit the command, set the **counter_field** to 0, and execute the next command in the program.

help_label - label - Optional - What subroutine to **GOSUB** to if the user presses the F1 key. You must **RET** from that routine.

menu_title - f/c/e - Optional - A title to display on the top line of the menu. Will print in the same space the box would normally use so you don't lose any space in the menu.

HOLD - Optional - If this option is included the menu will remain on the screen after the user's choice has been made. Normally, it would be automatically cleared.

COMMENTS

For those familiar with TAS Professional 3.0 this is the old MENU command with a couple of changes. The most important is that all commands which make a box of some kind on the screen (**MENU**, **WINDOW**, etc.) now start at the upper left hand corner and go down instead of lower left hand corner and go up.

NOTE: The most important element in the entire command is to make sure that all of your values are in the proper location within the array element line. If you stipulate that the **choice_print_width** is 10 then the first character of the column coordinate for each line must start at position 11 within the array element line. Each column coordinate must be 2 characters, zero padded if necessary. Each row coordinate must immediately follow the column value and also be 2 characters, zero padded. And immediately following the row value must be the single choice character. It should be in upper case since the program will automatically convert to upper case the entry made by the user.

For example:

```
"G - General Ledger   0101G"
"R - A/R & Invoicing  0102R"
"A - A/P & P/O        0103P"
"I - Inventory Control0104I"
```

Each one of the lines above would be an element in the array. The **choice_print_width** should be 21, and each column value starts at the 22nd position within the element. Also note the zero padding. If you have trouble with a menu not appearing correctly it is probably due to the column and row coordinates not being where the program expects them.

PROGRAM EDITOR User interface -> 3.0 menu

SEE ALSO NMENU

MESSAGE

Use this command to display a message at the standard message location.

MSG *expression* **WINDOWS** *icon_type NOWAIT*

expression - *f/c/e* - Required - You may use this in two ways. The **expression** can be set as an A type field and the program will display the field as the message. It can also be set as a numeric value (any appropriate type) and the program will read the appropriate record in the ERRMSG.B file. The value set by you must relate to the ERROR_NUM field in the ERRMSG record. For more information please see **Chapter 3, Main Menu - Utilities**.

icon_type - ***Windows Only*** - f/c/e - Optional - If you want to display a standard Windows information or warning dialog box you can use this option. By setting this value to 'info' or 'warn' (note: only the first character matters) the program will display a dialog box with an information icon or with a warning icon, the message and an OK button at the bottom of the box. The program will stop until the user clicks on the button or presses the ENTER key. If you use this option you must supply the message expression. It cannot be retrieved from the ERRMSG.B file. Also, in a normal **MSG** command you are limited to 320 characters. That limit does not apply when this option is used. If this option is specified the **NOWAIT** option is ignored.

NOWAIT - Optional - If this option is included in the command the message will be displayed but it will not wait for the user to press a key to continue as would be normal. The message will remain displayed on the screen. It is then up to you to either **REDSP** (Redisplay) a previously saved screen or **CLRSCR** (Clear Screen) to remove the message.

COMMENTS

By using the ERRMSG.B file as the repository of any messages (e.g., help messages), you can easily add, change, delete, etc. any appropriate message. You should start any message numbers at 5000 or more. This will help make certain that no messages will have to change due to interference with records written by Business Tools and provided with the system. Obviously, the ERROR_NUM values don't have to be sequential.

NOTE: In a normal MSG command (where the WINDOWS option is not specified) you are limited to 320 characters total for any message.

PROGRAM EDITOR User interface -> Messages -> Disp message

SAMPLE PROGRAM WINTEST ***(Windows Only)***

MID

This command will either overwrite a field or insert a field into another field.

MID *receiving_field* **START** *starting_character* **NCHR** *number_of_characters*
FLD *characters_to_add* **MEM_AREA** *mem_area# INS*

receiving_field - fn/v - Required - The receiving field. Must be of type A.

starting_character - f/c/e - Optional - Where to start adding the new characters. The first character in the **receiving_field** is character 1. If no value is given the value will default to 1.

number_of_characters - f/c/e - Optional - The number of characters to add. If no value is given the program will use the internal size of the **characters_to_add** field.

characters_to_add - f/c/e - Required - The characters that will be put into the receiving field. Any type of field may be used; however, the field is not converted to type A. It will be added in its natural form.

mem_area# - f/c/e - Optional - If you are actually getting the characters from a memory area then this is the number (from 1-4). NOTE: Even if you want to use a memory area you must still specify a **characters_to_add** field, although it will be ignored during execution of the command.

INS - Optional - If this option is specified the program will insert the characters, moving the current characters to the right starting with the character at the start location. The characters at the end of the field, if any, will be deleted. The default is to overwrite the characters at the current location.

PROGRAM EDITOR Field -> alpha Fld cmds -> Mid (stuff)

SAMPLE PROGRAM P1

MOUNT

This command will 'mount', or set active, a screen or report format compiled in the program. There can be only one screen or report format active at the same time.

MOUNT *format_name* **TYPE** *format_type* **PTO** *print_to* **PFN** *print_to_file* **EXTERN**

format_name - sac - Required - The name of the format to mount. This name follows the standard file name requirements. The compiler tries to find a file with the format name in the same path as the original source file. The extension defaults to .FMT, although you may specify a different one if desired.

format_type - SR - Required - There are two possible format types, **S** - screen and **R** - report.

The *SCREEN* format is used to define location and type of field and constant data. When mounted, as much as possible of the entire body of the format will be displayed in the current window. The fields are put in a buffer to be used in conjunction with the **ENTER** command. Then if a field is referenced in the active screen format, you need not use the column/row specifiers in the **ENTER** command; the program will know where it is. This also applies when a record is found. If fields in the mounted

screen format are a part of the record found the values will be automatically displayed at the appropriate locations. No other programming is required.

The *REPORT* format is used in conjunction with the **PFMT** (Print Format) command. As opposed to the screen format the report format is treated as a set of individual lines. However, the same basic process applies. When you print a format line the program not only uses the constant characters on the format line, but also the current data of the fields specified.

For more information about the layout of *SCREEN* or *REPORT* formats please see **Chapter 8, Screen/Report Format Information**.

print_to - prt_where - Optional - If this is a report format, use this option to direct the output of the report. If no **print_to** option is set the program sets it automatically to **S** (for screen).

print_to_file - *f/c/e* - Optional - If you set the **print_to** option to **D** the name of the file to use may be specified with this option. If the file name is not specified the program will ask for one at runtime.

EXTERN - Optional - If this format is not included in the source code file then use this option. This tells the compiler to look for a separate file in the same sub-directory as the main source code file. You can also specify the extension as part of the format name, e.g., MAIN_SCR.TT1. If no extension is specified the compiler will add the default value depending on the type of format being mounted.

COMMENTS

Each time you specify a **MOUNT** format in your program you also create a special buffer for the fields on that format. This buffer is maintained in memory while the program is running. If you need to **MOUNT** another screen format during program execution and then return to the original screen, you should execute the **SAVES** (Save Screen) command, which not only saves the screen image but also saves the information about the screen fields. Then **MOUNT** the new screen format, do whatever needs to be done, and when you **REDSP** (Redisplay) the previous screen, your fields will be active again without the need to **REMOUNT** the format.

For example, you might have the customer master screen displayed and need to allow entry to a subsection of a record not on the screen or from another file. You would:

saves
mount subsection_format **type** *s*
..do appropriate entries..
redsp

and then the original screen, the one displayed before the **SAVES** command, would be active again.

If you have fields as part of a screen or report format that are also part of a file make sure that file is opened before the format is mounted. If you don't, you will get an error stating that the field was not allocated, and is probably part of a file that has not been opened yet or has been removed from memory, and the program will cease executing.

PROGRAM EDITOR

User interface -> Screen control -> Mount

SAMPLE PROGRAM

RPTFMT

SEE ALSO

SAVES, REDSP, SAY, CLRSF

MOUSE (*DOS only*)

This command turns the mouse on and off.

MOUSE *ON/OFF*

ON/OFF - Required - If this option is *ON* the program will display the mouse on the screen and allow the user to take advantage of the features relating to the mouse. To return to normal mode the option would be *OFF*.

COMMENTS

Other commands within TAS Professional 5.1 will take advantage of the mouse if it has been turned on. With some commands this is more or less automatic; with others you have to set up some supporting mouse commands. The various traps that work with the mouse (e.g., for detecting movement) are covered under the **TRAP** command. Current mouse location is available using two functions and there is also a function for determining what the mouse is doing. See the documentation in **Chapter 6, Function Reference** for more information on **MCOL()**, **MROW()** and **MOUSE_ACT()**.

To turn the mouse on automatically you can set the Mouse On value to Y in the Set Configuration program (Utilities-G). Please refer to the Set Configuration documentation in **Chapter 3, Main Menu Programs**. If you have that value set to Y you do not need to use this command.

If there is no mouse connected to the computer this command will be ignored.

USER INTERFACE

Use of the mouse is fully integrated with the **LISTF**, **LISTM** and **MENU** commands automatically. For more information about this please refer to the Introduction - Mouse Control in **Chapter 3, Main Menu Programs** .

PROGRAM EDITOR System -> Mouse

SAMPLE PROGRAM MOUSETST

MOVE DATA IN MEMORY

This command will move a specified number of characters from one location to another without concern to type or individual field sizes.

XFER FROM *from_field* **TO** *to_field* **NCHR** *number_of_characters* **FMEM** *from_mem_area#* **TMEM** *to_mem_area#*

from_field - fn/v - Required - The **from** field.

to_field - fn/v - Required - The **to** field or recipient.

number_of_characters - f/c/e - Required - The number of characters to move.

from_mem_area# - f/c/e - Optional - The memory area number to transfer from. This must resolve to a value from 1 through 4. If you specify this value then the **from_field** is evaluated as the offset within this area.

to_mem_area# - f/c/e - Optional - The memory area number to transfer to. This must resolve to a value from 1 through 4. If you specify this value then the **to_field** is evaluated as the offset within this area.

COMMENTS

This command does not convert any data but just moves raw characters from one location to another without regard for the type. You must specify either fields or memory areas to use.

You may use P and F type pointers in this command.

NOTE: Don't forget to calculate **number_of_characters** based on internal size and not display size.

PROGRAM EDITOR

Field -> Move data

NEXT

This is the end point of the process control structure **FOR/NEXT**.

NEXT

No options

COMMENTS

For more information on the different structured programming commands, and the **NEXT** command in particular, please see **Chapter 7, Structured Programming Commands.**

PROGRAM EDITOR

Prg control -> For -> Next

SEE ALSO

FOR, FLOOP, FLOOP_IF, FEXIT, FEXIT_IF

NMENU

Create a pull down menu and allow the user to make a choice. The choice is returned to the program. This is an alternative to the **MENU** command.

NMENU *lines* **AT** *starting_column,starting_row* **LEN** *length* **WDT** *width ARRAY*
CNTR *counter_field* **NCH** *number_of_choices* **CHXPR** *choice_expression*
BOX *box* **BCOLOR** *box_color* **MCOLOR** *menu_color*
CCOLOR *choice_color* **SHD** *shadow_where* **SCOLOR** *shadow_color*
TTLW *title_where* **TTL** *title* **HELP** *help_udf* **LTA** *left_arrow_label*
RTA *right_arrow_label* **MTXT_COLOR** *menu_text_color*
CTXT_COLOR *choice_text_color HOLD AUTO NOWAIT USE_COLORS*

lines - f/c/e1, f/c/e2,..., f/c/ex - Required - The fields/constants that make up the menu items. A menu can have up to 20 different items. You can also put single (—) or double (==) dividing lines between items. The user will not be able to choose these items and they will be ignored when returning the **counter_field** value. For example:

'Choice1','Choice2','—','Choice3','Choice4'.....

will display (box not shown):

Choice1
Choice2
———
Choice3
Choice4

You can also specify your own dividing lines; however, the first two characters must be single/double dashed lines (and they will be displayed). If the user chooses 'Choice3' the program would return 3 in **counter_field**. This has nothing to do with the name (Choice3) but only reflects the fact that Choice 3 is the 3rd choice in the list, ignoring the single dashed line. If the user presses the Dn Arrow key the 'cursor' will skip the single dashed line.

This can also be an array field reference. If it is, the array option must be **Y**. All the same rules apply; the only difference is that there is only one field reference for this option instead of a list. The list is actually the array elements.

starting_column - f/c/e - Required - Upper left corner of the menu, column value.

starting_row - f/c/e - Required - Upper left corner of the menu, row value.

length - f/c/e - Required - Number of rows long, including box around the menu. If there are two lines within the menu the **length** is 4.

width - f/c/e - Required - Number of columns wide, including box around the menu. If the choices are 10 columns wide within the menu the total **width** is 12.

ARRAY - Optional - If the lines field is actually an array field this must be included in the command..

counter_field - fn/v - Required - This is an I type field that will be used for passing the user's choice to your program.

NOTE: If the user presses the ESC key this field is set to 0.

number_of_choices - f/c/e - Required - How many choices the user has. This is only required if the *ARRAY* option is specified.

choice_expression - f/c/e - Optional - The user may choose the appropriate menu item by entering a single character. The default method is the first capitalized character of each item. You can specify an expression which will return a character that will be compared to that entered by the user. The program will move the cursor to the appropriate line. If not found the bell will sound.

box - f/c/e - Optional - The box type to use for defining the edge of the menu. If this is specified you need to take into consideration the 2 extra rows and columns that it requires when determining **length**, **width**, and location. **S** specifies a single line box, **D** is double, and **C** is custom (dependent on what you have set in TAS50.OVL).

box_color - f/c/e - Optional - If you have specified some sort of **box** you may also specify the color for it. If this isn't set the program will use the **menu_color** as this value also.

menu_color - f/c/e - Optional - The color value for the main body of the menu. If nothing is specified the program will use the current normal color.

choice_color - f/c/e - Optional - The color used to designate the current cursor location and the first capitalized character in each choice line. If not specified the program uses the reverse color value.

NOTE: If you have specified a **choice_expression** the program will not automatically color the first capitalized character in each line. It assumes that you have marked the appropriate character in some other manner yourself.

shadow_where - f/c/e - Optional - You may specify that a 'shadow' is to be displayed behind the menu. The **shadow_where** options are **R** - right, **L** - left or **N** - none. The shadow is displayed two characters to the right or left, and one line down from the menu. If you specify the appropriate **shadow_color** value, the resulting display will give the menu a 3-D feel. The default value is **N**.

shadow_color - f/c/e - Optional - If you have specified **L** or **R** for the **shadow_where** option you may also specify the color to be used.

title_where - f/c/e - Optional - If you wish to put a title message at the top of the menu you may choose where it will be. **L** is left side of menu, **R** right, **C** is centered and **N** is none. The default is **N**.

title - f/c/e - Optional - If you specified a **title_where** value other than **N** you use this as the value to be displayed.

help_udf - udf - Optional - This is the UDF to execute if the user presses the F1 key.

NOTE: The **counter_field** value is constantly updated by the program. This way you can use that value in a help message if desired and tailor the response to the current line or situation.

left_arrow_label - label - Optional - Where to transfer control if the user presses the left arrow key.

right_arrow_label - label - Optional - Where to transfer control if the user presses the right arrow key.

menu_text_color - ***Windows Only*** - f/c/e - Optional - The text color for the menu characters. If this value is not specified the program will use black. For more information on colors in Windows programs please refer to **Chapter 11 - Windows Programming**.

choice_text_color - ***Windows Only*** - f/c/e - Optional - This is the text color for the choice (or highlight) bar. If this value is not specified the program will use black. For more information on colors in Windows programs please refer to **Chapter 11 - Windows Programming**.

HOLD - Optional - If this is included in the command the program will not automatically clear the menu upon leaving this command. This will allow you to 'stack' menus if desired.

AUTO - ***DOS only*** - Optional - If this is included in the command the program will exit the menu automatically when the user presses a key that matches one of the possible choice characters. Normally, the cursor would move to that line and the program would wait for him/her to press the RETURN or ENTER key.

NOWAIT - ***DOS Only*** - Optional - If this is included in the command the program will display the menu but not wait for any input. Should be used in conjunction with the *HOLD*

option or the menu will just flash on the screen and the user will wonder what happened.

USE_COLORS - ***Windows Only*** - Optional - In a normal windows program the colors specified in a command are ignored and the program uses the colors in the TP5WIN.INI file. The colors in the TP5WIN.INI file become the default colors.. This is due to the major differences between DOS and Windows colors. If you use this option the program will look to the colors specified in the command first and, if they are 0, will then use the colors in the TP5WIN.INI file. For more information on colors in Windows please refer to **Chapter 11 - Windows Programming**.

PROGRAM EDITOR User interface -> 4.0 Nmenu

SAMPLE PROGRAMS MENU1, MENUTEST

NO REDISPLAY

Normally in the **LISTM** (List Array) or **LISTF** (List File) commands, when the program returns to the command from a UDF call, the screen displayed previous to the UDF call is redisplayed. In the case where you now have something new on the screen and don't wish for it to reappear this should be specified.

NOREDSP

No options

PROGRAM EDITOR User interface -> Lists -> No redisplay

SEE ALSO LISTF, LISTM, LIST_EXIT, AUTOINC, NORSTRT

NO RESTART

Normally in the **LISTF** (List File) commands when the program returns to the command from an **enter_udf** option the record just accessed becomes the active record at the top of the list. Use this command to keep this from happening.

NORSTRT

No options

PROGRAM EDITOR User interface -> Lists -> no reStart

SEE ALSO LISTF, LISTM, LIST_EXIT, AUTOINC, NOREDSP

NO VALID MESSAGE

This command will turn off the valid message that would normally appear during an **ENTER** command when the **vld** option returns .F.

NOVLDMSG

No options

COMMENTS

Normally in the **ENTER** command if the **vld** option is set and the expression returns .F., a **vldm** (valid message) is displayed, or if you have not provided one, a generalized message is displayed. In certain instances you might have multiple valid messages depending on other circumstances. In that situation you can display the appropriate message in the **vld** check itself and by using this command, a generalized message will not be displayed when the program returns to the **ENTER** command.

PROGRAM EDITOR User interface -> No Valid Msg

SEE ALSO ENTER

ON

Use this command to transfer control to a line label depending on a value. This is almost like a flat **CASE** command.

ON *expression GOTO/GOSUB line_label_list*

expression - f/c/e - Required - The value to test. Must resolve to a numeric value.

GOTO/GOSUB - Required - *GOTO* or *GOSUB* to the appropriate line.

line_label_list - label1,label2,...,labelx - Required - The list of line labels used to determine where to transfer control. The first label corresponds to a value of 1, the second to 2, etc. If the **expression** resolves to a value of 0 or greater than the number of labels -1 then control is transferred to the last label in the list. The line labels are separated by commas.

PROGRAM EDITOR prg Control -> Goto/gosub -> On

SEE ALSO GOSUB, GOTO

OPEN

This command will set up a file so that you may access data from it or write records to it from within your program. Use of this command forces you to refer to the file in every other command through the use of the file name as a special alpha constant (sac). However, this command is included to provide upward compatibility to TAS Professional 3.0 programs.

OPEN *filename* **EXT** *extension* **LOCK** *lock_type* **ERR** *error_label* **OWNER** *owner* **PATH** *path* **FD** *file_descriptor*

filename - sac - Required - A special alpha constant with a maximum of 8 characters that is the actual file name. No extension or path is given as part of the **filename**. For example:

BKARCUST

extension - f/c/e - Optional - If you wish to open a copy of the file with an **extension** something other than the current default value (.B or whatever the **company_code** value is set to), you may specify up to 3 characters here.

lock_type - NRFXA - Optional - The type of locking to use. **N** - none, **R** - record, **F** - file, **X** - fix, and **A** - accelerated access (this essentially locks the file). The default is **N** - none. By specifying **A** you can significantly speed up operations to Btrieve files as long as you are not concerned with recovering data in case of a failure and you are running on a single-user system. Specifying **X** tells the program to open the Btrieve file in read-only or FIX mode. You will then be able to read records in direct mode (i.e., without an index). This feature is also part of the Reindex and Restructure routines and should allow you to recover from some Btrieve errors without restoring the file from a backup.

error_label - label - Optional - Where to transfer control if an error occurs while attempting to open the file. If this is omitted you may check for an error opening the file through the use of the **FLERR(FNUM())** functions. See below. If you do not want to have the program display an error if the file can't be opened use the label name *NO_ERR*.

owner - f/c/e - Optional - A maximum 8 character code to be specified if the file is protected with the **OWNER** command.

path - f/c/e - Optional - You may specify the **path** for this file if it is not in the FILELOC.B file or if you want to access it somewhere else. This value will override any other path for the file.

NOTE: The path value must end with a backslash character. See the CHK_PATH_CHR() UDF in TASRTNS.LIB for an easy method of making sure that the last backslash is always there.

file_descriptor - f/c/e - Optional - You may specify the **file_descriptor** name for this file if it is not in FILELOC.B.

COMMENTS

Several commands and all functions require a file_number instead of a filename. Since this command doesn't supply a file_number there is a function that does: **FNUM()**. By giving it the filename as a standard constant (i.e., the name surrounded by quotes--"BKARCUST"), it will return the proper file_number. **FNUM()** can be 'embedded' in other functions so that the whole process is done at the same time. For example, the **FLERR()** function (file error) is probably the most widely used file command. It requires the file_number as its single option. The embedded FNUM() in the following example will provide the file_number for the file BKARCUST:

```
FLERR(FNUM("BKARCUST"))
```

The example shown above would return any file errors for the file BKARCUST.

If you open more than one file that uses the same FD the last file opened will have control in any **FIND** (match, first, last, etc. where you specify the key name instead of the file) command. If you wish to have the initial control set with a different file, either open it last or choose that specific file through the use of the **SETACT** (Set Active) command.

NOTE: The maximum number of files that may be open at one time with either the **OPEN** or **OPENV** command is 32. This limit applies across all programs that are currently active.

PROGRAM EDITOR

fiLe -> Open/close -> Open

SEE ALSO CLOSE, FIND, SAVE, DEL

OPEN VARIABLE

This command will set up a file so that you may access data from it or write records to it from within your program. This is the 'preferred' method of opening files since it affords more flexibility than the standard **OPEN** command.

OPENV *filename* **FNUM** *file_number* **TYPE** *file_type* **EXT** *extension*
LOCK *lock_type* **ERR** *error_label* **OWNER** *owner* **PATH** *path*
FD *file_descriptor* **SIZE** *size* **BUFF** *buffer_field_name*
CREATE NOCLR

filename - f/c/e - Required - The name of the file to be opened. You may specify the entire file name including disk drive specifier, path, filename and extension. If this is a TAS Pro 5.1 file then the extension defaults to .B (other extensions are provided by the **extension** option or are blank by a previously set default). The **filename** can be a field, expression or alpha constant. If a path is specified in FILELOC.B and not here, that value will be used. If a path is included here it will be used no matter what's in FILELOC.B for this file.

file_number - fn/v - Required - After the program opens a file (of any type) it returns a value that represents the number of the file opened. It is used by all the other commands that access the file for any purpose. This must be an I type field.

file_type - TDFXB - Optional - The type of file being opened. The options are: **T** - TAS Pro 5.1, **D** - dBASEIII+, **F** - Fixed length records (SDF), **X** - Text type and **B** - non-TAS Pro 5.1 Btrieve files. **T** is the default value; it need not be specified. TEXT is for files such as letters, or others that have records that are not all the same size. Each TEXT record is terminated by a CR/LF (carriage return/line feed - 0Dh/0Ah) pair. In FIXED files all records are of the same length; they are also terminated with CR/LF pairs. **D** is for files created by dBASE III+ or those that fit the same requirements. You must still set the **size** for the **D** files, although the program knows where the data actually starts and can keep track of the number of records.

NOTE: This value will also default to that value set in FILELOC.B for file type.

extension - f/c/e - Optional - If you wish to open a copy of the file with an **extension** other than the current default value (.B or whatever the **company_code** value is set to), you may specify up to 3 characters here.

lock_type - NRFXA - Optional - The type of locking to use. **N** - none, **R** - record, **F** - file, **X** - fix, and **A** - accelerated access (essentially file locks). The default is **N** - none. By specifying **A** you can significantly speed up operations to Btrieve files as long as you are not concerned with recovering data in case of a failure and you are running on a single-user system. Specifying **X** tells the program to open the Btrieve file in read-only or FIX mode. You will then be able to read records in direct mode (i.e., without an index). This feature is also part of the Reindex and Restructure routines and should allow you to recover from some Btrieve errors without restoring the file from a backup.

NOTE: This option is ignored for non-TAS files. They are opened automatically as shared if the multi-user flag is set to **Y** in the setup.

error_label - label - Optional - Where to transfer control if an error occurs while attempting to open the file. If this is omitted you may check for an error opening the file through the use of the **OPEN()** function. If you do not want to have the program display an error if the file can't be opened use the label name *NO_ERR*.

owner - f/c/e - Optional - A maximum 8 character code to be specified if the file is protected with the **OWNER** command. This applies to Btrieve files only.

path - f/c/e - Optional - You may specify the **path** for this file if you wish to override the value in the FILELOC.B file.

NOTE: The path value must end with a backslash character. See the CHK_PATH_CHR() UDF in TASRTNS.LIB for an easy method of making sure that the last backslash is always there.

file_descriptor - f/c/e - Optional - You may specify the **file_descriptor** name for this file if it is not in FILELOC.B and it is defined in the file Data Dictionary. This allows you to compile a program using one data dictionary and then run it on a system that has a different data dictionary without having to worry whether or not that FD exists in the new dictionary.

size - f/c/e - Optional - In the case of a non-TAS Pro 5.1 file you must specify the maximum size of the record. In the case of a text or fixed length type file this must also include the final CR/LF pair (2 extra characters).

NOTE: If the non-TAS file is defined in the data dictionary you do not need to include this value.

buffer_field_name - fn/v - Optional - In the case of a non-TAS Pro 5.1 file you must specify the buffer that will hold the record when it is read, unless it is defined in the file Data Dictionary. You may use multiple fields to make up a buffer; however, you would specify just the first field in this option. You may also define a buffer in a program for a TAS Pro 5.1 file. To use the buffer in the program instead of the normal buffer created in memory specify the name of the first field here just as you would with a non-TAS file.

If you are specifying a buffer for a non-TAS file don't forget to include the extra CR/LF bytes at the end of the record in both the size of the buffer and the size of the record if stipulated in this command.

NOTE: Non-TAS files that are defined in the data dictionary allocate their own buffers automatically just like regular TAS files.

CREATE - Optional - If this option is included in the command the program will automatically create a new file upon opening. If the file exists it will be deleted. This only applies to file types: **F** and **X**. To create a dBaseIII+ file use the Initialize File option in the Database sub-menu. For more information please refer to **Chapter 3, Main Menu Programs**.

NOCLR - Optional - Normally the buffer is cleared when a non-TAS Pro 5.1 file is opened. If you have data in it already, and do not want to clear it then include this option in the command.

COMMENTS

If the file is defined in the File Data Dictionary any fields that are a part of it cannot be accessed in the program until the file is opened. Any attempt to do so will generate a warning error at runtime.

With this command you may open dBaseIII+, Text, and Fixed Length records and treat them as though they were standard TAS Pro 5.1 files. This means you can do standard searches on the files, read and write records at will and in general treat them just as though they were Btrieve files. **ALL** commands that access files can be used to access these non-TAS files also. You would also use this command to open non-TAS Btrieve files where the file layout is a group of defined fields in memory (predefined or added during operation) instead of a standard defined record in the data dictionary. This command brings a level of flexibility to file handling not found in other 4GLs.

You can also define non-TAS 4.0 files in the file Data Dictionary. Make sure you set the proper file type when initializing the file. When a file is in the file Data Dictionary all the rules that apply to a standard TAS Pro 5.1 file also apply to it, including setting up the file name, etc. This gives you the ability to access non-TAS files as easily as regular TAS files.

NOTE: The maximum number of files that may be open at one time with either the **OPEN** or **OPENV** command is 100. This limit applies across all programs that are currently active.

PROGRAM EDITOR fiLe -> Open/close -> open Variable

SAMPLE PROGRAMS EXPERRFL, EXPERRFX

SEE ALSO CLOSE, FINDV, SAVE, DEL

OPEN NON-TAS FILE

This is a TAS Professional 3.0 command here for compatibility. The preferred method is to use the **OPENV** command.

OPNO *filename* **TYPE** *file_type* **SIZE** *size* **BUFF** *buffer_field_name*

filename - f/c/e - Required - The name of the file to be opened. You may specify the entire file name including disk drive specifier, path, filename and extension.

file_type - FX - Required - The type of file being opened. The options are: **F** - Fixed length records (SDF) or **X** - Text type and **B** - non-TAS Pro 5.1 Btrieve files. TEXT is for files such as letters, or others that have records that are not all the same size. Each TEXT record is terminated by a CR/LF (carriage return/line feed - 0Dh/0Ah) pair. In FIXED files all records are of the same length; they are also terminated with CR/LF pairs.

size - f/c/e - Required - You must specify the maximum size of the record. This must also include the final CR/LF pair (2 extra characters).

buffer_field_name - fn/v - Required - You must specify the buffer that will hold the record when it is read. You may use multiple fields to make up a buffer; however, you would specify just the first field in this option. Don't forget to include the extra CR/LF bytes at the end of the record in both the size of the buffer and the size of the record.

COMMENTS

This is the equivalent of the TAS Professional 3.0 command Open Non-TAS File.

PROGRAM EDITOR 3.0 Commands -> J - Opn non-TAS

OTHERWISE

This is a process control command, i.e., it will control whether a command, or group of commands will be executed. This is one part of a complete command structure.

OTHERWISE

No options

COMMENTS

For more information on the different structured programming commands, and the **OTHERWISE** command in particular, please see **Chapter 7, Structured Programming Commands.**

PROGRAM EDITOR

prg Control -> Case -> Otherwise

SEE ALSO

SELECT, CASE, ENDC

OWNER

This command is used to encrypt, or protect against unauthorized usage, a standard TAS Professional 5.1 file.

OWNER *name* **FILE** *filename/@file_number CLR/SET ENC RDOK*

name - f/c/e - Required - This is either the 'password' provided during the original execution of this **OWNER** command in relation to the specific file, or the new 'password' for this file. The field must be of A type and cannot be more than 8 characters in length. If you are going to clear current protection (*CLR* option), then this needs to be provided so that the program can proceed. In the case of the set option (*SET*), then it is needed for future operations. Once it is set you will need to provide it to the **OPEN** or **OPENV** (Open Variable) command each time the file is opened. This can be accomplished in two ways. You can store the value in a file somewhere (not very safe) or you can ask for the value before opening the file using the **password** option in the **ENTER** command so that the value entered will not be displayed to the screen.

filename/@file_number - f/c/e - Required - This is the name or number of the file being set. The file must currently be opened.

CLR/SET - Required - What to do. If *SET* is chosen then the protection will be set; if *CLR* then any protection currently provided will be removed.

ENC - Optional - If this is a *SET* operation then you can choose to encrypt the file also. This means that if anyone should try to access that file through any other means (debugger, other applications that use Btrieve, etc.) they will not be able to make sense of the data. If this is being accomplished on an existing file the process may take a considerable length of time depending on the number of records active. To choose this option include *ENC* in the command.

RDOK - Optional - If this is a *SET* operation and this option is included in the command then subsequent programs that access this file, but don't set the **OWNER** option during the **OPEN** or **OPENV** (Open Variable) command, will be able to read records from this file but won't be able to update them.

COMMENTS

Through the use of this command you can make sure that a file will have a certain level of protection without concern as to what the program may do, or how the user may access the file through utilities.

PROGRAM EDITOR fiLe -> Open/close -> oWner

SEE ALSO OPEN, OPENV

PAINT

Use this command to 'paint' a block of color on the screen.

PAINT *color* **FROM** *up_lt_col,up_lt_row* **THRU** *low_rt_col,low_rt_row*

color - f/c/e - Required - The color to paint. Must be a value from 1 through 255.

up_lt_col - f/c/e - Required - The upper left column value. Together with the upper left row value defines the upper left corner of the block. The upper left corner of any window is 1,1.

up_lt_row - f/c/e - Required - The upper left row value. Together with the upper left column value defines the upper left corner of the block. The upper left corner of any window is 1,1.

low_rt_col - f/c/e - Required - The lower right column value. Together with the lower right row value defines the lower right corner of the block. If the values are beyond the current window they will be truncated to those values.

low_rt_row - f/c/e - Required - The lower right row value. Together with the lower right column value defines the lower right corner of the block. If the values are beyond the current window they will be truncated to those values.

COMMENTS

The same effect can also be accomplished using the **MOUNT** command. Please see **Chapter 8, Screen/Report Format Information** for more information about specifying color blocks in a screen format.

PROGRAM EDITOR User interface -> Color control -> Paint

SAMPLE PROGRAM PAINTSCR

PARAMETER

This is the command used to 'receive' values from a previous program in which the **with** option was set in the **CHAIN** command.

PARAM *field_list*

field_list - fn/v1,fn/v2,...,fn/vx - The names of the fields that will be used to receive the values from the **with** option in the **CHAIN** command. If the receiving field type does not match the type of the 'giving' field, the value will be converted to the receiving field type. Fields are separated by commas.

COMMENTS

If this program is run directly or there are no values to pass, the fields in the **PARAM** command will remain blank.

PROGRAM EDITOR System -> Other programs -> Parameters

SEE ALSO CHAIN, RAP

PEEK (*DOS only*)

Use this command to 'peek' at a specific location in memory.

PEEK *offset* **FLD** *receiving_fld* **NCHR** *number_of_chrs*

offset - f/c/e - Required - The actual memory location.

receiving_fld - fn/v - Required - The field that will receive the value at the specified memory location.

number_of_chrs - f/c/e - Optional - The number of characters to get. If you don't specify a number here the program defaults to 1.

COMMENTS

This command, along with **POKE**, gives you a quick and easy way to check certain settings on a PC when you know the value will always be in the same location in memory.

PROGRAM EDITOR System -> Programming -> pEek

SAMPLE PROGRAM CAPSLOCK

SEE ALSO POKE, TEST()

PICTURE (*Windows only*)

Use this command to put a .BMP (bitmap), .ICO (icon) or .WMF (Windows metafile) picture on the screen. The user can also click on this picture with their mouse. Each time they click the picture it will be just as though they had pressed the key that you setup as the related value.

PICTURE AT *col,row* **LEN** *length* **WDT** *width* **FILE** *file_name* **KEY** *related_key*
NUM *picture_handle NO_BORDER AUTOSCROLL LOAD REMOVE*

col - f/c/e - Required - The column value for the upper left corner of the picture. The first column on the screen (far left position) is number 1. This value is always based on the largest window, generally the window that is created when the program is run. Even if the active window is smaller the column and row values are for the base window. This value is required only when creating the picture initially.

row - f/c/e - Required - The row value for the upper left corner of the picture. The first row on the screen (top row) is number 1. This value is always based on the largest window, generally the window that is created when the program is run. Even if the active window is smaller the column and row values are for the base window. This value is required only when creating the picture initially.

length - f/c/e - Required - The length or height of the picture in rows. This is required only when creating the picture initially.

width - f/c/e - Required - The width of the picture in columns. This is required only when creating the picture initially.

file_name - f/c/e - Optional - The name of the file to be loaded, including full path and extension. This would be specified when ever you want to display a new picture on the screen. You can also specify the file when creating a new picture. If you don't supply the file name the program will create a blank picture frame.

You can also load new pictures in a current picture frame by specifying the **file_name** and the **LOAD** option in this command.

key - f/c/e - Optional - The keyboard key this picture will emulate when it is clicked. This is required only when creating the picture initially. If you don't specify a key value, or one doesn't match those available, the program will ignore any user clicks on the picture. The following are the acceptable key values:

F1 through F10
SF1 through SF10 (Shift F1 etc.)
^F1 through ^F10 (Ctrl F1 etc.)
@F1 through @F10 (Alt F1 etc.)
^A through ^Z (Ctrl A etc.)
@A through @Z (Alt A etc.)
ESC, UP (up arrow), DOWN (down arrow), LTA (left arrow), RTA (right arrow), HOME, END, PGUP, ^PGUP, PGDN, ^PGDN, INSERT, DELETE, WDLT (ctrl left arrow), WDRT (ctrl right arrow), TAB and BCKTAB.

An example of this option would be: ... Key 'ESC' ...

picture_handle - fn/v - Required - When you initially create the picture you must supply a field that will hold the picture number or handle. This is how you will refer to this picture whenever you want to load a new file or to remove the picture from the screen. This must be an I type field.

NO_BORDER - Optional - Normally, a narrow single line is drawn around the picture frame. If you don't want this border to appear then add this option to the command line.

AUTOSCROLL - Optional - If you specify this option the program will automatically add vertical and horizontal scroll bars to the picture frame as needed if the file loaded would exceed the frame size. If the picture would fit within the frame then no scroll bars will be displayed. This can only be specified when creating the picture initially.

LOAD - Optional - When you first create the picture or after the picture has been created the first time you can load new files into the frame. This will clear the current picture and display the new one. You must supply both the **NUM** and **FILE** values. You can change the picture file as many times as you wish, however, once a file is displayed it remains until you load a new one or **REMOVE** the picture completely.

REMOVE - Optional - Remove the picture from the screen. If the NUM (or handle) value is 0 then all pictures will be removed from the screen.

COMMENTS

When you create a new picture what you're really doing is creating a picture frame. If you supply a file name with the original command then that picture will be automatically displayed, however, you are not required to. Each time you load a new file you are, in a sense, replacing the picture in that frame.

PROGRAM EDITOR Win Commands-> Picture

SAMPLE PROGRAM WINTEST

SEE ALSO BUTTON, HOT_SPOT

POINTER

You can use this command to get the internal 'pointer' value for the field.

receiving_field -> target_field

receiving_field - fn/v - Required - The receiving field. The internal 'pointer' value for **target_field** will be saved into **receiving_field**. If you are saving the pointer for a non-array field it may be saved into either a type F or P field. However, if the pointer is for an array field, the result must be saved into a type P field or an inaccurate value will be saved.

-> - Required - The pointer command symbol.

target_field - fn/v - Required - The field being pointed at.

COMMENTS

The difference in receiving fields comes from the way fields are accessed in TAS Professional 5.1. An F type pointer keeps track of the field type and address in the field list buffer. Only 5 bytes are required for this type of field. The P type pointer uses the actual location in memory and keeps track of the type, size, decimal characters, etc. of the field being pointed to. The P type is necessary when pointing to an array field since the array value is not kept in an F type pointer. For more information about the layout,

size, and specifications of TAS Professional 5.1 fields please see **Chapter 1, Installation and General Information**.

Through the use of this command you can access fields indirectly either in a TAS Professional 5.1 program or an external program.

NOTE: You must keep at least one space on both sides of the pointer command symbol.

PROGRAM EDITOR Field -> Pointer

SAMPLE PROGRAMS FPTRTST, FTST, MTEST, PTRMATH

POKE (*DOS only*)

Use this command to 'poke' or put a byte or string of bytes into a specific location in memory.

POKE *offset* **FLD** *giving_fld* **NCHR** *number_of_chrs*

offset - f/c/e - Required - The actual memory location.

giving_fld - fn/v - Required - The field that contains the value to be inserted into memory at the specified location.

number_of_chrs - f/c/e - Optional - The number of characters to poke. If you don't specify a number here the program defaults to 1.

COMMENTS

This command, along with **PEEK**, gives you a quick and easy way to make certain settings on a PC when you know the appropriate area will always be in the same location in memory.

PROGRAM EDITOR System -> Programming -> pOke

SAMPLE PROGRAM CAPSLOCK

SEE ALSO PEEK, TEST()

POP FIELD

Restore a value in a field or group of fields that was saved previously using the **PUSHF** (Push Field) command.

POPF *field_list*

field_list - fn/v1,fn/v2,...,fn/vx - Required - The list of fields to restore.

COMMENTS

When you use the **PUSHF** command the field values are saved on an internal stack. This stack, just like all others used in microcomputers, is a Last In-First Out process. That means that the last value pushed on the stack must be the first value popped out. The TAS Professional 5.1 stack, however, is different from a 'normal' stack in that it is intelligent. It knows the sizes of the values being pushed/popped and automatically allows for the differences.

The standard internal stack size is 1024 bytes. Each field put on the stack using **PUSHF** requires or takes up a number of bytes equal to the internal size of that field; i.e., A type fields - display size, I type fields - 2 bytes, N type fields - 8 bytes, R/T/D type fields - 4 bytes, F type fields - 5 bytes, L/B type field - 1 byte, and P type fields - 14 bytes.

This is a powerful but potentially dangerous command. When a programmer does a **GOSUB** or a UDF/UDC within a program the return address is also placed on this same stack, and if an unbalanced number of **PUSHx/POPx** commands have been used, the program will not return to the proper location. Also, if the **POPx** items are not in the exact reverse order from the **PUSHx** list, incorrect values will be restored. This could cause untold problems. For these reasons, all the **PUSHx/POPx** commands should be used with the greatest of care. However, the existence of these commands helps to make TAS Professional 5.1 a very powerful development tool.

PROGRAM EDITOR Field -> pOp field

SEE ALSO PUSHF

POP STACK

Remove the value pushed on the internal stack by a previous **GOSUB** command.

POPS

No options

COMMENTS

This command needs to be used with great caution. However, it will allow you to do something other than just returning from a **GOSUB** if necessary.

You may use this command only in connection with a **GOSUB** or **PUSHx** command. Each **POPS** removes 2 bytes from the internal stack. You may **NOT** use this command in connection with a **UDF** or **UDC**.

PROGRAM EDITOR prg Control -> Goto/gosub -> Pop stack

SEE ALSO GOSUB

POP TRAP

Restore a value in a trap or group of traps that was saved previously using the **PUSHT** (Push Trap) command.

POPT *trap_list*

trap_list - trap1,trap2,...,trapx - Required - The list of traps to restore.

COMMENTS

When you use the **PUSHT** command the current trap settings are saved on an internal stack. This stack, just like all others used in microcomputers, is a Last In-First Out process. That means that the last value

pushed on the stack must be the first value popped out. The TAS Professional 5.1 stack, however, is different from a 'normal' stack in that it is intelligent. It knows the sizes of the values being pushed/popped and automatically allows for the differences.

The standard internal stack size is 1024 bytes. Each **PUSHT** uses 8 bytes.

This is a powerful but potentially dangerous command. When a programmer does a **GOSUB** or a UDF/UDC within a program the return address is also placed on this same stack, and if an unbalanced number of **PUSHx/POPx** commands have been used, the program will not return to the proper location. Also, if the **POPx** items are not in the exact reverse order from the **PUSHx** list, the incorrect values will be restored. This could cause untold problems. For these reasons, all the **PUSHx/POPx** commands should be used with the greatest of care. However, the existence of these commands helps to make TAS Professional 5.1 a very powerful development tool.

PROGRAM EDITOR prg Control -> Trap -> pOp trap

SEE ALSO PUSHT

PORT (*DOS only*)

Use this command to output a byte or string of bytes to a specific computer port.

PORT *portnum* **FLD** *giving/receiving_fld* **IN/OUT**

portnum - f/c/e - Required - The actual computer port number. Normally port numbers are in hex form, this must be in decimal, i.e., port 200h is 512 decimal.

giving/receiving_fld - fn/v - Required - The field that contains the value to be output to the port or input from the port. If you are using a 16550 UART you can have a FLD buffer of more than 1 character, otherwise, you should use get/receive only one character at a time. This command will attempt to get/send the same number of characters as the size of this field.

IN/OUT - Required **- IN** gets characters from the port, **OUT** sends characters out the port.

PROGRAM EDITOR System -> poRt

PRINT ALL

If you want to be able to print CR or LF characters to the screen and not go to the next line then set this option to *ON*.

PRTALL *ON/OFF*

O*N/OFF* - Required - If *ON* the program will treat CR and LF characters as though they were normal, displayable characters. To return to the normal process this should be *OFF*.

COMMENTS

You will probably never have the need for this command since in almost every case you want the program to move the display to the next line when a CR/LF is encountered. If this option is *ON*, the

only way to move to the next line (or another line) is to specify the row and column coordinates in the appropriate command.

PROGRAM EDITOR User interface -> Screen control -> Print all

PRINT BLANK LINES

Use this command to print a certain number of blank lines or spaces to the printer, disk file or screen.

PBLNK *number* **PTW** *print_to*

number - f/c/e - Required - The number of blank lines to print.

print_to - SPD - Optional - You can use this option to specify where to print the blank lines. The option ASK (a regular **print_to** option) is not available. The only options available are **S** - screen, **P** - printer or **D** - default.

COMMENTS

This command accomplishes its task through the repeated use of the CR/LF pair (carriage return/line feed - 0Dh/0Ah).

PROGRAM EDITOR Reports -> Blank lns

PRINT BOX

Use this command to display a box on the screen.

PBOX *type* **AT** *up_lt_col,up_lt_row* **LEN** *length* **WDT** *width* *CLR* **COLOR** *color* *ABS*

type - SDC - Optional - The box type options. **S** - single line, **D** - double line and **C** - custom. The default is **S**.

up_lt_col - f/c/e - Required - The upper left column value. Together with the upper left row value defines the upper left corner of the box. The upper left corner of any window is 1,1.

up_lt_row - f/c/e - Required - The upper left row value. Together with the upper left column value defines the upper left corner of the box. The upper left corner of any window is 1,1.

length - f/c/e - Optional - The vertical depth of the box. For example, if the starting row value is 5 and you want the lower horizontal portion of the box to be on row 10 then the length is 6. If no value is given the default is 0. In this case the program will print a single horizontal line.

width - f/c/e - Optional - The horizontal width of the box. For example, if the starting column value is 5 and you want the far right corner of the box to be on column 10 then the width is 6. If no value is given the default is 0. In this case the program will print a single vertical line.

CLR - Optional - You can use this option if you want to clear a box that currently exists.

color - f/c/e - Optional - What color value to use when displaying the box. If no value is given the program will use the current screen color. Please note: this will affect just the box-drawing characters only. The interior of the box will not be changed.

ABS - Optional - If this option is included in the command the program will ignore any windows currently active and will place the box according to the location calculated from the absolute upper left hand corner of the screen. Normally, the location would be figured from the upper left hand corner of the current window.

COMMENTS

If you provide no **length** value a single horizontal line will be displayed. No **width** value and only a single vertical line will appear. If neither value is provided nothing will be displayed.

PROGRAM EDITOR

User interface -> Screen control -> print Box

SAMPLE PROGRAM

BOXTEST

PRINT CONTROL CHARACTERS

This command is used to send a series of control characters or a single control character to the printer. You may choose to send a string of characters provided within the program or use a value saved in the TASPRTR.CTL file (or whichever driver is currently active).

PCHR *ctl_name* **CHR** *character_list* **PTW** *print_to*

ctl_name - f/c/e - Optional - The name of the series of characters to send. It must correspond to a name in the currently loaded CTL driver. If you are going to print a string of characters then this value should be ‘’ (null string).

character_list - f/c/e1,f/c/e2,...,f/c/ex - Optional - This must be a series of characters (CHR() functions or B type fields). The program will output the first character of each field. If this option is specified the program will use this string instead of the **ctl_name**, whether or not it is a legal name. A series of characters might be:

CHR(27),CHR(91),...

print_to - PD - Optional - You can use this option to specify where to print the characters. The option ASK (a regular **print_to** option) is not available. The only options available are **P** - printer or **D** - default.

COMMENTS

For more information on the printer driver files please see **Chapter 3, Main Menu -Utilties**. Output will only go to printer or disk; if output is directed to the screen (PON S) nothing will happen.

PROGRAM EDITOR

Reports -> Chr strg

PRINT FORMAT

This command will print a line or group of lines from a Report Format that has been previously mounted.

PFMT *format_lines* **THRU** *thru_line* **AT** *starting_column,starting_row WAIT NOCR* **PTW** *print_to ABS*

format_lines - f/c/e1, f/c/e2,..., f/c/ex - Required - This is (these are) the line number(s) to print of the appropriate format. The first line in the format is line 1 and blank lines are counted also. Up to 80 different format lines may be included. The line numbers need not be sequential.

thru_line - f/c/e - Optional - If you include this option, the program will print all the report format lines starting with the last one specified in the **format_lines** list through the one specified in this option. For example:

PFMT 3,5,7 THRU 10

will print lines 3,5,7,8,9 and 10.

starting_column - f/c/e - Optional - The number of the column at which to start the printing. The first column on the screen or printer (far left position) is number 1. The default value, if none is provided, is the current column value as set by the last output. This value can be easily determined through the use of the **COL()** function for the screen or **PCOL()** for the printer.

starting_row - f/c/e - Optional - The number of the row at which to start the display. The first row on the screen or printer (top) is number 1. The default value, if none is provided, is the current row value as set by the last output. This value can be easily determined through the use of the **ROW()** function for the screen or **PROW()** for the printer.

WAIT - Optional - If this option is included, it will force the program to wait after completing this command for the user to press any key at the keyboard before continuing with the next command.

NOCR - Optional - Normally, this command will print a CR/LF (0dh/0ah) after each format line. If this option is included in the command that will not happen. That will force each line, or the next line printed, to be printed on the same line until the line is automatically truncated or a print command without this option is executed.

NOTE: If you use this feature and find that your lines are not being printed properly make sure they don't extend off the edge of the screen or printer. If they do use the **WRAP** command (for long fields) or split up the lines.

print_to - SPD - Optional - You can use this option to specify where to print. The option ASK (a regular **print_to** option) is not available. The only options available are **S** - screen, **P** - printer or **D** - default.

ABS - Optional - If this option is included in the command the program will ignore any windows currently active and will place the box according to the location calculated from the absolute upper left hand corner of the screen. Normally, the location would be figured from the upper left hand corner of the current window. This only applies when displaying to the screen.

COMMENTS

This command makes up the largest part of any report program. The ability to print the same format to any device gives you great flexibility. To create complete reports with no extra programming required see the Report Writer, **Chapter 3, Main Menu - Programming.**

The maximum number of characters that may be printed in a single report format line is 254.

PROGRAM EDITOR

Reports -> prt Format

SAMPLE PROGRAM

RPTFMT

PRINT MESSAGE

The command will allow you to send a message, data, or combination of both to the *SCREEN*, *PRINTER* or default device.

PMSG *message_list* **AT** *starting_column,starting_row WAIT NOCR* **PTW** *print_to*
ENTER *enter_field* **COLOR** *color ABS*

message_list - f/c/e1, f/c/e2,..., f/c/ex - Required - Any combination of constants, expressions, and or field values. These will print in a continuous line. For example:

PMSG 'ABC','DEF',X

assuming X = 'GHI', will display:

ABCDEFGHI

You may include a maximum of 80 fields, constants, or expressions.

starting_column - f/c/e - Optional - The number of the column at which to start the display. The first column on the screen (far left position) is number 1. The default value, if none is provided, is the current column value as set by the last display or enter to the screen. This value can be easily determined through the use of the **COL()** function.

starting_row - f/c/e - Optional - The row or line number to display at. The first row on the screen (top) is number 1. The default value, if none is provided, is the current row value as set by the last display or enter to the screen. This value can be easily determined through the use of the **ROW()** function.

WAIT - Optional - If this option is included, it will force the program to wait after completing this command for the user to press any key at the keyboard before continuing with the next command.

NOCR - Optional - Normally, this command will print a CR/LF (0dh/0ah) after the **message_list** is printed. If this option is included in the command that will not happen. That will force the next line printed to print on the same line until or unless the line is automatically truncated or a print command without this option is executed.

NOTE: If you use this feature and find that your lines are not being printed properly, make sure they don't extend off the edge of the screen or printer. If they do, use the **WRAP** command (for long fields) or split up the lines.

print_to - SPD - Optional - You can use this option to specify where to print. The option ASK (a regular **print_to** option) is not available. The only options available are **S** - screen, **P** - printer or **D** - default.

enter_field - fn/v - Optional - By using this option you can allow the user to enter a value into the field specified. The entry will occur after any **message_list** value(s) is displayed. It will be entered on the same line, if possible. If the field is specified as **UP** (upcase) in the **DEFINE**/Data Dictionary that will apply here as will any **PICT** specification.

This is not the same as a field that is part of a mounted screen format. The field to be entered is displayed in 'enhanced' video. However, whatever color was present before the **PMSG** command will be restored after the **ENTER** option. Also, the field value displayed is not automatically updated if changes are made to it further in the program.

color - f/c/e - Optional - What color value to use when displaying the **message_list** value(s). If no value is given the program will use the current screen color. Please note: this will not affect the 'enhanced' color value if the **ENTER** option is used. This only applies to messages displayed on the screen.

ABS - Optional - If this option is included in the command the program will ignore any windows currently active and will place the box according to the location calculated from the absolute upper left hand corner of the screen. Normally, the location would be figured from the upper left hand corner of the current window. This only applies when displaying to the screen.

COMMENTS

This command provides you with a quick and easy method of displaying information to the user. Almost all programs use this command at least once.

If you want to use this command to have the program wait for a key stroke from the user but do not want to print any specific message you may print a null string. For example:

PMSG '' *WAIT*

In this case nothing will appear on the screen but execution will cease until any key is pressed by the user.

If you have specified coordinates that are outside of the current window and do not specify the *ABS* option or use the **FORCE** command the message will not print at all and will give you no indication of that.

In TAS Professional 5.1, when you send output to the printer or disk, if you specify a row value and it is prior to the current row, the program will do an automatic form-feed before printing the data.

NOTE: You may use the '**?**' (question mark) instead of **PMSG** for the command name, if desired.

PROGRAM EDITOR

User interface -> Messages -> Print message

PRINT TO

This command will set the default output device for any other output command.

PON *which_device* **PFN** *print_to_disk_file_name*

which_device - ASPD - Required - This sets the default output device. The options are **A** - ask, **S** - screen, **P** - printer or **D** - disk. If you specify option **A**, the program will ask the user at runtime where to print.

print_to_disk_file_name - f/c/e - Optional - If you set the **which_device** value to **D** the name of the file to use may be specified with this option. You need to indicate the entire file name, including any path and extension. If the file name is not specified the program will ask for one at runtime.

NOTE: When checked at runtime, if the file exists on disk already, the user will be asked if he/she wants to overwrite. If the answer is **N** the user will be able to enter a different file name. This applies whether the file name is supplied here or at runtime.

COMMENTS

This command sets the default output device; however, individual commands may override this value. In the case of all other print commands, this command must be used to set the device to *DISK* if it is to be used for output.

If a print to disk file is already open, it will be closed when this command is executed.

PROGRAM EDITOR

Reports -> On

SEE ALSO

PSET

PRINT TOP OF FORM

The command will prepare the output device for a new page.

PTOF *print_to*

print_to - SPD - Optional - You can use this option to specify which output device. The option ASK (a regular **print_to** option) is not available. The only options available are **S** - screen, **P** - printer or **D** - default.

COMMENTS

If the output device is the screen, the "Press Any Key" message will be displayed. When the user presses a key, the screen will be cleared and the default row and column will be reset to 1. If the output device is printer or disk then a form_feed (binary character 12 - 0ch) will be sent to the device. If the number of lines per page (**total_lines**) is set to something other than 66, and the device is printer or disk, the program will use CR/LF's to get to the next page instead of a form_feed.

PROGRAM EDITOR

Reports -> Top of form

PRINT VERTICAL TAB

The command will move the next output to the line specified in the command.

PVERT *line_number* **PTW** *print_to*

line_number - f/c/e - Required - The line number to move to. If the value specifies a row before (or above) the current row location, this command will be ignored.

print_to - SPD - Optional - You can use this option to specify the output device. The option ASK (a regular **print_to** option) is not available. The only options available are **S** - screen, **P** - printer or **D** - default.

COMMENTS

The column value is automatically reset to column 1 after this command is executed. The command does a series of CR/LF's (0dh/0ah) until the appropriate row is reached.

PROGRAM EDITOR

Reports -> Vert tab

PRINTER NUMBER (*DOS only*)

Use this command to switch among the 3 LPT printers: LPT1, LPT2 or LPT3.

PRT_NUM *lpt_number*

lpt_number - f/c/e - Required - The printer number to use as the printer output device. This may be 1, 2 or 3.

COMMENTS

This value defaults to what you have set in the TAS50.OVL file. You can also change the printer driver with the **LD_PDRV** (Load Printer Driver) command.

PROGRAM EDITOR

Reports -> Printer -> Number

SEE ALSO

LD_PDRV

PRINTER SET

This command will allow you to change the default printer sizes.

PSET WDT *width* **TLNES** *total_lines* **PLNES** *printable_lines* **PWHR** *print_where*

width - f/c/e - Optional - The maximum number of printable columns. The default is 80.

total_lines - f/c/e - Optional - The maximum number of lines to a page. The default is 66.

printable_lines - f/c/e - Optional - The maximum number of printable lines on a page. The default is 60.

print_where - f/c/e - Optional - You can redirect the output to any of the standard devices with this option without using the **MOUNT** or **PON** (print to) command. This option will

not cause a page break as the **PON** would when changing to the screen. The value must resolve to **S**, **P** or **D**. If it doesn't then nothing will be changed and no error is returned. If you redirect to **D** then an output file must have been opened previously by the **MOUNT** or **PON** command.

COMMENTS

All output commands that are directed toward the printer or disk make use of these values to determine how wide a line to print, when to do a top of form, etc.

You can retrieve these values from within a program using the **PSET()** function.

NOTE: This command will clear the **PROW()** and **PCOL()** values (except in the case of the **PWHR** option by itself).

PROGRAM EDITOR

Reports -> Printer -> Setup

SEE ALSO

LD_PDRV

PUSH FIELD

Save a value in a field or group of fields to the internal TAS Pro 5.1 stack so that they may be restored later with the **POPF** (Pop Field) command.

PUSHF *field_list*

field_list - fn/v1,fn/v2,...,fn/vx - Required - The list of fields to save.

COMMENTS

When you use the **PUSHF** command, the field values are saved on an internal stack. This stack, just like all others used in microcomputers, is a Last In-First Out process. That means that the last value pushed on the stack must be the first value popped out. The TAS Professional 5.1 stack, however, is different from a 'normal' stack in that it is intelligent. It knows the sizes of the values being pushed/popped and automatically allows for the differences.

The standard internal stack size is 1024 bytes. Each field put on the stack using **PUSHF** requires or takes up a number of bytes equal to the internal size of that field; i.e., A type fields - display size, I type fields - 2 bytes, N type fields - 8 bytes, R/T/D type fields 4 bytes, F type fields - 3 bytes, L/B type field 1 byte, and P type fields 10 bytes.

This is a powerful but potentially dangerous command. When a programmer does a **GOSUB** or a UDF/UDC within a program, the return address is also placed on this same stack, and if an unbalanced number of **PUSHx/POPx** commands have been used the program will not return to the proper location. Also, if the **POPx** items are not in the exact reverse order from the **PUSHx** list, the incorrect values will be restored. This could cause untold problems. For these reasons, the **PUSHx/POPx** commands should be used with the greatest of care. However, it is the existence of these commands that helps to make TAS Professional 5.1 a very powerful development tool.

PROGRAM EDITOR

Field -> pUsh field

SEE ALSO

POPF

PUSH TRAP

Save a value in a trap or group of traps to the internal TAS Pro 5.1 stack so they may be restored later with the **POPT** (Pop Trap) command.

PUSHT *trap_list*

trap_list - trap1,trap2,...,trapx - Required - The list of traps to save.

COMMENTS

When you use the **PUSHT** command the current trap settings are saved on an internal stack. This stack, just like all others used in microcomputers, is a Last In-First Out process. That means that the last value pushed on the stack must be the first value popped out. The TAS Professional 5.1 stack, however, is different from a 'normal' stack in that it is intelligent. It knows the sizes of the values being pushed/ popped and automatically allows for the differences.

The standard internal stack size is 1024 bytes. Each **PUSHT** uses 4 bytes.

This is a powerful but potentially dangerous command. When a programmer does a **GOSUB** or a UDF/ UDC within a program, the return address is also placed on this same stack, and if an unbalanced number of **PUSHx/POPx** commands have been used the program will not return to the proper location. Also, if the **POPx** items are not in the exact reverse order from the **PUSHx** list, the incorrect values will be restored. This could cause untold problems. For these reasons, the **PUSHx/POPx** commands should be used with the greatest of care. However, it is the existence of these commands that helps to make TAS Professional 5.1 a very powerful development tool.

PROGRAM EDITOR

prg Control -> Trap -> Push trap

SEE ALSO

POPT

PUT FIELD IN BUFFER

This is a TAS Professional 3.0 command here for compatibility. The preferred method is to use the **EXPORT/IMPORT** commands.

PUT_FLD *fieldname*

fieldname - fn/v - Optional - This is the field that will be added to the end of the non-TAS file buffer. The command will use the active **OPNO** file. If you are initializing the buffer then don't put a field name here.

COMMENTS

This is the equivalent of the TAS Professional 3.0 command Put Field In Buffer.

PROGRAM EDITOR

3.0 Commands -> K - Put fld

QUIT

Provide a process for leaving a program.

QUIT *quit_level*

quit_level - f/c/e - Optional - This is the program level to quit to. If this is not provided the program will quit to the previous program. If there is no previous program you will be returned to DOS.

Quit_level 0 will return you to DOS, 1 will return you to the first program, 2 to the second, etc. Through the use of this option you can force the program to exit all the way to DOS or to a specific program. All appropriate action at each level is taken so that all files that need to be closed are closed, and all fields that need to be deallocated are removed from memory.

COMMENTS

TAS Professional 5.1 will automatically exit a program upon reaching the end of that program (the last executable line). (This doesn't apply if the last executable line is a **RET** command in a subroutine or UDF/UDC.) For example, the following program will exit automatically after the print command:

```
clrscr
? 'this is a test'
```

However, if you want to control where the program exits then the **QUIT** command must be used. For example, that same program with a subroutine would require the **QUIT** command:

```
        clrscr
        gosub prt_test
        quit
prt_test:
        ? 'this is a test'
        ret
```

If there was a previous program (i.e., the program being quit was chained to), that previous program will continue executing with the next line.

PROGRAM EDITOR System -> Quit

RAP

Please see the description under "RUN ANYTIME PROGRAM."

READ

This command will cause the program to read a specific number of characters from a non-TAS file at a certain location.

READ *file_number* **START** *start* **NCHR** *number_of_characters* **TO** *to* **MEM_AREA** *mem_area#* **OFST** *offset_within_mem_area*

file_number - f/c/e - Required - The file number value. This is the same value as that received in the **OPENV** (Open Variable) command. The file must have been previously opened.

start - f/c/e - Optional - The first byte position to read from. The first character position in the file, for this command, is 1. If no **start** value is given or the value is 0 the program will use the current internal file position value. (When the file is opened the position value is automatically set to 1.) This must be an R type field.

number_of_characters - f/c/e - Optional - The number of characters to read. If this is not supplied the program will use the **size** value stated in the **OPENV** (Open Variable) command.

to - fn/v - Optional - Normally the characters read would be saved in the **buffer** specified in the **OPENV** (Open Variable) command. However, you may use this option to specify a different buffer field to be used during the execution of this command.

mem_area# - f/c/e - Optional - If you want to store the characters read in a memory area you can specify the number here. If you use this it must resolve to a value of 1 through 4. NOTE: If you don't specify an **OFST** value the characters will start at position 1.

offset_within_mem_area - f/c/e - Optional - If you specify a memory area (**MEM_AREA**) you can also set an offset within that area. The first character in a memory area is at position 1.

COMMENTS

The internal file position value will be set to the first character following the characters read. For example, if the **start** value is set at 10 and the number of characters to be read is 10, the position after the **READ** command will be 20.

PROGRAM EDITOR

fiLe -> Read

SEE ALSO

WRITE, OPENV

READ ARRAY

The command is used for reading a group of records into array fields in memory. Within this single command you can accomplish many tasks that would normally take many more lines of code and time to execute.

RDA *from_field_list* **TO** *to_field_list* **FILE** *filename/@file_number*
KEY *keyname/@key_number* **START** *start_value* **SCOPE** *scope scope_value*
FOR *for_filter_expression* **WHILE** *while_filter_expression* **CNTR** *counter_field DISP*

from_field_list - fn/v/e1, fn/v/e2,..., fn/v/ex - Required - The fields in the record that are being read. You may also create expressions that can be transferred to the receiving (**to_field_list**) fields. The first **from_field_list** field is paired to the first **to_field_list** field, the second to the second, etc. There must be the same number of **from_field_list** fields as **to_field_list** fields.

NOTE: You cannot use constants in the **from_field_list** but you can use expressions. For example, you can read the record number for each record into an array of record numbers by using the **RCN()** function as one of the **from_field_list** fields and the appropriate R type array field as the corresponding **to_field_list** field.

to_field_list - fn/v1, fn/v1,..., fn/vx - Required - These are the recipient fields. They must be array fields of the same type as the respective **from_field**. The first **from_field_list** field is paired with the first **to_field_list** field, the second to the second, etc. There must be the same number of **to_field_list** fields as **from_field_list** fields.

filename/@file_number - file_expr - Optional - The name or number of the file to be used. If you do not include this option, the program will look for a default search file set in the **SRCH** (Search File) command. If a default file hasn't been previously set the program will report an error and the command will be skipped. This entry will override any default value set in the **SRCH** command.

keyname/@key_number - key_expr - Optional - Set this to the appropriate value to read the records in the file in a particular order. If you do not include this option, the program will look for a default key set in the **SRCH** (Search File) command. If a default key hasn't been previously set the program will report an error and the command will be skipped. This entry will override any default value set in the **SRCH** command.

start_value - f/c/e1,f/c/e2,...,f/c/ex - Optional - The value to use as the beginning record. This is similar to doing a **FIND** before the command. If the index you're searching on has multiple segments you may list the proper values for each of the segments, separating them with commas. This will allow you to specify the exact type for each of the segments. For example, suppose you're searching on an index that has two segments: a 5 character alpha and an I type field. Each record you want has the same alpha but the I field will change, and is in order. In the first record the I type field has a value of 0. The following would find the first record for that group, if it exists:

start 'ABCDE',0!

Notice that we separate the values with a comma. The only requirement is that the values be of the same type (and size) as the segments in the key. In this case we put the I type constant specifier (!) after the 0 to make sure that it is passed as an I type field.

scope - Optional - This option may help determine how many records are read. For more information about the **scope** specifier please see the general information at the beginning of this chapter.

scope_value - f/c/e - Optional - If **scope** is set to **N** or **F** this is the number of records to read.

for_filter_expression - lexpr - Optional - You would use this option to restrict the records read in this command. If the expression did not resolve to .T. the record would not be read, and therefore would not be added to the array. In this option, the search continues until the program reaches the end of file and then will quit.

while_filter_expression - lexpr - Optional - This option also restricts the records read. However, the first time the expression resolves to .F., the program stops reading records and continues to the next command. Thus if the program is reading records in a certain order (by a specific key) and the expression returns .F., then the program knows that all the appropriate records have been read and there is no need to continue reading. For example: Suppose you want to read all the invoice records for a specific customer. The first record for that customer is found in the invoice file either by using the **start_value** option within this command, or through the **FIND** command before executing this command. Then the **while_filter_expression** would be:

INV_CUST_CODE = CUSTOMER_CODE

(This assumes that there is a field called INV_CUST_CODE in the invoice file and a similar field in the customer file called CUSTOMER_CODE. The **keyname/ @key_number** option must be set to the proper value so that the records are read in CUSTOMER_CODE order, or you must have set the search key using **SRCH** previous to this command.) When the first invoice record for the next customer is read the program will stop reading records.

NOTE: If there is no **start_value** you must set the **scope** value to *R* or *N xxx*. If this is not done, the program will find the first record in the file and the **while_filter_expression** option will probably fail the first time. However, if the **start_value** option is used you can ignore this requirement.

counter_field - fn/v - Optional - This is an I type field that will be used for passing the number of records read, and the current array number, to your program. You can also use this to count the number of records read.

DISP **-** NY - Optional - Each time a record is read the program will redisplay the screen fields, if you specify this option. This will slow down the operation of this command, with the amount of slow-down depending on how many fields are displayed on the screen. The net effect will probably be negligible unless you are reading through a very large file.

COMMENTS

If the **while_filter_expression** option is used you must be careful to set **scope** and/or the **start_value** options properly. If you think you've set everything properly and yet no records are being read, make sure that the proper first record is in memory prior to the execution of this command or that correct use has been made of the **start_value** option. The last record read before exiting the command is still in the record buffer after the program leaves this command.

You must make sure you won't exceed the number of elements in the array. If there is a chance this might happen, you should use the **scope** to insure this doesn't happen. For example:

... **scope** *n 300* ...

This would limit the command to the next 300 records in the file.

PROGRAM EDITOR fiLe -> Mult Rec Cmds -> Read Array

SAMPLE PROGRAM RDARRAY

SEE ALSO WRTA

READ RECORD

This is a TAS Professional 3.0 command here for compatibility. The preferred method is to use the **OPENV/FINDV** commands.

RDREC *err_label*

err_label - label - Optional - This is the label to go to if an error occurs while reading from this non-TAS file.

COMMENTS

This is the equivalent of the TAS Professional 3.0 command Read non-TAS Record.

PROGRAM EDITOR

3.0 Commands -> L - Rd record

RECORD NUMBER

This command will either return the record number (position) of an active record, or will use the value supplied to read a record.

RCN *filename/@file_number* **RCN** *record_number_field GET/SET NLOCK*

filename/@file_number - file_expr - Required - The name or number of the file to use.

record_number_field - fn/v - Required - The field that will either hold the results of a *GET* or provide the value for a *SET*. This must be an R type field.

GET/SET - Required - *GET* the current record number, or *SET* the value in the **record_number_field** into the appropriate location and read the record.

NLOCK - Optional - Normally, in multi-user mode, when a record is read it is automatically locked so that other users cannot read it until you are finished with it. If this is a *SET* operation and you don't want to lock the record upon reading it then include this option in the command.

NOTE: If you include this option and another user reads the record and then you try to save the record back to disk you will get an error message telling you that this has occurred. You need to read the record again before saving it back to disk.

COMMENTS

This command will work with both TAS Pro 5.1 and non-TAS files.

The screen will not be automatically refreshed when this command executes. If you need to refresh data on the screen you should execute the **SCRN R** command immediately after a *GET* operation.

PROGRAM EDITOR

fiLe -> Find -> Record number

SEE ALSO

RCN()

REDEFINE

This command is used to change a field that has been previously defined. This field may be a part of a standard TAS Professional 5.1 file. However, the changes will only apply during the execution of the program.

REDEF *fieldname* **TYPE** *type* **SIZE** *size* **DEC** *decimal_characters*
FILE *filename/@file_number* **OFST** *offset_in_file* **KEY** *key_number* **PICT** *picture* **UP** *set_up_case* **LOC** *location*

fieldname - fn/v - Required - The name of the field being redefined. Must be a 'standard' TAS Professional 5.1 field name.

type - f/c/e - Optional - The new type value. May be:

	Name	Internal size
A	Alphanumeric	maximum 4 gbytes
N	Numeric	8 bytes
D	Date	4 bytes
T	Time	4 bytes
I	Integer	2 bytes
B	Byte	1 byte
F	F type pointer	5 bytes
P	P type pointer	14 bytes
R	Record number	4 bytes
L	Logical	1 bytes
O	TAS Pro 3.0 BCD	max of 10 bytes

NOTE: If you don't specify a new field **type** the program will use the current value.

size - f/c/e - Optional - The new display size. If the **type** is **A** then this is required and is both the display and internal size.

decimal_characters - f/c/e - Optional - If the **type** is **N** then you can also define the number of decimal characters. The maximum number is 8 and must be at least 2 less than the **size** value.

filename/@file_number - file_expr - Optional - If the field being redefined is part of a TAS Professional 5.1 file then this is the name or number of that file. The file must be opened before a field can be redefined.

offset_in_file - f/c/e - Optional - If the field being redefined is part of a TAS Professional 5.1 file then you can specify the new **offset** within the file buffer. The first character of the record is number 0.

key_number - f/c/e - Optional - If this field is a key you can specify the key number here. Then if this field is used in the **ENTER** command the user will be able to search for records using the file search keys (F5-F9).

NOTE: If the **SRCH** (Set Search File) command has been executed setting a specific search file and key it will override any other key fields.

picture - f/c/e - Optional - The new **picture** value to be used with this field. If this is a type 'A' field, you can use this option to enter a field into a space that is shorter than the defined length of the field. This is called a slider field. A slider is a field that shows fewer characters on the screen than actually exist in the field. The user can see the other characters by pressing the LEFT and RIGHT ARROW keys and the characters will appear to 'slide' back and forth across the available space. This is very useful when you have several large fields and want them all to fit on the same screen. A slider field is specified with an alpha constant of "Sxx", where xx is the number of columns to display.

set_up_case - f/c/e - Optional - This value must resolve to **Y** or **N**. If the value is **Y**, then any entries made to this field will be forced to upper case characters.

location - fn/v - Optional - If you specify a fieldname here, the program will use the location (offset) of that field for this one.

COMMENTS

The original size (internal) of the field being redefined must be enough to cover the size of the new field. The program does not check that this is true, and you can end up crossing from one field to another.

You should consider using the **ADD** command before using this one. It is much more flexible and doesn't have the restrictions on original size.

PROGRAM EDITOR Field -> Create/chg -> Redefine

SAMPLE PROGRAM REDEFINE

SEE ALSO DEFINE

REDISPLAY LIST

Use this command when you are in a subroutine called from the **LISTM** (List Array) or **LISTF** (List File) command and you want to redisplay all the current lines when you return.

RDLIST

No options

COMMENTS

You can see this process at work in the Maintain Database, Import and Export utilities. If you press the F3 key to choose all the fields in a record the program calls a UDF to do the work and then executes this command so that the 'stars' will appear when the process is finished.

PROGRAM EDITOR User interface -> Lists -> Redisplay list

SEE ALSO LISTF, LISTM

REDISPLAY SCREEN

This command will redisplay a screen that was previously saved using the **SAVES** (Save Screen) command.

REDSP *screen_holder_name*

screen_holder_name - fn/v - Optional - The name of the field that was used in the **SAVES** (Save Screen) command. If you did not specify a field name in that command (i.e., you saved the screen to an internal buffer), then you should not specify one here.

NOTE: If you do specify a field name it must be large enough to hold the entire screen and certain information about the current screen environment. This is approximately 4300 bytes.

COMMENTS

If you allow the program to save and redisplay the screen from an internal buffer you need to be sure that you balance your **SAVES** (Save Screen) and **REDSP** (Redisplay Screen) commands. Each time the program saves a screen to an internal buffer it allocates the memory required to do so. If you don't redisplay the screens and continue to save them you will eventually run out of memory. This does not apply to the situation where you save the screen to a named field.

When you **QUIT** the program any buffers used for saving screens are automatically released.

For more information about the effect of the **REDSP** command on screen formats and fields please refer to the **MOUNT** command.

PROGRAM EDITOR User interface -> Windows -> Redsply scrn

SEE ALSO SAVES

REDISPLAY SCREEN 3.0

This is a TAS Professional 3.0 command here for compatibility. The preferred method is to use the **REDSP** command.

REDSP3 *screen_buff_num*

screen_buff_num - f/c/e - Required - This is the screen buffer number. This must resolve to one of the following values:

1-10 - Redisplay entire screen including active windows
21-30 - Redisplay characters and color attributes only
41-50 - Redisplay characters only

COMMENTS

This is the equivalent of the TAS Professional 3.0 command Redisplay Screen.

PROGRAM EDITOR 3.0 Commands -> M - Redsp3

REENTER

This will re-execute the last **ENTER** command.

REENT

No options

COMMENTS

This is generally used when checking values after the **ENTER** command to make sure that what the user entered is acceptable. A better method is to use the **valid** and **valid_message** options in the **ENTER** command. This command has been provided for upward compatibility with TAS Professional 3.0.

PROGRAM EDITOR User interface -> reenT

SEE ALSO ENTER

RELATE

You would use this command to set a file as 'related' to another file. This means that when a record from the 'master' file is found, the appropriate record will be found in the 'slave' file without any further action from you.

REL *slave_filename/@file_number* **KEY** *slave_keyname/@key_number* **MSTR** *master_filename/@file_number* **FLDLST** *master_field_list*

slave_filename/@file_number - file_expr - Required - The file name or number for the 'slave' file. The program will search this file for the appropriate record when a new 'master' record is found.

slave_keyname/@key_number - key_expr - Required - The key name or number in the 'slave' file. The program will use this index in searching for the appropriate record when a new 'master' record is found.

master_filename/@file_number - file_expr - Optional - If a record is found in this file the program will attempt to find the slave records. If this value is not included the relation will be turned off.

master_field_list - fn/v1,f/c/e2,...,f/c/ex - Optional - The fields that contain the values to be used in searching for the 'slave' record.

COMMENTS

A maximum of 32 relations may be set up for all the programs. A list of all relations is kept and checked each time a record is found. A slave for one relation may be a master for another. However, you must be careful about creating an endless loop. For example, suppose the files in a master/slave relationship are also set with the original slave now the master, and the original master now the slave. When the master record (using the original relation) is found and the program finds the slave record, the slave will now act like a master and will find the slave (the original master), which will ... an endless loop.

Once a **REL** command has been executed it will remain active until the current program has been **QUIT**. If you want to 'turn off' the relationship then execute the command with the Slave file and key set properly but with the **MSTR** option (master file) blank.

Relations only affect the program in which they are set, whether or not the files are reopened (**ROPEN**) in a subsequent program.

If a file is cleared and it is a master, the slaves will be cleared also. However, slaves will not be automatically saved or deleted; you must do that explicitly within the program.

EXAMPLE

Assume you have a customer file and a invoice file. You want to find the first invoice record, if any, in the invoice file for any customer code entered by the user. The command to set up this relation might be:

REL *invoice* **KEY** *inv_cust_code* **MSTR** *customer* **FLDLST** *cust_code*

(This assumes that **inv_cust_code** is a keyname; that there is only one field in the index; and that the index is the same size and type as **cust_code**.)

This command would need to be executed only once, after the appropriate files have been opened. Each time a new customer is found the program will attempt to find the related record in the invoice file. If a record is not found, the appropriate error will be saved in the invoice file information buffer and can be checked, if needed, by the program through the use of the **FLERR()** function. You could also find a matching slave record explicitly by setting an equality between indexes and then doing a **FIND**. For example, to find an invoice record given the customer code key value, you might do the following:

```
inv_cust_code = cust_code
FIND record in INVOICE file
```

The major difference is that in using the **REL** command, the first find is event driven and can occur anytime within the program, regardless of how the master record is found.

PROGRAM EDITOR fiLe -> Find -> rElate

REMARK

This command allows the programmer to include a message or remark within the program that will have no effect on the program.

REM *statement*

statement - sac - Optional - Anything you want. This command is ignored by the compiler and is for your use only.

COMMENTS

If the remark is going to extend over several lines then the **REM** command (or appropriate symbol) must begin each line.

Several characters may be used instead of the **REM** command. The semicolon (;) or asterisk (*) may be used instead of **REM** so long as it is the first non-space character on the line. For remarks following a command on the same line, the programmer may use the double ampersand (&&). Some examples:

```
REM All of the following remarks are legal
;  This is a legal remark
*  This is also.
? 'TEST'                    && This is a legal remark after a command.
```

The programmer cannot put a remark on a command line if the command is continued. For example:

```
? 'THIS IS A TEST'+ \   && this is not a legal remark
        ' ALSO'         &&  this is a legal remark
```

A remark cannot be put on the continued line since the continuation character (\) must be the last character on the line. Please note that a continuation character is treated as such, even if it appears on a line after a remark character.

PROGRAM EDITOR System -> Programming -> Remark

REMOVE ARRAY

This command will deallocate an array previously allocated using the **ALLOC** (Allocate Field) command or **ALOCARY()** function. The memory used will be released to be used again, if desired.

REMVA *array_field_name*

array_field_name - *fn/v* - Required - The name of the field that had been previously allocated.

COMMENTS

This command will not remove any fields that have been defined in the original program, only those that have been allocated with the **ALOCARY()** function, or **ALLOC** (Allocate Field) command while running the program.

PROGRAM EDITOR Field -> Array -> Remove

RENAME FILE

Use this command to rename any file on the disk.

RENF *from* **TO** *to*

from - f/c/e - Required - The current file name including path and extension.

to - f/c/e - Required - The new file name including path and extension.

COMMENTS

Unlike the DOS REN command you can 'move' a file from one path to another by changing the path value in the **to** option.

PROGRAM EDITOR System -> File commands -> Rename file

REOPEN FILE

Use this command to 'attach' a file opened in a previous program to the current program.

ROPEN *file_number*

file_number - f/c/e - Required - This is the number of the file in the previous program. This value can be passed to a subsequent program through the use of the **reset** option in the **DEFINE** command or as a **PARAM** (Parameter). Once the subsequent program has the **file_number** value this command can be executed and the new program will be able to access the file completely and in its current state.

COMMENT

If a record was active when the new program was run then that data will be available. Any action taken by the subsequent program will still be there when it returns to the previous program. This includes any records deleted, added, etc. Also, if a file error occurs in the subsequent program and it returns to the previous program with no other actions to that file, the error will still be in effect and you will be able to test for it using the **FLERR()** function.

You may use any fields from a reopened file in a subsequent program just as though that file was opened in that program. The file, however, cannot be closed with the **CLOSE** command and will not be closed automatically when the subsequent program quits.

NOTE: You must use the file_number in this command. If the file was opened using the **OPEN** command and not the **OPENV** (Open Variable) command you can get that value with the **FNUM()** function.

PROGRAM EDITOR fiLe -> Open/close -> Reopen

SEE ALSO OPENV, FNUM()

REPLACE

The programmer can use this command to change the values in fields in a file in some or all records.

REPL *field_list* **WITH** *with_list MEM* **NUM** *number* **CNTR** *counter_field* **FILE** *filename/ @file_number* **KEY** *keyname/@key_number* **START** *start_value* **SCOPE** *scope scope_value* **FOR** *for_filter_expression* **WHILE** *while_filter_expression DISP*

field_list - fn/v1, fn/v2,..., fn/vx - Required - This is the list of fields whose values will be replaced in this command.

If the *MEM* option is set then the program assumes that each field is an array field. In this command you must use regular array fields. When used in this option you would use just the FIELD portion, with no array element specified. The program will automatically increment the array element number. You can specify the beginning array element number by setting this value in the **counter_field**.

A maximum of 80 fields/expressions/constants or any combination thereof may be included in a single **field_list** group.

with_list - f/c/e1, f/c/e2,..., f/c/ex - Required - These are the values that will be used as replacements.

MEM - Optional - If you are replacing from an array in memory instead of a standard file include this option in the command line.

number - fn/v - If the *MEM* option is included, this is Required - If *MEM* is specified then this is the number of elements in the array to be replaced.

counter_field - fn/v - Optional - This is an I type field that will be used for passing the number of records read, and the current array number, to your program. You can also use this to count the number of records replaced.

filename/@file_number - file_expr - Optional - The name or number of the file to be used. If you do not include this option, the program will look for a default search file set in the

SRCH (Search File) command. If a default file hasn't been previously set, the program will report an error and the command will be skipped. This entry will override any default value set in the **SRCH** command.

keyname/@key_number - key_expr - Optional - Set this to the appropriate value to read the records in the file in a particular order. If you do not include this option, the program will look for a default key specified in the **SRCH** (Search File) command. If a default key hasn't been previously set, the program will report an error and the command will be skipped. This entry will override any default value set in the **SRCH** command.

start_value - f/c/e1,f/c/e2,...,f/c/ex - Optional - The value to use as the beginning record. This is similar to doing a **FIND** before the command. If the index you're searching on has multiple segments you may list the proper values for each of the segments, separating them with commas. This will allow you to specify the exact type for each of the segments. For example, suppose you're searching on an index that has two segments: a 5 character alpha and an I type field. Each record you want has the same alpha but the I field will change, and is in order. In the first record the I type field has a value of 0. The following would find the first record for that group, if it exists:

start 'ABCDE',0!

Notice that we separate the values with a comma. The only requirement is that the values be of the same type (and size) as the segments in the key. In this case we put the I type constant specifier (!) after the 0 to make sure that it is passed as an I type field.

scope - Optional - This option may help determine how many records are changed. For more information about the **scope** specifier please see the general information at the beginning of this chapter.

scope_value - f/c/e - Optional - If **scope** is set to **N** or **F** this is the number of records to be changed.

for_filter_expression - lexpr - Optional - You would use this option to restrict the records changed in this command. If the expression did not resolve to .T. the record would not be changed. In this option, the search continues until the program reaches the end of file and then will quit. You can also use this option to perform the same task if *MEM* is specified. Use the **counter_field** as the array specifier in the expression.

while_filter_expression - lexpr - Optional - This option also restricts the records changed. However, the first time the expression resolves to .F., the program stops reading records and continues to the next command. Therefore, if the program is reading records in a certain order (by a specific key) and the expression returns .F., then the program knows that all the appropriate records have been read and there is no need to continue reading. For example: suppose you want to replace some value in all the invoice records for a specific customer. The first record for that customer is found in the invoice file either by using the **start_value** option within this command, or through the **FIND** command before executing this command. Then the **while_filter_expression** would be:

INV_CUST_CODE = CUSTOMER_CODE

(This assumes that there is a field called INV_CUST_CODE in the invoice file and a similar field in the customer file called CUSTOMER_CODE. The **keyname/@key_number** option must be set to the proper value so that the records are read in

CUSTOMER_CODE order, or you must have set the default key using **SRCH** previous to this command.) When the first invoice record for the next customer is read the program will stop changing records.

NOTE: If there is no **start_value** you must set the **scope** value to *R* or *N xxx*. If this is not done, the program will find the first record in the file and the **while_filter_expression** option will probably fail the first time. However, if the **start_value** option is used you can ignore this requirement.

DISP - Optional - Each time a record is read the program will redisplay the screen fields, if you specify this option. This will slow down the operation of this command, with the amount of slow-down depending on how many fields are displayed on the screen. The net effect will probably be negligible unless you are reading through a very large file.

COMMENTS

This command provides the programmer with an easy and quick method to replace the values in a group of fields from a file or in a memory array. Any file, TAS or non-TAS, may be used.

PROGRAM EDITOR fiLe -> Mult rec cmds -> rePlace

SAMPLE PROGRAM REPLTEST

RESET SCREEN

This command removes the effects of any windows and mounted screen fields from the screen.

RSCR

No options

PROGRAM EDITOR User interface -> Screen control -> reset screeN

RETURN

The programmer would use this command to end a subroutine called with the **GOSUB** command, or a UDF/UDC.

RET *return_value*

return_value - f/c/e - Required - If this is a return from a UDF the programmer may specify a value to be returned. This would be the value passed by the function. For example:

```
X = USER_FUNC()
QUIT


FUNC USER_FUNC
        RET 1
```

The value of X will be set to 1 in the example above.

A value can only be returned in the case of a UDF. In the case of a UDC or a return from a subroutine called through the **GOSUB** command the **RET** should not be followed by a value. In any case it will have no effect.

PROGRAM EDITOR prg Control -> Goto/gosub -> Ret

SEE ALSO GOSUB, UDC, UDF

REVERSE (*DOS only*)

This will change the display color to the color you have set as reverse.

REV

No options

PROGRAM EDITOR User interface -> Color control -> Reverse

SEE ALSO COLOR, BKG, FRG, CC(), CCE(), CCF(), CCH(), CCR()

REWRAP

This command will re-word wrap a field that has been previously set up with the WRAP command.

REWRAP *starting_line_number*

starting_line_number - f/c/e - Required - The line number at which to start the word wrap process.

COMMENTS

By starting at a different line than the first you can speed up the word wrap process. Also, the program already knows what the field is, where it's located, and has done all the preliminary work which also speeds up the process.

PROGRAM EDITOR User interface -> rewrAp

SEE ALSO WRAP, WRAP(), WRAPO(), WRAPL(), WRAPS()

ROW COLOR (*Windows only*)

Use this command to change the color of a single row on the screen..

ROW_COLOR FROM *from_row* **THRU** *thru_row* **COLOR** *background_color*
TEXT_COLOR *text_color*

from_row - f/c/e - Required - The starting row number. The first row in a window is number 1. This value will always be related to the current active window.

thru_row - f/c/e - Required - The ending row number. If you only want to change the color in a single row then this should be the same value as the FROM value.

background_color - f/c/e - Required - The background color for the chosen row(s). If you don't enter a value here the program will use the current background color for the first row chosen..

text_color - f/c/e - Optional - The color for the text in the chosen row(s). If you don't enter a value here the program will use the current text color for the first row chosen..

COMMENTS

Due to the way text is displayed in Windows and colors are set you can only change the color for an entire row. However, if you have a window active the row would be within that window and wouldn't affect other lines outside that window. For more information on colors and windows please refer to **Chapter 11 - Windows Programming**.

PROGRAM EDITOR Win Commands -> Row Color

RUN

This is a TAS Professional 3.0 command here for compatibility. The preferred method is to use the **CHAIN** command.

RUN *program_name*

program_name - f/c/e - Required - This is the name of the TAS Professional 5.1 program to be run. It must have been previously compiled. You can specify the path if desired, otherwise, it should reside in your current sub-directory.

COMMENTS

This is the equivalent of the TAS Professional 3.0 command Run.

PROGRAM EDITOR 3.0 Commands -> N - Run

RUN ANYTIME PROGRAM

This command will put the name of a program into memory so that it can be executed later by a user who presses a specific key.

RAP *program_name* **NUM** *number* **WITH** *with*

program_name - f/c/e - Required - The name of the program to run. This needs to include the path also, if any.

number - f/c/e - Required - The RAP number to be assigned. May be from 1 to 10 and corresponds to the keys ALT-1 through 10. For example, if **number** is 5, the user would press the ALT key plus the 5 key at the same time to run the program.

with - fn/v1,fn/v2,...,fn/vx - Optional - A list of field values to be passed to the program. The receiving program can test for these with the **PARAM** (Parameters) command.

NOTE: Do not use constants or expressions in this command since TAS Pro 5.1 clears that information out before chaining to the next program. You can, however, pass pointers (P type only) to the receiving program so that you can access a value

directly in the previous program. Also, see the **reset** option in the **DEFINE** command for another way of doing this.

COMMENTS

The original program, upon returning from the program run with the **RAP** command, will continue execution at the same line. Contrast this with the standard **CHAIN** command, which continues with the next line. All data files previously opened will still be open, and all data will be available (except for data solely defined in the program called using the **RAP** command).

The only limit to the depth of chaining is available memory. To help alleviate the use of memory TAS Pro 5.1 saves only the data that is specific to this current execution. The portions of the program that remain constant are reloaded, automatically, when you return. If you have expanded memory, and an appropriate memory manager, TAS Pro 5.1 will save some of this information to that memory, which will free up even more memory in the lower 640k. You can determine the minimum amount of memory to be used for a program by setting the **General Buffer Size** in the TAS50.OVL file. For more information please refer to **Chapter 3, Main Menu - Utilities** and the details about the **Set Configuration** process.

Each program when executed uses only that memory which is absolutely required to run. This can be as little as 5 or 10k or as much as the remaining memory in the system. Each program, when executed, is absolutely independent from the program that chained to it, unless you desire to pass data or files. Each program can have the maximum number of fields, line labels, etc. However, the limitation on the maximum number of files that may be opened applies system wide, no matter how many or how few programs are executed.

A **RAP** program is executed when the user presses the ALT and 1 through 0 key at the same time (0 corresponds to buffer number 10). The **CHAINR** (Chain Rap) command can also be used. When the **RAP** program quits it will return to the same line in the original program.

The **RAP** programs are maintained in a central buffer, so only 10 can be saved for any group of programs and they can, in turn, be executed from any subsequent or previous program once they have been loaded.

You can access a **RAP** program during any **ENTER, LISTM** (List Array)**, LISTF** (List File) or **NMENU** command.

PROGRAM EDITOR

System -> Other programs -> rAp

SAMPLE PROGRAMS

RAPTST, RAPTST2

SEE ALSO

CHAINR

SAVE RECORD

This command is used to save records to any file, TAS or non-TAS.

SAVE *filename/@file_number NOCNF NOCLR* **GOTO** *goto_label* **ERR** *error_lbl*

filename/@file_number - file_expr - Required - This is the name or number of the file that will receive the record.

NOCNF - Optional - The normal process is for the program to confirm that it is okay to save the record. If you do not want the program to confirm, include this option in the

command line. In that case the program will just save the record without any indication to the user.

NOCLR - Optional - Normally, after TAS Professional 5.1 saves a record it automatically clears the record and record number buffers. This makes sure that everything is ready to create a new record. However, if you include this option in the command line you will leave both the record buffer and number active. If your desire is to use the data already entered as the default for the next record, you can use the **CLR** (Clear Record Buffer) command (**record** option). Make whatever changes are necessary to the data, and when the next **SAVE** command is executed for this particular file, it will save the data as a new record and not just overwrite the last record saved.

goto_label - label - Optional - This is the line you want to transfer control to if the user answers the confirm question with an **N**. If you do not specify a label here the program will continue with the line directly after this **SAVE** command.

error_lbl - label - Optional - If you specify a label here, the program will transfer control to the appropriate line if an error occurs during the command. If you specify NO_ERR as the label name, then no error will be displayed if one occurs. You can still test for an error after the command using the **FLERR()** function.

COMMENTS

A major feature of TAS Professional 5.1 is the ability to treat non-TAS files as though they were TAS files. This is true here also. If a record is not active for a non-TAS file the program will automatically append it to the end of the file. Also, you can use the **FINDV** (Find Variable) command to get a record, make changes to it, and save it back to the same location within the file.

If you save a record with the *NOCLR* option set and the file was opened with **lock R** (record locks and multi-user is set to **Y** in setup file) the program will attempt to immediately get the record again after it is saved. This is done to assure that the locks have been properly set. A situation could occur where another user is attempting to read the same record. However, if you have set the cycle time between retries high enough (the default setting is 32), it is very likely that the record will be re-read before the other user attempts another find. If the other user does intercept the record, the user who saved the record will be locked until the new user releases the record. The first user is now under all the same restrictions as any user trying to get a locked record, including, if the **TRAP RLCK** is not set, a message telling him/her that the record is locked by another user etc. In all situations the record has been safely saved before the lock is released and even if the second user makes changes those changes will be part of the record when it is re-read by the first user.

PROGRAM EDITOR

fiLe -> Save

SAMPLE PROGRAM

EXPERRFL, EXPERRFX

SEE ALSO

OPEN, OPENV

SAVE SCREEN

This command will save a screen so that it can later be redisplayed using the **REDSP** (Redisplay Screen) command.

SAVES *screen_holder_name*

screen_holder_name - fn/v - Optional - The name of the field that is to be used for saving the screen. If you did not specify a field name in this command, the program will allocate memory space for a temporary buffer to hold the screen and the appropriate information. In this case, when you execute the **REDSP** (Redisplay Screen) command you would leave out the field name there also.

NOTE: If you do specify a field name it must be large enough to hold the entire screen and certain information about the current screen environment. This is approximately 4300 bytes.

COMMENTS

If you allow the program to save and redisplay the screen from an internal buffer you need to be sure that you balance your **SAVES** and **REDSP** (Redisplay Screen) commands. Each time the program saves a screen to an internal buffer it allocates the memory required to do so. If you don't redisplay the screens and continue to save them you will eventually run out of memory. This does not apply to the situation where you save the screen to a named field.

When you **QUIT** the program any buffers used for saving screens are automatically released.

PROGRAM EDITOR

User interface -> Windows -> Save scrn

SEE ALSO

REDSP

SAVE SCREEN 3.0

This is a TAS Professional 3.0 command here for compatibility. The preferred method is to use the **SAVES** command.

SAVES3 *screen_buff_num*

screen_buff_num - f/c/e - Required - This is the screen buffer number. This must resolve to one of the following values:

1-10 - Save entire screen including active windows
21-30 - Save characters and color attributes only
41-50 - Save characters only

COMMENTS

This is the equivalent of the TAS Professional 3.0 command Save Screen.

PROGRAM EDITOR

3.0 Commands -> O - Saves3

SAY

This command will add a field/expression to the screen buffer. This would have the same effect as though it were included in the original **MOUNT** command.

SAY *fieldname* **AT** *starting_column,row* **COLOR** *color* **PICT** *picture*

fieldname - fn/v/e - Required - The field/expression to be added to the screen buffer.

starting_column - f/c/e - Optional - The number of the column at which to start the display of the **SAY** field. The first column on the screen (far left position) is number 1. The default value, if none is provided, is the current column value as set by the last display or enter to the screen. This value can be easily determined through the use of the **COL()** function.

row - f/c/e - Optional - The row or line number at which to display the **SAY** field. The first row on the screen (top) is number 1. The default value, if none is provided, is the current row value as set by the last display or enter to the screen. This value can be easily determined through the use of the **ROW()** function.

color - f/c/e - Optional - This is the **color** value to be used when displaying the field or expression in **fieldname**.

picture - f/c/e - Optional - This is a **picture** value to be used when displaying or entering the field.

COMMENTS

You should **MOUNT** some sort of a screen before using this command, even if the screen is blank. You can create a blank screen by running the Screen Painter and then, when the initial screen is displayed, pressing ESC and saving it at that time. If you have the Source Code to the Utilities a screen format named CLR_SCR.SCP was included with your programs. This is a blank screen.

PROGRAM EDITOR User interface -> Screen control -> saY

SAMPLE PROGRAM SAYTEST

SEE ALSO MOUNT, CLRSF

SCAN

This command combines the work of a **WHILE/ENDW** loop and the **FIND** command into one super command. This is actually the starting point of a control structure.

SCAN *filename/@file_number* **KEY** *keyname/@key_number* **START** *start_value* **SCOPE** *scope scope_value* **FOR** *for_filter_expression* **WHILE** *while_filter_expression DISP NLOCK REV*

filename/@file_number - file_expr - Optional - The name or number of the file to be used. If you do not include this option, the program will look for a default search file set in the **SRCH** (Search File) command. If a default file hasn't been previously set the program will report an error and the command will be skipped. This entry will override any default value set in the **SRCH** command.

keyname/@key_number - key_expr - Optional - Set this to the appropriate value to read the records in the file in a particular correct order. If you do not include this option, the program will look for a default key specified in the **SRCH** (Search File) command. If a default key hasn't been previously set the program will report an error and the command will be skipped. This entry will override any default value set in the **SRCH** command.

start_value - f/c/e1,f/c/e2,...,f/c/ex - Optional - The value to use as the beginning record. This is similar to doing a **FIND** before the command. If the index you're searching on has multiple segments you may list the proper values for each of the segments, separating them with commas. This will allow you to specify the exact type for each of the segments. For example, suppose you're searching on an index that has two segments: a 5 character alpha and an I type field. Each record you want has the same alpha but the I field will change, and is in order. In the first record the I type field has a value of 0. The following would find the first record for that group, if it exists:

start 'ABCDE',0!

Notice that we separate the values with a comma. The only requirement is that the values be of the same type (and size) as the segments in the key. In this case we put the I type constant specifier (!) after the 0 to make sure that it is passed as an I type field.

scope - Optional - This option may help determine how many records are scanned. For more information about the **scope** specifier please see the general information at the beginning of this chapter.

scope_value - f/c/e - Optional - If **scope** is set to **N** or **F** this is the number of records to scan.

for_filter_expression - lexpr - Optional - You would use this option to restrict the records scanned in this command. If the expression did not resolve to .T. the search would continue until the program reached the end of file and then would quit.

while_filter_expression - lexpr - Optional - This option also restricts the records scanned. However, the first time the expression resolves to .F., the program stops reading records and continues to the command immediately following the corresponding **ENDS** (Endscan). Therefore, if the program is reading records in a certain order (by a specific key) and the expression returns .F., then the program knows that all the appropriate records have been read and there is no need to continue reading. For example: suppose you want to find/read all the invoice records for a specific customer. The first record for that customer is found in the invoice file either by using the **start_value** option within this command, or through the **FIND** command before executing this command. Then the **while_filter_expression** would be:

INV_CUST_CODE = CUSTOMER_CODE

(This assumes that there is a field called INV_CUST_CODE in the invoice file and a similar field in the customer file called CUSTOMER_CODE. The **keyname/@key_number** option must be set to the proper value so that the records are read in CUSTOMER_CODE order, or you must have set the default key using **SRCH** previous to this command.) When the first invoice record for the next customer is read the program will stop reading records.

NOTE: If there is no **start_value** you must set the **scope** value to *R* or *N xxx*. If this is not done, the program will find the first record in the file and the

while_filter_expression option will probably fail the first time. However, if the **start_value** option is used you can ignore this requirement.

DISP - Optional - Each time a record is read the program will redisplay the screen fields, if you specify this option. This will slow down the operation of this command, with the amount of slow-down depending on how many fields are displayed on the screen. The net effect will probably be negligible unless you are reading through a very large file.

NLOCK - Optional - If this option is included in the command the program will NOT lock the record upon finding it as it would normally do in a multi-user situation. The default operation is to place a lock on the record upon reading it.

REV - Optional - If this option is included, the **SCAN** will be performed in reverse order.

COMMENTS

This command, together with the other parts of the control structure, could easily become your most used command when accessing records in any file. With just two commands, **SCAN** and **ENDS** (Endscan) you can create a loop that will automatically provide to the commands within that loop just the records necessary for the operation and no others. When the last record in the file, or the last appropriate record, has been read, the program will automatically transfer control to the line after the appropriate **ENDS** command and continue with the program. The **SLOOP**, and **SLOOP_IF** commands will transfer control back to the beginning of the loop and the next record will be found without any other Find command! This single structure can replace many lines of code in your programs.

The Report Writer uses this command to a large extent when creating code for multiple record reports. If you are interested in learning more, just generate a report and take a look at the code.

PROGRAM EDITOR

fiLe -> Mult rec cmds -> Scan -> Scan

SEE ALSO

SEXIT, SEXIT_IF, SLOOP, SLOOP_IF, ENDS

SCREEN CHARACTER (*DOS only*)

Use this command to get or put a single display character or attribute on the screen.

SCRCHR AT *column,row* **DISP** *display_field* **ATRB** *attribute_field GET/SET*

column - f/c/e - Required - The column position on the screen for the character. The upper left corner of the active window (or the full screen) is column 1, row 1.

row - f/c/e - Required - The row position on the screen for the character. The upper left corner of the active window (or the full screen) is column 1, row 1.

display_field - fn/v - Required - The name of the field that either holds the character to be displayed (set) or will receive the character currently on the screen (get). This must be a 1 character alpha (type A) field.

attribute_field - fn/v - Required - The name of the field that either holds the color attribute to be displayed (set) or will receive the color attribute currently on the screen (get). This must be a 1 character alpha (type A) field.

GET/SET - Required - *GET* will put the character/attribute on the screen at the column,row position into the appropriate fields. *SET* will take the current values in the fields and move them to the screen.

PROGRAM EDITOR User interface -> Screen control -> screen cHr

SCREEN CONTROL

This command will allow you to lock, unlock, refresh, and reset the screen.

SCRN *what_to_do*

what_to_do - LURS - **L** - lock the screen; all display to the screen will be stopped until the unlock or refresh option is run. **U** - unlock the screen; DO NOT refresh any screen fields currently displayed. **R** - refresh any fields currently displayed on the screen. **S** - reset the screen, and eliminate the effect of the current window.

PROGRAM EDITOR User interface -> Screen control -> screen L/u/r/s

SCROLL

This command will move a set of characters on the screen as a block.

SCROLL AT *starting_column,starting_row* **LEN** *number_of_rows*
WDT *number_of_columns* **NUM** *how_many_to_move*
UP/DN/LEFT/RIGHT

starting_column - f/c/e - Required - The upper left column value. Together with the **starting_row** value defines the upper left corner of the block. The upper left corner of any window is 1,1.

starting_row - f/c/e - Required - The upper left row value. Together with the **starting_column** value defines the upper left corner of the block. The upper left corner of any window is 1,1.

number_of_rows - f/c/e - Optional - The number of rows in the block. If this is not included the block will be 1 row long.

number_of_columns - f/c/e - Optional - The number of columns in the block. If this is not included the block will be 1 column wide.

how_many_to_move - f/c/e - Optional - How many rows/columns to move the block of characters. If this is not included the program will clear the entire block.

UP/DN/LEFT/RIGHT - Required - Which direction to move the block of characters.

COMMENTS

This command will not move the background colors, only the characters displayed on the screen.

PROGRAM EDITOR User interface -> Screen control -> Scroll

SAMPLE PROGRAM SCRLTEST

SEARCH FILE

This command will set the default search file and key.

SRCH *filename/@file_number* **KEY** *keyname/@key_number*

filename/@file_number - file_expr - Required - This is the name or number of the file to be used.

keyname/@key_number - key_expr - Required - Set this to the appropriate value to read the records in the file in a particular order.

If you do not want to specify any key but want to search sequentially through the file, set the null key. For example:

KEY @0 (0 is the number zero and not the letter 'O')

COMMENTS

By using this command the programmer can set the default values to be used by any other command that accesses data in files. This also includes the user operated file keys (F9 - Find, F5- First, F6 - Last, etc.). If default values have not been set and the cursor is not on a key field that has the same name as a key, the user will not be able to use the file keys. However, if you set the defaults, the program will use the values for file key searches no matter where the cursor is. Using this process you can make sure that only the file and key you want is used for searching. For more information on the file keys please refer to **Chapter 1, Installation and General Information**.

To reset the **SRCH** value execute the command without a file or key specified. For example:

SRCH

This will 'turn off' this feature and will use the current field for searching, if it's a key, or will not allow keyboard searches if it isn't.

PROGRAM EDITOR fiLe -> Find -> Search file

SELECT

This is a process control command, i.e., it will control whether a command, or group of commands will be executed. This is one part of a complete command structure. This command is the first part of a **CASE** structure.

SELECT *expression*

expression - f/c/e - Required - This must resolve to an integer type value that will be used as the comparison value by each of the subsequent **CASE** commands.

COMMENTS

For more information on the different structured programming commands, and the **SELECT** command in particular, please see **Chapter 7, Structured Programming Commands.**

If you are missing an **ENDC** (End Case) command in a **SELECT/CASE** structure you will get the If without Endif error message during the compile process.

PROGRAM EDITOR Prg Control -> Case -> Select

SEE ALSO CASE, OTHERWISE, ENDC

SET ACTIVE

If you have two or more TAS Pro 5.1 files that all use the same buffer (i.e., the same file descriptor name or schema), and you need to switch from one to the other when searching for a particular record (**FIND** command), then you need to use this command to make the switch.

SETACT *file_descriptor_name* **FILE** *file_name*

file_descriptor_name - sac - Required - The FD name (or schema for 3.0 users) for the files to be switched. The name can be a maximum of 8 characters. Must be the actual FD name.

file_name - sac - Required - The name of the file that needs to be **SETACT**. The name can be a maximum of 8 characters. Must be the actual file name.

COMMENTS

We recommend that you use the **OPENV** (Open Variable) and **FINDV** (Find Variable) commands which obviate the need for this command. However, the **SETACT** command is included here for compatibility with TAS Pro 3.0.

PROGRAM EDITOR fiLe -> Open/close -> Set active

SET LINE

This command will set up the fields to be used in a **LISTF**/**LISTM** command as a First Line (FLINE).

SETLINE *recv_fld* **WITH** *title_list*

recv_fld - fn/v - Required - The name of the field that will be used to hold the title list.

title_list - f/c/e,f/c/e,...f/c/e - Required - The list of fields, constants, and/or expressions that will make up this title line.

COMMENTS

This command will only work in conjunction with the LISTF command.

PROGRAM EDITOR fiLe -> Write

SEE ALSO LISTF

SEXIT

This is a process control command, i.e., it will exit from a **SCAN** loop. This is one part of a complete command structure.

SEXIT

No options

COMMENTS

For more information on the different structured programming commands, and the **SEXIT** command in particular, please see **Chapter 7, Structured Programming Commands.**

PROGRAM EDITOR fiLe -> Mult rec cmds -> Scan -> eXit

SEE ALSO SCAN, SEXIT_IF, SLOOP, SLOOP_IF, ENDS

SEXIT_IF

This is a process control command, i.e., it will control whether to exit from a **SCAN** loop. This is one part of a complete command structure.

SEXIT_IF *expression*

expression - lexpr - Required - If the **expression** resolves to .T. then the program will transfer control to the line immediately following the appropriate **ENDS** (Endscan) command.

COMMENTS

For more information on the different structured programming commands, and the **SEXIT_IF** command in particular, please see **Chapter 7, Structured Programming Commands.**

PROGRAM EDITOR fiLe -> Mult rec cmds -> Scan -> exiT_if

SEE ALSO SCAN, SEXIT, SLOOP, SLOOP_IF, ENDS

SLOOP

This is a process control command, i.e., it will loop back to the beginning in a **SCAN/ENDS** loop. This is one part of a complete command structure.

SLOOP

No options

COMMENTS

For more information on the different structured programming commands, and the **SLOOP** command in particular, please see **Chapter 7, Structured Programming Commands.**

PROGRAM EDITOR fiLe -> Mult rec cmds -> Scan -> Loop

SEE ALSO SCAN, SEXIT, SEXIT_IF, SLOOP_IF, ENDS

SLOOP_IF

This is a process control command, i.e., it will control whether to loop back to the beginning in a **SCAN/ENDS** loop. This is one part of a complete command structure.

SLOOP_IF *expression*

expression - lexpr - Required - If the expression resolves to .T. then the program will transfer control to the appropriate **SCAN** line. This would be equivalent to executing the **ENDS** (Endscan) command.

COMMENTS

For more information on the different structured programming commands, and the **SLOOP_IF** command in particular, please see **Chapter 7, Structured Programming Commands.**

PROGRAM EDITOR fiLe -> Mult rec cmds -> Scan -> loop_If

SEE ALSO SCAN, SEXIT, SEXIT_IF, SLOOP, ENDS

SORT ARRAY

This command will sort an array of fields in ascending or descending order. You can combine sort fields and can also move associated fields, even though they aren't part of the sort field.

SORTA *sort_field_expr* **NUM** *number_of_elements* **MOVE** *move_list* **CNTR** *counter_field ASC/DSC*

sort_field_expr - fn/v/e - Required - The field to be sorted. This can also be an expression containing several fields. The **counter_field** must be used as the array element specifier in the field or expression. For example:

cust_state[cntr]+cust_zip[cntr]
or
cust_state[cntr]

number_of_elements - f/c/e - Required - The number of elements to sort. The actual maximum number of array elements must be at least **number_of_elements**+1.

move_list - fn/v1, fn/v2,..., fn/vx - Required - A list of array fields to be moved along with the field to be sorted. You must include at least the sort field itself. You may also include any other array fields that you want to be ordered in the same fashion as the sort field. The **counter_field** must be specified as the array element in these fields. For example:

cust_state[cntr],cust_zip[cntr],cust_sales[cntr],etc...

counter_field - fn/v - Required - This is the field that you will use in the command as the array specifier. It must be of type I.

ASC/DSC - Optional - Which way to sort. Use *ASC* for ascending (names and numbers get larger as the array element number increases) or *DSC* for descending (names and numbers get smaller as array element number increases).

PROGRAM EDITOR Field -> Array -> Sort

SAMPLE PROGRAM SORTEST

SEE ALSO RDA, WRTA

SORT 3.0

This is a TAS Professional 3.0 command here for compatibility. The preferred TAS Professional 5.1 command is **SORT ARRAY**.

SORT3 *mem_area#* **FLD** *sort_expression* **SIZE** *size_of_sort_expr* **NUM** *#_flds_to_sort* **DO***what_to_do*

mem_area# - f/c/e - Required - This is the memory area to use. This must resolve to a value of 1 through 4.

sort_expression - f/c/e - Required - This resolves to the value to sort.

size_of_sort_expr - f/c/e - Required - The internal size of the value to sort.

num_flds_to_sort - f/c/e - Required - This is the maximum number of entries that will be made to the sort buffer. This is only required when initializing the sort buffer at the beginning of the process.

chg_value - f/c/e - Required - The amount to add to or subtract from each pointer. NOTE: The size and type of this field must be the same as the pointer.

what_to_do - f/c/e - Required - This must resolve to either 0 (initialize the sort buffer) or 1 (insert a new item into the buffer).

COMMENTS

This is the equivalent of the TAS Professional 3.0 command Sort.

PROGRAM EDITOR 3.0 Commands -> P - Sort3

SOUND

This command will use an array of notes and beats (number of clock ticks to keep sound on) to make noise come out of the PC speaker.

SOUND *note_array* **BEAT** *beat_array* **NUM** *num_of_notes* **FILE** *wav_file*

note_array - Required - An array of values that represent notes. Each note is the number 1,193,180/frequency desired. Must be an I type array.

beat_array - Required - An array of beat times. Each element corresponds to the same element in **note_array**. A beat is 1/18 of a second (18 clock ticks per second). You must have at least the same number of elements in this array as in the **note_array**. To put wait beats after the note multiply the number of waits (each is again 1/18 of a second) by 256 and add it to the **beat_array** number.
For example, if you want a note to sound for 1/3 of a second with a wait (pause after the note before the next is played) of another 1/3 of a second the value to be placed in the appropriate **beat_array** element would be 6 + (256*6) or 1542.

num_of_notes - The number of notes to play (active array elements).

wav_file - ***Windows Only*** - f/c/e - Required in Windows - In Windows the only way to get noise out of your computer is to 'play' a .WAV (sound) file. The file will play through your sound card if you have one installed. You must provide the entire file name including path and .WAV extension. NOTE: Even though the **BEAT** and **NUM** options are ignored you must still provide a dummy field for the ***note_array*** value or the command will not compile.

PROGRAM EDITOR System -> Sound

SAMPLE PROGRAM PLAYSNG, WINTEST ***(Windows Only)***

SET SPECIAL FILE NUMBER

This is a TAS Professional 3.0 command here for compatibility. The preferred method is to use the **OPENV/FINDV** commands.

SSPCF *non_TAS_file_num*

non_TAS_file_num - f/c/e - Required - This is the number of the non-TAS file opened with the **OPNO** command. This must resolve to a value of 1 through 3. The files are numbered in the order they are opened.

COMMENTS

This is the equivalent of the TAS Professional 3.0 command Set Special File Num.

PROGRAM EDITOR 3.0 Commands -> Q - Sspcf

TIME

Get the current system time, or reset it to a new value.

TIME *time_field GET/SET*

time_field - fn/v - Required - The T type field that will either receive the current time (*GET*) or contain the value to be used in setting the time (*SET*).

GET/SET - Required - *GET* means to obtain the current time and put it into **time_field**. *SET* means to set the time using the value in **time_field**.

COMMENTS

This command will affect just the workstation it is running on and no others if you are running on a network. This has the same effect on the computer as if you typed in DATE at the DOS prompt and entered the value.

PROGRAM EDITOR

System -> Date/time -> Time

TRACE (*DOS only*)

Use this command to execute the trace routine when you are debugging a program.

TRACE *what_to_do* **VALUE** *prg_fld* **BRK** *break_routine_label*

what_to_do - sac - Required - What you want the trace routine to do. The options are: *STEP* - turn on the single line step option. This will cause the program to stop at each line and if you have compiled the program with the -D (debug) option the actual line from the source file will be displayed at the bottom of the screen. You can then look at field values, set other break points, etc. *PRG* - Set the next break point for a specific location in the program. This is better done using the option in the actual trace routine once it has been executed since you need the actual location in the program of the line you want to break on. *FIELD* - By naming a field the program will break at the line the field is accessed, either read or updated. *OFF* - Turn off the trace routine. You can also use this at the beginning to just set up the routine. Then during program operation you can press the ^B (ctrl+B) key and execute the trace routine at any point in the program.

prg_fld - f/c/e - Required - If you have specified *PRG* or *FIELD* you must put in the appropriate value here.

break_routine_label - label - Required - The first time you execute this command you must tell the program the location of the trace routine. You do this by putting the line label name here. This will be **BREAK_ROUTINE** if you are using the standard trace routine (BREAK.SRC) from Business Tools Incorporated. You need only do this once; however, it won't harm anything if you include it with every command.

COMMENTS

The trace routine is actually a separate program that must be included in any program you wish to debug using this command. The standard program provided with this version of TAS Professional 5.1 is BREAK.SRC. You include it in any program by putting the following line in the main program:

```
#inc break
```

If you plan to distribute your program, make sure you remove this line, and not just because it takes up extra room. **You may not include this routine in any program not being run at the same site that holds the license for the compiler.**

When this command is executed the program will display the trace window at the top of the screen and, if you have compiled the program with the -D (debug) flag, it will also display the actual source code line at the bottom. You can enter any expression and the program will display the result immediately below the entry line. Press the F2 key and a list of fields, along with their current values, will be displayed. Press the ^R key and the full screen will be displayed without any trace info; press any key to return to the trace box.

Using this option you will be able to quickly and easily find annoying bugs in your code.

PROGRAM EDITOR System -> Programming -> Trace

TRANSACTION

Using this command you can be assured that an entire set of file updates will be completed. Otherwise, they can be rolled back to an original state.

TRANSX *what_to_do* **ERR** *error_field*

what_to_do - BCR - Required - **B** - Begin; this option is used at the beginning of the transaction. **C** - Commit; will finalize the transaction and permanently update the files. **R** - Rollback; will return the files to the state before the original Begin command.

error_field - fn/v - Required -You must provide a field that will be used to receive any error number provided by the routine. If the command step is successful the value returned is 0. This field must be of type I.

COMMENTS

The correct transaction process is:

```
TRANSX B            && begin
    ...
    all file updates done here
    ...
TRANSX C            && commit
```

However, if an error occurs while the file updates are being accomplished (between the **TRANSX** *b* and **TRANSX** *c* steps) you would use the command:

```
TRANSX R            && rollback
```

and any updates that were completed after the **TRANSX** *b* but before the **TRANSX** *c* would be undone.

PROGRAM EDITOR fiLe -> Transactions

TRAP

This command is used to set internal traps so that if a certain key is pressed the program will take an appropriate action.

TRAP *trap_name_list GOTO/GOSUB/IGNR/DFLT label*

trap_name_list - sac - Required - The name of the trap. These are listed below. You may include multiple trap names for a single trap command. Each of the traps will be set to the same values. This is generally used when you are setting a group of traps to *IGNR* (ignore) or *DFLT* (default).

GOTO/GOSUB/IGNR/DFLT - Required - The options are: *DFLT* - Default; do whatever is normally defined for that key. *IGNR* - Ignore; don't do anything when the user presses that key. *GOTO* - transfer control to the line label indicated when the user presses that key. *GOSUB* - transfer control to the line label indicated (typically the beginning of a subroutine) when the user presses that key; you need to make sure the routine is terminated with a **RET**.

label - label - Required if *GOTO* or *GOSUB* specified - You must specify the label name so the program will know where to transfer control.

TRAP NAME	What it does
F1 - F10	Executes trap if user presses key
SF1 - SF10	Executes trap if user presses SHIFT & key
CTL_F1-CTL_F10	Executes trap if user presses CTRL & key
CTL_PG_UP	Executes trap if user presses CTRL & key
CTL_PG_DN	Executes trap if user presses CTRL & key
ALT_F1-ALT_F10	Executes trap if user presses ALT & key
CTL_A - CTL_Z	Executes trap if user presses CTRL & key
ALT_A - ALT_Z	Executes trap if user presses ALT & key
ESC	Executes trap if user presses Escape key during entry
INT	Executes trap if user presses Escape key during program operation, not at an entry
T_ESC	Executes trap if user presses Escape key during entry. Temporary and will override the ESC trap. This is automatically cleared after any **ENTER** or **RETURN**.
UPAR	Executes trap if user presses up arrow (^) key
DNAR	Executes trap if user presses down arrow key
LT_A	Executes trap if user presses left arrow (<-) key
RT_A	Executes trap if user presses right arrow (->) key
LT_A_AS	Executes trap if user is at the beginning of the field and presses the left arrow key
RT_A_AS	Executes trap if user is at the end of the field and presses the right arrow key
HOME	Executes trap if user presses HOME key
END	Executes trap if user presses END key
PG_UP	Executes trap if user presses Page Up key
PG_DN	Executes trap if user presses Page Down key
INSRT	Executes trap if user presses Insert key
DEL_KEY	Executes trap if user presses DEL or Delete key
WD_LT	Executes trap if user presses the CTRL (control) and left arrow (<-) keys at the same time
WD_RT	Executes trap if user presses the CTRL (control) and right arrow (->) keys at the same time
TAB	Executes trap if user presses the TAB key
BCK_TAB	Executes trap if user presses the SHIFT and TAB key at the same time
RSRCH	Executes if user searches for a record in a file using either the default system keys or a FINDx command. NOTE: This trap cannot use the **GOTO** option. If it is not **DFLT** or **IGNR** (both have same effect) then it must be **GOSUB**. Even if you set the do what option to **GOTO** it will still be treated as a **GOSUB**.
RLCK	Executes trap if the program attempts to read a record that is locked (it has been read by another user's program) or if

	a file is attempted to be opened that is locked in full by another program. For more information see below.
L_EXIT	Executes trap if record is locked and user answers **N** (No) to system's offer to retry. See below.
FERR	Executes trap if an error occurs while reading a file. This may not have any effect in a command that accesses a complete file, such as **DELETE_ALL**, since the command will exit normally once an error occurs.
PERR	Executes trap if an error occurs during the execution of a program. Normally if a program error occurs the error message will be displayed and the program will continue executing. If this trap is set the error message will not be displayed.
PG_BRK	Executes trap if during a printing routine the internal line position counter should exceed the 'printable lines' for the appropriate device. This is generally used so that the programmer can output the correct header lines at the top of the page or screen.
MOUSE_MOV	Executes trap if mouse is moved.
MOUSE_LBD	Executes trap if left mouse button is pressed.
MOUSE_LBU	Executes trap if left mouse button is released.
MOUSE_RBD	Executes trap if right mouse button is pressed.
MOUSE_RBU	Executes trap if right mouse button is released.

Default Key Operations

F1	Help
F3	Clear Record Buffer
F4	Delete Active Record
F5	Find First Record
F6	Find Last Record
F7	Find Previous Record
F8	Find Next Record
F9	Find Record by Generic Value
F10	Save Record

COMMENTS

Traps are event driven rather than process commands. Nothing actually happens when the command is executed until the appropriate key is pressed or the action occurs.

The ESC and INT trap work in the following manner:

ESC - If set this trap will be executed at any **ENTER** command. If it is not set, or set to DFLT, if the user presses the ESC key at an **ENTER** command the program will terminate, returning to a previous program, or DOS if there was no previous program.. If set to IGNR the bell will sound and the user will remain in the **ENTER**.

INT - This trap is checked between each command and is generally used during reports or other processing where you want to allow the user to exit (or not) and there won't be an **ENTER** command. The only way TAS Pro 5.1 will allow your user to exit during non-ENTER operations is if this trap is set to some value. Both DFLT and IGNR will not allow an exit if the ESC key is pressed.

The RLCK trap works in the following manner:

If option is set to DFLT (default) and you attempt to read a record that is already locked, the program will display a message alerting the user to that fact and will ask if the user wishes to try again or exit from the program entirely. The default option is to try again.

If option is set to IGNR (ignore) and the record is locked a message will be displayed on the screen to that effect but no further input will be required from the user. The message will be automatically cleared and the program will attempt to get the record again.

If the option is set to GOTO or GOSUB the program will transfer control to the appropriate line.

The L_EXIT trap works in the following manner:

If the user tries to access a record that is locked and gets the system message about retrying and answers No (N), this trap will be executed. The only option that applies is the GOTO option. If that option is set, when the user answers **N**, the system will transfer control to the **label** specifiedin the command.

PROGRAM EDITOR prg Control -> Trap -> Trap

SAMPLE PROGRAM TRAPTEST

TRAP OPERATIONS

This command can be used to save, restore or change the entire set of traps.

XTRAP *SAVE/RSTR/CHG* **FLD** *save_to_field NOCHG/DFLT/IGNR*

S*AVE/RSTR/CHG* - Required - *SAVE* - save the trap values to the **save_to_field**. *RSTR* - restore the trap values from the **save_to_field**. *CHG* - change the trap values only.

save_to_field - fn/v - Optional - The total size for all traps is 1000 bytes. If you do not specify a field, the program will allocate one during *SAVE* and deallocate during *RSTR*.

NOCHG/DFLT/IGNR - Optional - What to do with the traps. *NOCHG* - make no changes. *DFLT* - reset traps to their default values. *IGNR* - reset to ignore.

NOTE: If you choose *IGNR* the program will set all traps to *DFLT* (which for most is the equivalent of *IGNR*), and then the following are actually set to *IGNR*: F1 - F10 and ESC. This allows other keys that would be used in the normal course of operations to remain available, such as Up Arrow, Dn Arrow, Pg Up, Pg Dn, Pg_Brk, etc. If you wish to set those to *IGNR* also you must do so with the **TRAP** command.

COMMENTS

This command will allow you to treat the traps as a block. If you only need to save 2 or 3 you should use the **PUSHT** (Push Trap)/**POPT** (Pop Trap) commands and reset them with the standard **TRAP** command.

PROGRAM EDITOR prg Control -> Trap; Xtrap

TRIM FIELD

This will change the leading or trailing spaces in an A type field to binary 0s.

TRIM *fieldname TRAIL/LEAD*

fieldname - fn/v - Required - The name of the field to be trimmed.

TRAIL/LEAD - Optional - Trim the *TRAIL* - trailing, or *LEAD* - leading spaces in the field. Default is *TRAIL*.

COMMENTS

When you use the function **TRIM()** the program puts the 'trimmed' field in the temporary data space. If you are trying to **TRIM** a field that is larger than that space you will get the message that the program is out of room in the temporary data area. This command is provided to you so that you can **TRIM** a field larger than the temporary data area within its own space. Using this command you will not run out of memory.

PROGRAM EDITOR

Field -> alpha Fld cmds -> Trim

SEE ALSO

TRIM()

UNLOCK ALL

Use this command to easily unlock all locked records or files.

ULKALL

No options

COMMENTS

If no records/files are locked, nothing will be done.

PROGRAM EDITOR

fiLe -> Unlock all

UP ARROW

This command allows the programmer to control how far the user can go when s/he presses the Up Arrow key.

UPAR *test_udf* **GOTO** *goto_label*

test_udf - udf - Optional - This is the UDF that would be tested if this command was executed as the result of the user pressing the Up Arrow key. The UDF, if provided, must return a logical value (.T. or .F.). If a UDF is not provided or if the UDF returns a .T. value, the program will not progress above this command and the command immediately following will be executed.

goto_label - label - Optional - If the **test_udf** returns a .F. value and you have provided this **goto_label**, the program will transfer control to that line. If you don't provide a **goto_label**, the program will continue with the line immediately above this command line.

EXAMPLE

When the user presses the Up Arrow key (and no UPAR trap is set) the program progresses backwards from the currently executing line in the program to the next previous **ENTER** command. Using the **UPAR** command you can control this process. For example, suppose you want to use a **FOR/NEXT** loop to enter a group of fields with a single command. You might write the following code:

```
START:
        ENTER CUSTOMER_CODE
        FOR(cntr;1;10;1)
                UPAR PREV_ELEMENT()  GOTO START
                ENTER SALES[CNTR]
        NEXT
        .... (rest of the program)
        QUIT

FUNC PREV_ELEMENT
        IF CNTR = 1 THEN RET .F.
        DEC CNTR
        RET .T.
```

The program need only be concerned when the user presses the up arrow key. When the DN ARROW or ENTER key is pressed the **FOR/NEXT** loop will automatically increment the CNTR field. When the UP ARROW key is pressed the UDF *PREV_ELEMENT()* will be executed. If CNTR>1 it will be decremented, the function will return .T. and the **ENTER** command within the **FOR/NEXT** loop will be executed again, this time with the previous element number. Each time the user presses the UP ARROW key the function will decrement CNTR one more. When CNTR=1 the function will return .F. and the program will transfer control to the line label START.

This program code assumes that all the fields being entered are part of a screen format that has been mounted previously.

If you use the **UPAR** command alone without any other options the program will not allow the user to progress any further 'up' the program and will effectively put a cap on the Up Arrow key.

WINDOWS NOTE: When running under Windows the **UPAR** (without any options) will automatically limit the field search process by setting the start line value to the **UPAR** line instead of checking from the beginning of the program. This along with **CLIK_SRCH_LIMIT** will control where, in your program, the field search process will look for matching **ENTER** commands to the field the user clicks on. Each time the **UPAR** command is executed the upper limit value is changed. You need to be sure that if you have limited the search in a subroutine of your program that you reset that upper limit when you return to the main part. The main thing to remember is that the **UPAR** command with no other options and the **CLIK_SRCH_LIMIT** will define the group of lines the field search process will check. If there is no **CLIK_SRCH_LIMIT** command then the process will continue to the end of the program.

PROGRAM EDITOR

prg Control -> Up Arrow

UPCASE FIELD

This will change all or some subset of all the characters in an A type field to upper case.

UP *fieldname ALL*

fieldname - fn/v - Required - The name of the field to be changed to upper case.

ALL - Optional - If this option is included in the command line then all of the characters in the field will be changed to upper case. If it is not specified then only those characters not surrounded by quote marks (single or double) will be changed to upper case.

COMMENTS

When you use the function **UP()** the program puts the 'upcased' field in the temporary data space. If you are trying to **UP** a field that is larger than that space you will get the message that the program is out of room in the temporary data area. This command is provided to you so that you can **UP** a field larger than the temporary data area within its own space. Using this command you will not run out of memory.

PROGRAM EDITOR

Field -> alpha Fld cmds -> Upcase

SEE ALSO

UP()

UPDATE ARRAY

This command is used to change all or several of the values in an array or group of arrays. It will quickly and efficiently add or subtract values, or insert or delete a specific element number in an array.

UPDTA *array_field_list INS/DEL/ADD/SUB/CLR* **TIMES** *times* **VAL** *value*

array_field_list - fn/v1, fn/v2,..., fn/vx - Required - The names of the array fields to be changed. You may set the array element spec to the first element number to be updated, or in the case of *INS/DEL*, the location to start the insertion/deletion.

INS/DEL/ADD/SUB/CLR - Required - Options include:

ADD - Add a value to an array of values. The type of the value being added needs to be the same as the type of the array field(s). The **value** specified will added the number of **times** specified, starting with the array element indicated in **array_field_list**. For example, if the following were true:

Element #	FLD1
1	250
2	300
3	100
4	125
5	190

and the following were executed:

UPDTA *FLD1[2] ADD* **VAL** *10* **TIMES** *3*

the following would result:

Element #	FLD1
1	250
2	310
3	110
4	135
5	190

SUB - Subtract a value from an array of values. The type of the value being subtracted needs to be the same as the type of the array field(s). For example, if the following were true:

Element #	FLD1
1	250
2	300
3	100
4	125
5	190

and the following were executed:

UPDTA *FLD1[2] SUB* **VAL** *10* **TIMES** *3*

the following would result:

Element #	FLD1
1	250
2	290
3	90
4	115
5	190

INS - Insert a number of 'lines' with **value** being the number. In this case the program will move out the elements from the starting element number so that there are effectively **value** number of elements available for new values. For example, assume the following:

Element #	FLD1	FLD2	FLD3
1	ABC	250	9/23/90
2	NCC	300	12/10/90
3	QHI	100	10/11/90
4		0	0/ 0/ 0
5		0	0/ 0/ 0

If the following is executed:

UPDTA *fld1[2],fld2[2],fld3[2] INS* **VAL** *2*

the result would be:

Element #	FLD1	FLD2	FLD3
1	ABC	250	9/23/90
2		0	0/ 0/ 0
3		0	0/ 0/ 0
4	NCC	300	12/10/90
5	QHI	100	10/11/90

DEL - Delete a number of 'lines' with **value** being the number. In this case the program will move in the elements from the starting element number so that **value** number of element values will be overwritten. For example, assume the following:

Element #	FLD1	FLD2	FLD3
1	ABC	250	9/23/90
2	NCC	300	12/10/90
3	QHI	100	10/11/90
4		0	0/ 0/ 0
5		0	0/ 0/ 0

If the following is executed:

UPDTA *fld1[2],fld2[2],fld3[2] DEL* **VAL** *1*

the result would be:

Element #	FLD1	FLD2	FLD3
1	ABC	250	9/23/90
2	QHI	100	10/11/90
3		0	0/ 0/ 0
4		0	0/ 0/ 0
5		0	0/ 0/ 0

CLR - Clear the array element values, with the number of elements affected determined by **times**. If numerical, date, time, etc. set to 0; if alpha set to spaces. For example, assume the following is true:

Element #	FLD1	FLD2	FLD3
1	ABC	250	9/23/90
2	NCC	300	12/10/90
3	QHI	100	10/11/90
4	RST	75	8/22/90
5	UVW	350	6/15/89

If the following is executed:

UPDTA *FLD1[2],FLD2[2],FLD3[2] CLR* **TIMES** *3*

the following will result:

Element #	FLD1	FLD2	FLD3
1	ABC	250	9/23/90
2		0	0/ 0/ 0
3		0	0/ 0/ 0
4		0	0/ 0/ 0
5	UVW	350	6/15/89

times - f/c/e - Optional - In the case of *ADD/SUB/CLR* this is the number of elements to update. If this isn't included the program will automatically use the number of elements in the array field. This is ignored in *INS/DEL* since the program always uses the number of array elements specified by **value**.

value - f/c/e - Required - This option is required in all but *CLR*. It is the value to be added or subtracted or the number of elements ('lines') to insert or delete.

COMMENTS

Note that in the *INS* and *DEL* mode that the **times** value doesn't need to be set. The program will automatically continue moving in or out element values until it reaches the end of the array for that particular field.

PROGRAM EDITOR

Field -> Array -> Update

SAMPLE PROGRAM

UPDARY, UPDARY2

SEE ALSO

RDA, WRTA, SORTA

USER DEFINED COMMAND

When you want to use a UDC defined elsewhere in your program (using the **CMD** command) or in a separate library all you need to do is put the command name on the line followed by the values to be passed to the routine.

cmd_name cmd_options

cmd_name - sac - Required - The name of the UDC.

cmd_options - f/c/e1,f/c/e2...f/c/ex - Optional - The values to be passed to the UDC. There should be at least one space between the **cmd_name** and the **cmd_options**. Separate the options with commas. Do not surround them with parentheses as you would a UDF or function.

COMMENTS

This is how you will actually execute the UDC. You set up the UDC using the **CMD** command. See the details under **COMMAND**.

PROGRAM EDITOR

System -> Programming -> do udC

SAMPLE PROGRAM UDCTEST

USER DEFINED FUNCTION

Please see the **FUNCTION** command.

WHILE

This is a process control command, i.e., it will control whether a command, or group of commands will be executed. This is one part of a complete command structure.

WHILE *expression*

expression - lexpr - Required - If the **expression** resolves to .T., the command lines between the **WHILE** command and the next **LOOP_IF**, **EXIT**, **EXIT_IF** or **ENDW** (Endwhile) command will be executed.

COMMENTS

For more information on the different structured programming commands, and the **WHILE** command in particular, please see **Chapter 7, Structured Programming Commands.**

PROGRAM EDITOR prg Control -> While -> While

SEE ALSO EXIT, EXIT_IF, LOOP, LOOP_IF, ENDW

WINDOW ACTIVATE

This command will activate a window defined previously with the **WINDEF** (Window Define) command.

WINACT *fieldname*

fieldname - fn/v - Required - The name of the field specified as the **save_field** in the **WINDEF** (Window Define) command.

COMMENTS

Once the window is activated, it is just as though you had executed the standard **WINDOW** command. When you activate the window the program restores the contents to what they were when it was last active. When another window is activated, or you run the standard **WINDOW** command, the program will automatically save the information currently displayed within the window. This allows you to maintain independent windows, none of which will change the others when they are activated. Please note that if you use the standard **WINDOW** command to open a window and then later redisplay a screen, the current contents of the window are **not** saved.

There's an alternative to the WINDEF/WINACT commands which requires a bit of explanation. Whenever a record is found or an expression is evaluated, the program looks at all of the fields that are currently displayed on the screen (by a previous MOUNT command) and redisplays those that are

appropriate. This is how records can automatically appear when you press one of the record search keys (F5-F9). To do this TAS Professional 5.1 looks at a field in memory that gives the segment pointer (offset is always 0 for this buffer) for the field buffer. If a screen has been mounted then this value is non-zero. Each screen format has its own internal buffer location and determining which one is active depends on the value in this location field.

You can set that screen active yourself without having to use the REDSP or WINACT command. This is a two step process. The first step is to get the location of the mount_buffer location within TAS Professional 5.1. This is done through the use of an option in the FFLD() function. The receiving field must be of type P (a P-pointer). In the example below, MOUNT_BUFF is a P-type pointer used to get the location of the screen buffer pointer via the FFLD() function:

```
MOUNT_BUFF=FFLD('~M','P')
```

That's the tilde (~) immediately followed by an 'M', and the quote marks are required. The P surrounded by quotes after the comma tells the function to return a type P pointer. This then gives you the location of the screen buffer pointer itself. After a screen is mounted you would then get the buffer location using the command:

```
MOUNT_LOC=&MOUNT_BUFF
```

Obviously, you would need to do this for each screen after it is mounted the first time, with each buffer location being saved into a different field. If you want to do this for multiple screens, you could make MOUNT_LOC an array and then save each location as a different element. For example:

```
MOUNT_LOC[1]=&MOUNT_BUFF
```

Then, to reset the current buffer location value, you'd do just the opposite:

```
&MOUNT_BUFF=MOUNT_LOC[1]
```

NOTE: The receiving field in both cases must be of type I.

So, why do this when **WINDEF**/**WINACT** does the same thing? Because it's **not** the same thing. For example, suppose you have a screen with data at the top and bottom and a window opened in the middle, such as BKSOA in Advanced Accounting 4.0. In that case we have defined 3 windows (using **WINDEF**) and still have to manually update the data at the bottom of the screen (the current sales order amounts) by using the **PMSG** command each time a line is entered or changed. If instead we had used the process described above, we could **MOUNT** the entire background screen. Then, after a line item had been entered, we'd reset the mount buffer location, which would reactivate the appropriate fields and automatically update the totals at the bottom of the screen. Instead of several lines of code there would be only one!

The other important difference is that this process doesn't change the fixed (non-field) portion of the screen. So it isn't like doing a **SAVES**/**REDSP** each time, which would accomplish the same task, but would wipe out the information in the middle of the screen each time you did a **REDSP**. If you have struggled trying to set up several different **WINDEF** commands to manage and keep separate various sections on the screen, try this instead. It doesn't take any more data space; all the information you're accessing is already there. All you're doing is accessing it differently.

Windows Note: The process described above does NOT work in Windows. However, a similar process does. Each window, when it is active, has a particular pointer or handle value. You can get that value by using the WINDOW_PTR() function. You would save the value in a type R field (as opposed to a type P above). Then, when you want to activate the window you would still use the WINACT command, e.g.:

```
Window_hldr = Window_Ptr()
... other commands ..
Winact Window_hldr      ;this will set the window that was active when you executed the
                        ;Window_Ptr() function above active again.
```

For more information on how windows work run running in Windows please refer to **Chapter 11 - Windows Programming**.

PROGRAM EDITOR User interface -> Windows -> Activate window

SAMPLE PROGRAM MULTWIND

SEE ALSO WINDEF

WINDOW DEFINE

This command is used for defining a window to be opened later with the **WINACT** (Window Activate) command.

WINDEF AT *starting_column,starting_row* **LEN** *length* **WDT** *width*
WCOLOR *window_color* **TTLW** *title_where* **TTL** *title*
BOX *box* **BCOLOR** *box_color* **SHD** *shadow_where*
SCOLOR *shadow_color* **ICOLOR** *inactive_color* **SAVEF** *save_field*
WTXT_COLOR *window_text_color* **BTXT_COLOR** *border_text_color* *USE_COLORS*

starting_column - f/c/e - Required - Upper left corner of the window, column value.

starting_row - f/c/e - Required - Upper left corner of the window, row value.

length - f/c/e - Required - Number of rows long, including box around the window. If there are two lines within the window and there is a **box**, the **length** is 4.

width - f/c/e - Required - Number of columns wide, including box around the window. If the inside width required is 10 columns and there is a **box** then the total **width** is 12.

window_color - f/c/e - Optional - The color value for the main body of the window. If nothing is specified the program will use the current normal color.

title_where - f/c/e - Optional - If you wish to put a title message at the top of the window you may choose where it will be. **L** is left side of window, **R** is right, **C** is centered and **N** is none. The default is **N**.

title - f/c/e - Optional - If you specified a **title_where** value other than **N** you use this as the value to be displayed.

box - f/c/e - Optional - The box type to use for defining the edge of the window. If this is specified you need to take into consideration the 2 extra rows and columns that it requires when determining **length**, **width**, and location. **S** specifies a single line box, **D** is double, and **C** is custom (dependent on what you have set in TAS50.OVL).

box_color - f/c/e - Optional - If you have specified some sort of **box** you may also specify the color for it. If this isn't set the program will use the **window_color** as this value also.

shadow_where - f/c/e - Optional - You may specify that a 'shadow' be displayed behind the window. The **shadow_where** options are **R** - right, **L** - left, or **N** - none. The shadow is displayed two characters to the right or left, and one line down from the window. If you specify the appropriate **shadow_color** value, the results will give the menu a 3-D feel. The default value is **N**.

shadow_color - f/c/e - Optional - If you have specified **L** or **R** for the **shadow_where** option you may also specify the color to be used.

inactive_color - f/c/e - Optional - You can specify a color to be 'painted' when the window is deactivated through the activation of a different window or the **WINDOW** command.

save_field - fn/v - Required - The field that will hold the specifications for the window and actually save the current window display when it is deactivated. To determine the size the field needs to be, multiply the width in columns by the length in rows and double the result. (You double the result to allow one area for display characters, and one for color). Then add 300 to this result, and you'll have computed the storage needed for the screen and window support data. For example, if your window is 10 columns wide by 10 rows long the total field size is: 10 * 10 * 2 + 300 or 500. If the window was the total screen size it would be 25 * 80 * 2 + 300 or 4300.

window_text_color - ***Windows Only*** - f/c/e - Optional - The text color for the text within the window. If this value is not specified the program will use black. For more information on colors in Windows programs please refer to **Chapter 11 - Windows Programming**.

border_text_color - ***Windows Only*** - f/c/e - Optional - This is the text color for the window border and title. If this value is not specified the program will use black. For more information on colors in Windows programs please refer to **Chapter 11 - Windows Programming**.

PROGRAM EDITOR User interface -> Windows -> Windef

SAMPLE PROGRAM MULTWIND

SEE ALSO WINACT

WINDOW OPEN

This command is used for 'opening' a window on the screen. From the program's perspective, once this command is executed the screen becomes the window. Thus, the screen is the same size as the window, and the upper left hand corner of the window is column 1, row 1 for commands that display data to the screen. This will apply until a previously saved screen is redisplayed, or the **SCRN** *R* (Screen Reset) command is run.

WINDOW AT *starting_column,starting_row* **LEN** *length* **WDT** *width*
WCOLOR *window_color* **TTLW** *title_where* **TTL** *title* **BOX** *box*
BCOLOR *box_color* **SHD** *shadow_where* **SCOLOR** *shadow_color*
WTXT_COLOR *window_text_color* **BTXT_COLOR** *border_text_color USE_COLORS*

starting_column - f/c/e - Required - Upper left corner of the window, column value.

starting_row - f/c/e - Required - Upper left corner of the window, row value.

length - f/c/e - Required - Number of rows long, including the box around the window. If there are two lines within the window and there is a **box**, the **length** is 4.

width - f/c/e - Required - Number of columns wide, including box around the window. If the inside width required is 10 columns and there is a **box** then the total **width** is 12.

window_color - f/c/e - Optional - The color value for the main body of the window. If nothing is specified the program will use the current normal color.

title_where - f/c/e - Optional - If you wish to put a title message at the top of the window you may choose where it will be. **L** is left side of window, **R** is right, **C** is centered and **N** is none. The default is **N**.

title - f/c/e - Optional - If you specified a **title_where** value other than **N** you use this as the value to be displayed.

box - f/c/e - Optional - The box type to use for defining the edge of the window. If this is specified you need to take into consideration the 2 extra rows and columns that it requires when determining **length**, **width**, and location. **S** specifies a single line box, **D** is double, and **C** is custom (dependent on what you have set in TAS50.OVL).

box_color - f/c/e - Optional - If you have specified some sort of **box** you may also specify the color for it. If this isn't set the program will use the **window_color** as this value also.

shadow_where - f/c/e - Optional - You may specify that a 'shadow' be displayed behind the window. The **shadow_where** options are **R** - right, **L** - left or **N** - none. The shadow is displayed two characters to the right or left, and one line down from the window. If you specify the appropriate **shadow_color** value the results will give the menu a 3-D feel. The default value is **N**.

shadow_color - f/c/e - Optional - If you have specified **L** or **R** for the **shadow_where** option you may also specify the color to be used.

window_text_color - ***Windows Only*** - f/c/e - Optional - The text color for the text within the window. If this value is not specified the program will use black. For more information on colors in Windows programs please refer to **Chapter 11 - Windows Programming**.

border_text_color - ***Windows Only*** - f/c/e - Optional - This is the text color for the window border and title. If this value is not specified the program will use black. For more information on colors in Windows programs please refer to **Chapter 11 - Windows Programming**.

COMMENTS

The upper left hand corner of the window becomes column 1, row 1 for any commands that display data to the screen. If you have specified a box around the window the space within the window is reduced 2 columns and 2 rows. Therefore, if you need 10 columns and 5 rows of space within a window and you want to surround that window with a box, the total length must be 7 and the width 12. If you do not need the box, the length can be 5 and the width 10.

PROGRAM EDITOR

User interface -> Windows -> Window

SEE ALSO

SAVES, REDSP

WINDOWS COLOR SET (*Windows Only*)

This command will set the appropriate Windows color.

WIN_COLOR *color_name* **VAL** *color_value*

color_name - f/c/e - Required - The name of the color type. The acceptable names are: NormalBkg, NormalText, BoxBkg, BoxText, EnterBkg, EnterText, MsgBkg, MsgText, WindowBkg, WindowTextSet, MenuBkg, MenuTextSet, ButtonBkg, ButtonText, ChoiceBkg, ChoiceText, EcolorBkg and EcolorText.

color_value - f/c/e - Required - The new color value. Must by a type R field.

COMMENTS

This command will reset the internal value for a particular default color. For more information on colors in Windows please refer to **Chapter 11 - Windows Programming**.

PROGRAM EDITOR

Win Commands -> Win Color

SEE ALSO

GET_WIN_COLOR()

WRAP FIELD (*DOS only*)

This command will 'word wrap' an A type field.

WRAP *fieldname* **COL** *number_of_columns* **DLNES** *number_of_lines*

fieldname - fn/v - Required - The name of the field to be wrapped.

number_of_columns - f/c/e - Required - The maximum width of a single line.

number_of_lines - f/c/e - Required - The maximum number of displayable lines. This should be overstated so that you will never reach the max. However, the more lines you have here the slower the rewrap process.

COMMENTS

Once you have 'wrapped' a field using this command you can access individual lines with the **WRAPL()**, **WRAPO()** and **WRAPS()** functions. If you are inserting characters you need to **REWRAP** the line.

PROGRAM EDITOR

User interface -> wRap

WRAP FIELD 3.0

This is a TAS Professional 3.0 command here for compatibility. There is no direct replacement for this command.

WRAP3 *start_col_position*

start_col_position - f/c/e - Required - The column where the next line of text begins. If this number is zero, wrap printing is turned off.

COMMENTS

This is the equivalent of the TAS Professional 3.0 command Wrap Print.

NOTE: The value that is displayed is NOT word wrapped. When the characters won't fit on the first line they just continue with the next. It is recommended that you use a combination of the **WRAP** command and/or TASEDIT.RUN to get the same effect except with more flexibility.

PROGRAM EDITOR 3.0 Commands -> R - Wrap3

WRITE

This command will cause the program to write a specific number of characters to a non-TAS file at a certain location.

WRITE *file_number* **START** *start* **NCHR** *number_of_characters* **FROM** *from* **MEM_AREA** *mem_area#* **OFST** *offset_within_mem_area*

file_number - f/c/e - Required - The file number value. This is the same value as that received in the **OPENV** (Open Variable) command. The file must have been previously opened.

start - f/c/e - Optional - The first byte position to write to. The first character position in the file, for this command, is 1. If no **start** value is given or the value is 0 the program will use the current internal file position value. (When the file is opened the position value is automatically set to 1.) This must be an R type field.

number_of_characters - f/c/e - Optional - The number of characters to write. If this is not supplied the program will use the **size** value stated in the **OPENV** (Open Variable) command.

from - fn/v - Optional - Normally the characters read would be written from the **buffer** specified in the **OPENV** (Open Variable) command. However, you may use this option to specify a different buffer field to be used during the execution of this command.

mem_area# - f/c/e - Optional - If you want to write the characters from a memory area you can specify the number here. If you use this it must resolve to a value of 1 through 4. NOTE: If you don't specify an **OFST** value the characters will start at position 1.

offset_within_mem_area - f/c/e - Optional - If you specify a memory area (**MEM_AREA**) you can also set an offset within that area. The first character in a memory area is at position 1.

COMMENTS

After execution, the internal file position value will be set to the first character immediately following the characters written. For example, if the **start** value is set at 10 and the number of characters to be written is 10, the position after the **WRITE** command will be 20.

PROGRAM EDITOR fiLe -> Write

SEE ALSO READ

WRITE ARRAY

This command is used for writing to a group of records from array fields in memory. Within this single command you can accomplish many tasks that would normally take many more lines of code and more time to execute.

WRTA *from_field_list* **TO** *to_field_list* **RECA** *record_number_array* **MAXA** *number_of_elements* **CNTR** *counter_field* **FILE** *filename/@file_number* **FOR** *for_filter_expression* *DISP*

from_field_list - f/c/e1, f/c/e2,..., f/c/ex - Required - The fields in the array whose values are to be written. You may also create expressions that can be transferred to the receiving (**to_field_list**) fields. The first **from_field_list** field is paired to the first **to_field_list** field, the second to the second, etc. There must be the same number of **from_field_list** fields as **to_field_list** fields.

to_field_list - fn/v1, fn/v1,..., fn/vx - Required - These are the recipient fields in the record to be written. They must be of the same type as the respective **from_field_list** fields. The first **from_field_list** field is paired with the first **to_field_list** field, the second to the second, etc. There must be the same number of **to_field_list** fields as **from_field_list** fields.

record_number_array - fn/v - Optional - If you want to update current records then you must provide an array of record numbers so this command can read the appropriate record before updating the fields and saving it back. The record number can be obtained through the use of the **RCN()** function.

NOTE: If you read these records using the **RDA** (Array Read) command you may make one of the **from_field_list** fields the function **RCN()** and the corresponding **to_field_list** field an R type array field to hold the record number.

number_of_elements - f/c/e - Required - The number of array elements (or records) to be written.

counter_field - fn/v - Optional - This value is updated by the program during operation so that you may use expressions in the **from_field_list**. Use this field as the array specifier where appropriate. Must be of type I.

filename/@file_number - file_expr - Optional - The name or number of the file to be used. If you do not include this option, the program will look for a default search file set in the **SRCH** (Search File) command. If a default file hasn't been previously set, the program will report an error and the command will be skipped. This entry will override any default value set in the **SRCH** command.

for_filter_expression - lexpr - Optional - You would use this option to restrict the records written in this command. If the expression did not resolve to .T., the record would not be written and the program would continue to the next array element. The process continues until all array elements (**number_of_elements**) have been checked and written if appropriate.

DISP - Optional - If you include this option, each time a record is ready to be written the program will redisplay the screen fields as appropriate. This will slow down the operation of this command, with the amount of slow-down depending on how many fields are displayed on the screen. The net effect will probably be negligible unless you are reading through a very large file.

PROGRAM EDITOR fiLe -> Mult Rec Cmds -> Write Array

SAMPLE PROGRAM WRTARRAY

SEE ALSO RDA

WRITE RECORD

This is a TAS Professional 3.0 command here for compatibility. The preferred method is to use the **OPENV/SAVE** commands.

WTREC*err_label*

err_label - label - Optional - This is the label to go to if an error occurs while writing to this non-TAS file.

COMMENTS

This is the equivalent of the TAS Professional 3.0 command Write non-TAS Record.

PROGRAM EDITOR 3.0 Commands -> S - Wt Record

XFER

Please see the **MOVE DATA IN MEMORY** command.

XTRAP

Please see the **TRAP OPERATIONS** command.

TAS
PROFESSIONAL
VERSION 5.1

Chapter 6

Function Reference

INTENTIONALLY BLANK

CONVENTIONS USED IN THE FUNCTION REFERENCE SECTION

The following format is used in documenting the TAS Professional 5.1 Functions.

- The functions are listed in alphabetical order.
- The **PURPOSE** of each function is a brief definition of what it does. This provides a quick insight into the function, and why the programmer might use it.
- The **PARTS** of each function are listed by number.

  Each function part is described, along with its purpose, valid values, default value (if applicable), consequences, and whether or not the part is required.

  The programmer entry portion can consist of the following:

  f/c/e - Field/Constant/Expression. You can use any type of field or constant. This can consist of alpha or numeric constants as well as any type of field or variable field. You can also use an expression that results in the proper type of field.

  fn/v - Field name/Variable field. In certain commands/options you must use the actual field name or a variable field.

  key_expr - You may define the key number to be used in a command in several ways:

  number - The actual number of the key, as defined in the data dictionary, may be used preceded by an ampersand ('@'). *@3* refers to the third key in the appropriate file (the first key is 1).

  key name - The name of the key as assigned in the data dictionary may be used. *dict_overlay* is an example of a key name.

  no key - If you do not want to use a key but want to access the file through the direct method then use the no key symbol, the number 0 preceded by an ampersand ('@'). *@0* tells the program to use the direct access method when searching this file. This is the default method when you have opened a non-TAS file.

  lexpr - An expression that will resolve to a logical value of .T. or .F. For example:

  fld1 = 100

  another example:

  (fld1 = 100 .a. fld2='ABC') .o. fld3=.F.

- The **RETURN TYPE** and what is being returned is given.
- Special **COMMENTS** or restrictions are given where applicable.
- **SAMPLE PROGRAM** will list the name of a file containing a program that demonstrates the function in action.
- **EXAMPLES** will give the function followed immediately below by the result.

- Functions may be nested up to 10 deep:

  F1(F2(F3(F4(F5(F6(F7(F8(F9(F10()))))))))))

  Fx in this example represents a TAS Professional 5.1 function name. A Function may be used in any expression.

- (DOS Only) - If the function does not execute in the Windows environment then this line will be next to the function name at the top of the information block. In this situation the function will be ignored when running in Windows but will not cause an error. It will return a blank, 0 or .F. as appropriate.

- (Windows Only) - If the function does not execute in the DOS environment then this line will be next to the function name at the top of the information block. In this situation the function will be ignored when running in DOS but will not cause an error. It will return a blank, 0 or .F. as appropriate.

ABS(1)

PURPOSE

This function returns the absolute value of the N type field specified.

PARTS

1 f/c/e The N type value.

RETURN TYPE

N

EXAMPLE

```
? abs(-535.25)
535.25
```

ACOS(1)

PURPOSE

This function returns the arccosine of the N type field specified.

PARTS

1 f/c/e The N value.

RETURN TYPE

N The result will be from 0 to PI in radians.

COMMENTS

The value specified must be between -1 and 1.

EXAMPLE

```
? acos(-0.539)
2.14004577
```

ACS()

PURPOSE

This function will return the name of the current alternate collating sequence file, if any.

NO OTHER PARTS

RETURN TYPE

A If there is no ACS file active the function will return NONE.

AEV(1,2)

PURPOSE

This function will return an element value from an array field.

PARTS

1 fn/v The array field itself. You may use a P type field to point to the array field.

2 f/c/e The element number.

RETURN TYPE

The function will return the same type of field as the array field (option 1).

ALC_FLD(1,2)

PURPOSE

You can use this function to allocate a single previously defined field.

PARTS

1 fn/v The field to allocate. This must be a type A field.

2 f/c/e The new desired size. Maximum size possible is 65,535. If the amount of available memory is less than the size given the program will be allocated as much as possible.

RETURN TYPE

I The actual size allocated.

COMMENTS

The field (part 1) used in this command cannot be a part of a standard TAS Professional 5.1 file. It must have been created using the **DEFINE** command.

Once this function has been used to allocate the field the data space originally used by the field is lost to use. Due to this you should define the field with a size of 1 initially. This keeps the amount of wasted space to a minimum.

ALOC(1,2,3)

PURPOSE

You can use this function to find an array element number based on a value you specify.

PARTS

1 f/c/e The field value to search for.

2 fn/v The array field to search.

3 f/c/e The array element value to start with. If none is provided the program will start with the first.

RETURN TYPE

I The element number if found.

COMMENTS

If there is a match for the search field value the program will return the array element number. If no match is found the return value will be 0.

NOTE: Do not specify an array element in the array field to search for (part 2). If you wish to start with an element other than 1 specify that value in part 3.

ALOCARY(1,2)

PURPOSE

You can use this function to allocate an array for a previously defined field.

PARTS

1 fn/v The field to allocate array elements for.

2 f/c/e The number of elements to allocate. If there isn't enough available memory for the number of elements desired the program will allocate as many as possible. The maximum number of array elements for any single field is 65,535.

RETURN TYPE

I The actual number of elements allocated.

COMMENTS

The field (part 1) used in this command cannot be a part of a standard TAS Professional 5.1 file. It must have been created using the **DEFINE** command.

Once this function has been used to allocate the array the data space originally used by the field is lost to use. Due to this you should define the field with no elements initially. This keeps the amount of wasted space to a minimum.

ASC(1)

PURPOSE

This function will return the ASCII value of the first character of a field.

PARTS

1 f/c/e The field to be used. The function will use only the first character of the field which must be of A type.

RETURN TYPE

I The ASCII value.

ASIN(1)

PURPOSE

This function returns the arcsine of the N type field specified.

PARTS

1 f/c/e The N value.

RETURN TYPE

N The result will be from -PI/2 to PI/2 in radians.

COMMENTS

The value specified must be between -1 and 1.

EXAMPLE

```
? asin(-0.539)
-0.56924944
```

ASK()

PURPOSE

This function returns the logical value of what the user entered for the last **ASK** command entry.

NO OTHER PARTS

RETURN TYPE

L If the user answered the last **ASK** command question with a **Y** (yes) this function will return .T.; otherwise it will return .F.

ATAN(1)

PURPOSE

This function returns the arctangent of the N type field specified.

PARTS

1 f/c/e The N value.

RETURN TYPE

N The result will be from -PI/2 to PI/2 in radians.

EXAMPLE

```
? atan(-0.539)
-0.49435871
```

ATAN2(1,2) (*DOS only*)

PURPOSE

This function returns the arctangent of field value1/field value2.

PARTS

1 f/c/e Field 1, must be N type.

2 f/c/e Field 2, must be N type.

RETURN TYPE

N The result will be from -PI to PI in radians.

EXAMPLE

```
? atan2(-0.539,-1.325)
-2.75524446
```

AVAIL(1)

PURPOSE

This function returns the approximate number of elements that could be allocated given a field or group of fields.

PARTS

1 f/c/e The total internal size for the field or fields to be used in the function.

RETURN TYPE

I The maximum number of elements that could be allocated for the total size given.

BOF(1)

PURPOSE

This function will determine whether or not the program has tried to read a record that is previous to the beginning of the file.

PARTS

1 f/c/e The **file_number** value of the file to be checked.

RETURN TYPE

L If the file is at BOF the function will return .T.; otherwise it will return .F.

CBYT(1)

PURPOSE

This function will return the given field as a B type (1 byte integer) field.

PARTS

1 f/c/e The field to be converted to B type.

RETURN TYPE

B The original field is not changed.

COMMENTS

F, P, T, and D type fields cannot be converted to B; however, all others can. If an L type field is converted, a .T. value will be converted to 1, a .F. to 0. If the value of the field to be converted is larger than the maximum value of a B type field (255), the converted value will not be accurate.

CC(1) (*DOS only*)

PURPOSE

This function changes the color temporarily to the value specified. May be used in a PRINT MESSAGE command only.

PARTS

1 f/c/e The new color value.

RETURN TYPE

There is no value returned in this function. The only result is that the color value is temporarily changed.

CCE() (*DOS only*)

PURPOSE

This function changes the color temporarily to **error**. May be used in a PRINT MESSAGE command only.

NO OTHER PARTS

RETURN TYPE

There is no value returned in this function. The only result is that the color value is temporarily changed.

CCF() (*DOS only*)

PURPOSE

This function changes the color temporarily to **foreground**. May be used in a PRINT MESSAGE command only.

NO OTHER PARTS

RETURN TYPE

There is no value returned in this function. The only result is that the color value is temporarily changed.

CCH() (*DOS only*)

PURPOSE

This function changes the color temporarily to **highlight**. May be used in a PRINT MESSAGE command only.

NO OTHER PARTS

RETURN TYPE

There is no value returned in this function. The only result is that the color value is temporarily changed.

CCR() (*DOS only*)

PURPOSE

This function changes the color temporarily to **reverse**. May be used in a PRINT MESSAGE command only.

NO OTHER PARTS

RETURN TYPE

There is no value returned in this function. The only result is that the color value is temporarily changed.

CDOW(1)

PURPOSE

The function returns the character day of week value (e.g., MONDAY, TUESDAY, etc.) for a particular date.

PARTS

1 f/c/e The date value to use, must be of D type.

RETURN TYPE

A The function will return the name of the day. If the receiving field is too small to hold the entire name only the portion that will fit will be returned.

SAMPLE PROGRAM

DATETEST.SRC

CEIL(1)

PURPOSE

This function returns a whole number that is the closest value equal to or greater than the field value specified.

PARTS

1 f/c/e Must be N type field.

RETURN TYPE

N

EXAMPLE

```
? ceiling(-2.935)
-2.000                  && Don't forget, -2 is > -2.935
? ceiling(2.935)
3.000
```

CFLT(1)

PURPOSE

This function will return the given field as an N type (floating point) field.

PARTS

1 f/c/e The field to be converted to N type.

RETURN TYPE

N The original field is not changed.

COMMENTS

An F type field cannot be converted to N; however, all others can. If an L type field is converted, a .T. value will be converted to 1.00, a .F. to 0.00.

CHR(1)

PURPOSE

This program will return the ASCII character based on the value given.

PARTS

1 f/c/e The value to use. The legal values range from 0 thru 255.

RETURN TYPE

A The function returns a single character.

CINT(1)

PURPOSE

This function will return the given field as an I type (2 byte integer) field.

PARTS

1 f/c/e The field to be converted to I type.

RETURN TYPE

I The original field is not changed.

COMMENTS

F, P, T, and D type fields cannot be converted to I; however, all others can. If an L type field is converted, a .T. value will be converted to 1, a .F. to 0. If the value of the field to be converted is larger than the maximum value of an I type field (65535), the converted value will not be accurate.

CLICKED_ON() (*Windows only*)

PURPOSE

This function returns whether or not the user has moved to a specific field by clicking on it with the mouse.

NO OTHER PARTS

RETURN TYPE

L If the use has clicked on the field this function will return .T., otherwise, if normal program process brought the user to this field it will return .F.

COMMENTS

Line numbers in a program run consecutively, so the line after the current one is **CLNUM()**+1.

CLNUM()

PURPOSE

This function returns the current program line number.

NO OTHER PARTS

RETURN TYPE

I

COMMENTS

Line numbers in a program run consecutively, so the line after the current one is **CLNUM()**+1.

CMNTH(1)

PURPOSE

This function returns the character month of the year value (e.g., JANUARY, FEBRUARY, etc.) for a particular date.

PARTS

1 f/c/e The date value to use, must be of D type.

RETURN TYPE

A The function will return the name of the month. If the receiving field is too small to hold the entire name only the portion that will fit will be returned.

SAMPLE PROGRAM

DATETEST.SRC

CO()

PURPOSE

This function returns the character company code file extension modifier.

NO OTHER PARTS

RETURN TYPE

A

COMMENTS

The company code is added to the extension of all standard TAS Professional 5.1 files opened once it is set.

COL()

PURPOSE

This function returns the current internal screen column location value.

NO OTHER PARTS

RETURN TYPE

I

COMMENTS

Column 1 is the far left column for the active window.

COPY_FILE(1,2) (*Windows Only*)

PURPOSE

You must use this function to copy a file in Windows. You cannot use standard DOS commands.

PARTS

1 f/c/e From file name. Must be type A.

2 f/c/e To file name. Must be type A.

RETURN TYPE

L If the copy is complete the function returns .T., otherwise it returns .F..

COMMENTS

Wild card characters are not allowed in this function. You must specify both file names explicitly.

COS(1)

PURPOSE

This function returns the cosine of the N type field specified.

PARTS

1 f/c/e The N value.

RETURN TYPE

N The return value is in radians.

COMMENTS

The value specified must be between -1 and 1.

EXAMPLE

```
? cos(1.539)
0.03179097
```

CPATH()

PURPOSE

This function returns the user's current path value.

NO OTHER PARTS

RETURN TYPE

A The receiving field should be at least 47 characters long.

EXAMPLE

```
? cpath()
C:\TASPRO\
```

CREC(1)

PURPOSE

This function will return the given field as an R type (record or 4 byte integer) field.

PARTS

1 f/c/e The field to be converted to R type.

RETURN TYPE

R The original field is not changed.

COMMENTS

An F type field cannot be converted to R; however, all others can. If an L type field is converted, a .T. value will be converted to 1, a .F. to 0.

CTOD(1)

PURPOSE

This function converts an A (character string) type field to a D (date) type field.

PARTS

1 f/c/e The A type field that is to be converted.

RETURN TYPE

D

SAMPLE PROGRAM

DATETEST.SRC

CTOL(1)

PURPOSE

This function converts an A (character string) type field to a L (logical) type field.

PARTS

1 f/c/e The A type field that is to be converted.

RETURN TYPE

L

EXAMPLE

```
? ctol('.Y.')
.T.
```

CTOT(1)

PURPOSE

This function converts an A (character string) type field to a T (time) type field.

PARTS

1 f/c/e The A type field that is to be converted.

RETURN TYPE

T

SAMPLE PROGRAM

TIMETEST.SRC

DATE()

PURPOSE

This function will return the current system date.

NO OTHER PARTS

RETURN TYPE

D

COMMENTS

This is the same date that is returned if the user enters DATE at the DOS prompt.

DELF(1)

PURPOSE

This function will attempt to delete a file. If the deletion is successful, a logical .T. will be returned.

PARTS

1 f/c/e The name of the file to be deleted. This must include the path and extension in standard notation.

RETURN TYPE

L If the file is found and deleted the function will return a value of .T.; otherwise the returned value is .F.

DIFF(1,2)

PURPOSE

Using a process similar to the **SNDX()** function, this function will return a value that can be used to determine how 'close' the two values 'sound.'

PARTS

1 f/c/e Comparison value 1. Must be type A.

2 f/c/e Comparison value 2. Must be type A.

RETURN TYPE

I The return value is from 0 thru 4. 0 is no match at all; 4 is a very close match.

EXAMPLE

```
? DIFF('translate','trnslt')
4
? DIFF('John','Paul')
0
```

DMY(1,2)

PURPOSE

This function returns a date value in the form of Day Month Year.

PARTS

1 f/c/e Date value to use.

2 f/c/e If set to 'S' only the first three characters of the month value are used (e.g., Jan = January) and only the last two digits of the year. If set to 'L' (long) then the full month name and the full year are used.

RETURN TYPE

A

EXAMPLE

```
? dmy(date(),'s')
23 Jun 94

? dmy(date(),'l')
23 June 1994
```

DOM(1)

PURPOSE

This function will return the day of the month as a number.

PARTS

1 f/c/e The date value to use, must be of D type.

RETURN TYPE

I

SAMPLE PROGRAM

DATETEST.SRC

DOW(1)

PURPOSE

This function will return the day of the week as a number.

PARTS

1 f/c/e The date value to use, must be of D type.

RETURN TYPE

I The number range returned is from 1 thru 7, Sunday thru Saturday.

SAMPLE PROGRAM

DATETEST.SRC

DPATH()

PURPOSE

This function will return the default dictionary path value as set in the TAS50.OVL file or by the **CHANGE DICTIONARY PATH** command.

NO OTHER PARTS

RETURN TYPE

A

EXAMPLE

```
? dpath()
C:\ACCTG\
```

DSPCE()

PURPOSE

This function will return the amount of unused disk space on the current 'default' drive.

NO OTHER PARTS

RETURN TYPE

R The value returned is in number of bytes. If there are 5 Mbytes remaining the returned value would be: 5242880

DTOC(1,2)

PURPOSE

This function will convert a D type field into an A type field.

PARTS

1 f/c/e The D type field to be converted. The original field value is not changed.

2 f/c/e The size of the A type version of the date to return. The default is 8 (mm/dd/yy). You can use any legal date size as the value.

RETURN TYPE

A If the receiving field is not long enough the returned value will be truncated.

SAMPLE PROGRAM

DATETEST.SRC

DTOR(1)

PURPOSE

This function will convert a D type field to an R type field.

PARTS

1 f/c/e The D type field to be converted. The original field is not changed.

RETURN TYPE

R The returned value is the number of days since day 0.

SAMPLE PROGRAM

DATETEST.SRC

DTOS(1)

PURPOSE

This function will return a date value as a string in the form YYYYMMDD.

PARTS

1 f/c/e Date value to use.

RETURN TYPE

A An 8 character string is always returned.

EXAMPLE

```
? dtos(date())
19940623
```

EDIT(1) (*Windows Only*)

PURPOSE

This function will edit an existing note or create a new one. It handles word wrap automatically and provides all the normal Windows editing capabilities including marking a block of text, saving that block to the clipboard, and inserting a block of text from the clipboard to the note.

PARTS

1 fn/v Field name that contains the current note or will contain the new one.

RETURN TYPE

L If the user presses the SAVE button (or F10) then .T. will be returned. If the user presses the ESC key or button then .F. will be returned.

COMMENTS

This function replaces the TASEDIT.RUN program in DOS. It expects to have a field that contains the text for the entire note. CR/LFs are ok and are preserved by the function. Before you call this function be sure to create a window where the editing will take place. If you haven't created a window the function will use the entire base

window. At the bottom of the window the function creates two buttons, one is F10 To Save Memo, and the other is ESC Exit without Saving. If the user clicks on the F10 button or presses the F10 key the function will put the new note/memo in the field provided and returns .T. If the user presses the ESC key (or button) the function will not change the field provided and returns .F.

ELOC(1,2,3,4)

PURPOSE

This is an 'instring' function. It will return the location of a string of characters (A type field) within another A type field, starting at the end of the field and working towards the beginning.

PARTS

1 f/c/e Field to check for, must be A type.

2 f/c/e Field to check within, must be A type.

3 f/c/e The character position to start checking at. The first position (far left) is 1. If this isn't provided the default value is the end of the field.

4 f/c/e The number of characters to check. If this isn't provided the function will check the entire field (part 2).

RETURN TYPE

I The first character position of the matched field. If no match is found the value returned is 0.

EXAMPLE

```
x = 'ABCDEFGHCDIJ'
? loc('CD',x)
9

? loc('CD',x,3)
0
```

SAMPLE PROGRAM

LOCTEST.SRC

ENTER()

PURPOSE

This function will return the result of the last executed **post** UDF option in the **ENTER** command.

NO OTHER PARTS

RETURN TYPE

L The value that was returned by the **post** UDF.

EOF(1)

PURPOSE

This function will determine whether or not the program has tried to read a record that is past the end of the file.

PARTS

1 f/c/e The **filenumber** value of the file to be checked.

RETURN TYPE

L If the file is at EOF the function will return .T.; otherwise it will return .F.

ESC()

PURPOSE

If the user presses the ESC key at a message and the appropriate ESC traps are set to **ignore** then the internal esc_pressed flag is set to 1. You can test for that flag being set using this function.

NO OTHER PARTS

RETURN TYPE

L If the user pressed the ESC key this will be .T..

COMMENTS

Once this flag is checked it will be reset to .F.. It is also cleared before the next **MESSAGE** or **ENTER** command.

ETYP()

PURPOSE

If the program is executing a **LIST FILE** or **LIST ARRAY** command, and an **ENTER** *UDF* option is included, and the user presses the ENTER, RETURN, DELETE or INSERT key, this function will return the type of enter.

NO OTHER PARTS

RETURN TYPE

A A single character is returned.

COMMENTS

The possible returns are:

C - The cursor is on an active (existing) line. So this is a Change type.

I - The user has pressed the Insert key.

D - The cursor is on an active (existing) line, and the user has pressed the Delete key.

A - The cursor is on the line after the last active line and the user has pressed the Enter or Insert key. This is an Add enter instead of an Insert since the new entry will be added to the end of the list.

EXEC()

PURPOSE

This function will return whether or not a non-TAS program has executed. If it didn't execute at all this function will return .F.; otherwise, it will return .T.

NO OTHER PARTS

RETURN TYPE

L

EXP(1)

PURPOSE

This function returns the exponential of the field given.

PARTS

1 f/c/e The N type field to be used in the calculation.

RETURN TYPE

N

COMMENTS

The function returns e^x, with x being the field value given.

SAMPLE PROGRAM

LOGTEST.SRC

FARRAY(1)

PURPOSE

This function returns the number of array elements for a specific field.

PARTS

1 f/c/e The name of the field being checked.

RETURN TYPE

I The number of array elements. If the field is not an array or doesn't have any elements the function will return 0.

SAMPLE PROGRAM

FLDTEST.SRC

FCHR(1,2)

PURPOSE

This function will return the first displayable (non-space) character in an A type field or the position value of that character.

PARTS

1 f/c/e The field to be checked. Must be an A type field.

2 f/c/e If 'C' the function will return the character; if 'L' the location of the first character.

RETURN TYPE

A If 'C' the function returns a single A type character.
I If 'L' the function returns an I type value.

EXAMPLE

```
x = '    100.21'
? fchr(x,'c')
1

? fchr('    100.21','l')
5
```

SAMPLE PROGRAMS

ETST.SRC, LSTCTEST.SRC

FFILE(1,2)

PURPOSE

This function searches for and returns the name of a file if found.

PARTS

1 f/c/e The file to search for. You may use the standard DOS file search characters of * and ? in the same manner as they would be used searching at the DOS prompt.

2 f/c/e Whether to search for the first occurrence of the file or the next. 'F' searches for the first, 'N' for the next.

RETURN TYPE

A Returns the file name if found. If the file is not found a blank field ("") will be returned. If the receiving field is too short the returning value will be truncated.

COMMENTS

You can search for an entire group of files that match a certain pattern. The first time you search part 2 must be set to 'F'. For any searches after that, part 2 must be set to 'N'. When no more matches are found the file name returned will be blank (""). Part 1 is not consequential when Part 2 is set to 'N'. However, a null field ("") at least must be included as a space keeper.

You must include the path if the file being searched for isn't in the current path. However, the file name returned will not include any path value, only the name and extension.

EXAMPLE

```
? FFILE('D*.*','F')
DICTDAT.B or DATABASE.DAT etc.

? FFILE('D??DAT.B','F')
DBCDAT.B or D12DAT.B etc.
```

FFLD(1,2)

PURPOSE

This function will return the pointer (P or F) for a field. The function can also return a logical value to indicate that the field was not found.

PARTS

1 f/c/e The name or number of the field to search for.

2 f/c/e What to return. 'P' returns a P type pointer, 'F' returns an F type pointer, and 'L' returns a logical value.

RETURN TYPE

P, **F** or **L**

COMMENTS

If the return type is **P** or **F**, and you use a number and it is out of range, or the field name is not found, the program will return an error. If the return type is **L** the function will return .F.

FILL(1,2)

PURPOSE

This function will return an A type field of a specific size and made up entirely of a specific character.

PARTS

1 f/c/e The size of the field to be returned.

2 f/c/e The fill character to be used. This must be an A type field and only the first character of that field will be used.

EXAMPLE

```
? FILL(10,'*')
**********
```

FLDATE(1)

PURPOSE

This function returns the update date of the file named in the function.

PARTS

1 f/c/e The name of the file to be checked. You must include the path and extension in standard notation.

RETURN TYPE

D

COMMENTS

If the file isn't found the returned date will be 0.

FLDFDNUM(1)

PURPOSE

This function returns the file handle, if any, for the field named in the function.

PARTS

1 f/c/e An F type pointer field that is pointing at the appropriate field.

RETURN TYPE

I

COMMENTS

If the field isn't in a file or you haven't used an F type pointer the returned value will be 0.

FLERR(1)

PURPOSE

This function will return the error number for the last access for a particular file.

PARTS

1 f/c/e The **file_number** of the file to be checked.

RETURN TYPE

I The last file error number.

COMMENTS

If the last access was successful the FLERR() value returned will be 0. This function works for both TAS and non-TAS files. The error number returned will correspond to standard TAS Professional runtime errors. For more information please refer to **Chapter 10, Runtime Errors**.

FLDNME(1)

PURPOSE

This function will return the field name for the F type pointer value given.

PARTS

1 f/c/e The fptr value of the field name to be found.

RETURN TYPE

A The name of the field.

COMMENTS

This will only work with F type pointers. Do NOT preface the pointer with the redirector symbol '&'.

FLOOR(1)

PURPOSE

This function returns a whole number that is the closest value equal to or less than the field value specified.

PARTS

1 f/c/e Must be N type field.

RETURN TYPE

N

EXAMPLE

```
? floor(-2.935)
-3.000              && Don't forget, -3 is < -2.935
? floor(2.935)
2.000
```

FLSZE(1)

PURPOSE

This function returns the size of the file given in bytes.

PARTS

1 f/c/e The name of the file to be checked. You must include the path and extension in standard notation.

RETURN TYPE

R

COMMENTS

If the file isn't found the returned size will be 0.

FLTIME(1)

PURPOSE

This function returns the update time of the file named in the function.

PARTS

1 f/c/e The name of the file to be checked. You must include the path and extension in standard notation.

RETURN TYPE

T

COMMENTS

If the file isn't found the returned time will be 0.

FNUM(1)

PURPOSE

This function returns the **file_number** for a file opened with the **OPENV** (Open Variable) command.

PARTS

1 f/c/e The name of the file. This needs to be in standard alpha notation or as an A type field (e.g., 'BKARCUST').

RETURN TYPE

I Will return the **file_number** if found. If it is not found, the function will return 0.

COMMENTS

This function will allow you to use the old form **OPEN** command and still make use of certain functions that require the **file_number** instead of the file name.

EXAMPLE

This function can be embedded in any other function so that you don't need to keep the **file_number** in another field. For example:

```
? flerr(fnum('BKARCUST'))
0

? eof(fnum('BKARCUST'))
.F.
```

FTOT(1)

PURPOSE

This function will convert an N type field to a T (time) field.

PARTS

1 f/c/e The N type field value to be converted to a T field. The original field is unchanged.

RETURN TYPE

T

FTYP(1)

PURPOSE

This function will return the type of a specific field.

PARTS

1 f/c/e The name of the field being checked.

RETURN TYPE

A A single character representing the field type will be returned. If no field is found the function will return a blank.

SAMPLE PROGRAM

FLDTEST.SRC

GET_ELEM_NUM() (*Windows Only*)

PURPOSE

This function returns the element number for the last or current entry field.

NO OTHER PARTS

RETURN TYPE

R Returns the element number for the last or current entry field.

COMMENTS

This function would be used in programs where you enter a group of array fields within a for/next loop and one **ENTER** command for each field. In Windows the program can determine which element of a field the user has clicked on. In DOS this could not be determined and so when a user tried to click on an array field it was just ignored. This function gives you a way to determine which element has been

chosen, and, if the user presses the ENTER key after entry, to continue with the next element instead of the next counter value in the for/next loop. You should note, that the program will not change the value of the for/next counter automatically. It's up to you to do that and for that we provide this function.

You can also use this function in the PRE udf to make sure the field passes the tests necessary to allow entry.

NOTE: This function works the same way whether the user has clicked on the field with their mouse or is just pressing ENTER to go through a list of fields in an array. However, during the PRE function it changes temporarily to the element counter for the field being chosen so that you can test acurately. See example below

EXAMPLE

The following is an example of the use of this function. The first func is a PRE udf and the second is a POST. Note that we don't have to differentiate between a **CLICKED_ON()** field and an 'normal' field in the POST test since once the field is in entry mode the function returns the proper value.

```
func pre_test
      if CLICKED_ON()
            ;this part tests the field during the PRE testing
            ;in the field search process.
            if FLD[GET_ELEM_NUM()]>0 then ret .t.
      else
            ;this part tests the field during 'normal' entry.
            if FLD[CNTR]>0 then ret .t.
      endif
      ;if the field is not >0 then ret .f.
      ret .f.

func post_test
      CNTR = GET_ELEM_NUM()
      ret .t.
```

In the example above it is assumed that the name of the field being entered is FLD and that the counter in the for/next loop is CNTR. Note that if you're writing for both DOS and Windows that you don't need to use IF WINDOWS() .A. CLICKED_ON() since this function will return .F. automatically in the DOS environment.

GET_WIN_COLOR(1) (*Windows Only*)

PURPOSE

This function returns the internal color value for an alpha specifier..

PARTS

1 f/c/e The color name. See below for available names.

RETURN TYPE

R Returns the color value. Will return 0 if no color has been found, even though this is a legitimate color.

COMMENTS

Colors are siginificantly different in Windows than they are in DOS. To help with this problem we have provided this function. You pass it a color name and it returns the appropriate value. Acceptable color names are: (the following names are standard Windows colors and return values you have set for Windows in general) ScrollBar, Background, ActiveCaption, InactiveCaption, Menu, Window, WindowFrame, MenuText, WindowText, CaptionText, ActiveBorder, InactiveBorder, AppWorkSpace, Highlight, HighlightText, BtnFace, BtnShadow, GrayText, BtnText, InactiveCaptionText, BtnHighlight.

These next names are for regular colors: Black, Maroon, Green, Olive, Navy, Purple, Teal, Gray, Silver, Red, Lime, Yellow, Blue, Fuchsia, Aqua, LtGray, DkGray, White.

The last are for values you set in the TP5WIN.INI file: NormalBkg, NormalText, BoxBkg, BoxText, EnterBkg, EnterText, MsgBkg, MsgText, WindowBkg, WindowTextSet, MenuBkg, MenuTextSet, ButtonBkg, ButtonText, ChoiceBkg, ChoiceText, EcolorBkg, EcolorText.

EXAMPLE

An example would be:

```
WIN_COLOR 'MenuBkg' val GET_WIN_COLOR('Red')
```

This example will set the main menu color to red. You should note that red on one machine may not be the same as red on another.

GETENV(1)

PURPOSE

This function will check for a matching value in the DOS environment area and, if one is found, will return the characters after the =.

PARTS

1 f/c/e The environment variable name to search for. Don’t include the equal sign (=).

RETURN TYPE

A If a match is not found a null string will be returned.

EXAMPLE

```
? getenv(‘tas’)
D:\PROF\
```

GET_REC(1,2)

PURPOSE

This function will return an individual record from a buffer that has been used to store an entire file. Before using this function you should have read a non-TAS type file (general text or X type) into a field in your program that you are using as a temporary buffer.

PARTS

1 fn/v The name of the field you are using as a buffer.

2 f/c/e The record number you want to retrieve. The first record in the buffer is 1.

RETURN TYPE

A The size of this field depends on the number of characters in the record. The termination characters(s) at the end of the record is not returned as part of the record.

COMMENTS

Each record in the buffer must be terminated by a carriage return (CR) or a carriage return/line feed pair (CR/LF) or a binary 0. If you attempt to get more records than are in the file, the function will return a blank field with a length of 0. This also applies if there are no characters in the record to return.

GFL(1)

PURPOSE

This function will take a format line from a mounted report format and convert it to an alpha field with all appropriate values set.

PARTS

1 f/c/e The number of the format line to use.

RETURN TYPE

A If the format line number is not accurate or if a report format has not been mounted, a blank value will be returned. If the receiving field is not large enough to contain the entire line the remaining characters will be truncated.

COMMENTS

This is the same as printing the report format line except the line is being placed in an A type field.

GFLD(1,2,3,4)

PURPOSE

This function returns a field from a delimited file buffer previously read with the **READ** or **FINDV** (Find Variable) command.

PARTS

1 f/c/e The field number in the delimited record. The first value in the record is field 1.

2 f/c/e The **file_number** for the file that contains the delimited records. You must provide a **file_number** or provide a field or constant in part 4.

3 f/c/e The delimiter between fields in the buffer. If this value is not provided the program assumes a comma ‘,’.

4 f/n/v A field to be used instead of a **file_number**.

RETURN TYPE

A

COMMENTS

The quotes surrounding A type fields in the delimited record are automatically removed before the value is returned.

If you specify a field name as part 4, the program will treat that field as a record buffer so that the overall process will remain the same as if you had specified a **file_number**. If you use this option, be sure to put in a place-holding comma where the **file_number** would ordinarily go. In the example below, the programmer has not specified a **file_number** and has used the default delimiter value:

```
x=glfd(5,,,field_holder)
```

Please note that if the #PRO3 compiler directive is set, then the program will use the non-TAS file pointed to by the special file number as the buffer.

GSCHR(1,2,3)

PURPOSE

This function returns the character or color attribute from a specific location on the screen.

PARTS

1 f/c/e If ‘C’ the function will return the displayed character, if any, from the screen. If ‘A’, the function will return the attribute, or color number.

2 f/c/e The row value. The top row in the active window is 1.

3 f/c/e The column value. The leftmost column in the active window is 1.

RETURN TYPE

A If part 1 is 'C'
B If part 1 is 'A'

SAMPLE PROGRAM

GSCHRTST.SRC

HEX(1)

PURPOSE

This function converts a numeric value to a hexidecimal number and returns that number as an A type field.

PARTS

1 f/c/e The number to be converted.

RETURN TYPE

A

COMMENTS

All values are converted to R type first and then to hex.

EXAMPLE

```
? hex(100)
64

? hex(110)
6e
```

IFCR()

PURPOSE

This function checks if the last character entered by the user was the Enter key (just a CR - carriage return).

NO OTHER PARTS

RETURN TYPE

L If the last key pressed was the Enter key then the function will return .T., otherwise .F.

IFNA(1)

PURPOSE

This function will check if there is an active record for a specific file.

PARTS

1 f/c/e The **file_number** of the file to be checked.

RETURN TYPE

L If a record **IS NOT ACTIVE** for the file, the function will return .T., otherwise .F.

IIF(1,2,3)

PURPOSE

This function can return many different values depending on whether the logical expression provided returns .T. or .F.

PARTS

1 *lexpr* The expression to evaluate to determine what should be returned.

2 f/c/e If the *lexpr* in part 1 evaluates to .T. the function will return this value.

3 f/c/e If the *lexpr* evaluates to .F. the function will return this value.

RETURN TYPE

? The return type depends entirely on you.

COMMENTS

If you are only concerned about one type of return (i.e., only .T. or only .F.), the other part need not be included.

EXAMPLE

```
x = 1
? iif(x=1,,'False')
(the return above will be blank ("") )

? iif(x=1,'True')
True
```

SAMPLE PROGRAMS

IIFTEST.SRC, IIFTEST2.SRC

NOTE: You must be careful about what you put into the True/False options in this function. Due to the way the expression compiler and executor work, if the values to be returned are expressions also (including UDFs), they will be resolved (executed) before the value of the IIF() is determined. For example:

```
x=5
y=10
? iif(x>y,fill(x-y,' '),2)
```

If x>y was true then the routine above would use the difference to calculate how many space characters (' ') to print. However, since the inner expressions are evaluated before the IIF() an unexpected result could occur, as in this case. The program will attempt to make a string of spaces 65530 characters long (the FILL() function converts numbers to I type automatically). The proper way to set this up would be:

```
? fill(iif(x>y,x-y,2), ' ')
```

This would work properly each time.

INKEY(1)

PURPOSE

This function will check the keyboard and see if a character is waiting.

PARTS

1 f/c/e Normally, this function would just check the keyboard and then return. However, if you want the program to wait until a key is pressed then 'W' must be included here.

RETURN TYPE

I If you have not included the wait ('W') option and no character is waiting at the keyboard the value returned will be 0.

COMMENTS

If a character is waiting or one is entered, the ASCII representation value will be returned.

Once a key is pressed it will stay in the keyboard buffer until it is accessed by some procedure. The procedure could be a function such as this one or the normal TAS Professional 5.1 process, which checks the keyboard before each command is executed.

EXAMPLE

```
? inkey()
0       && no character was waiting
(This will return as soon as the keyboard is checked.)

? inkey('w')
32      && space bar was pressed
(This will return after a key was pressed.)
```

SAMPLE PROGRAMS

INKEY.SRC

INT(1)

PURPOSE

This function will return the integer portion only of a N type field.

PARTS

1 f/c/e The N type field to use. The original field is unchanged.

RETURN TYPE

N

COMMENTS

The decimal value is not rounded up but just eliminated. The decimal characters are still there; it's just that the value of those characters will be 0.

EXAMPLE

```
? int(2.935)
2.000
```

ISAL(1)

PURPOSE

This function will determine whether the first character of a field is an alphanumeric character (A-Z), or something else.

PARTS

1 f/c/e Field to check, must be of type A.

RETURN TYPE

L If the first character is from A-Z (upper or lower case), the function will return .T., otherwise .F.

EXAMPLE

```
x = 'SAMPLE'
? isal(x)
.T.

x = '!SAMPLE'
? isal(x)
.F.
```

SAMPLE PROGRAM

ISTEST.SRC

ISCLR()

PURPOSE

This function will determine whether the computer system on which the program is being run has a monochrome or color monitor active.

NO OTHER PARTS

RETURN TYPE

L If the system has a color monitor active the function will return .T., otherwise .F.

SAMPLE PROGRAM

ISCOLOR.SRC

ISLO(1)

PURPOSE

This function will determine whether the first character of a field is a lowercase alphanumeric character (a-z), or something else.

PARTS

1 f/c/e Field to check, must be of type A.

RETURN TYPE

L If the first character is from a-z (lower case), the function will return .T., otherwise .F.

EXAMPLE

```
x = 'SAMPLE'
? islo(x)
.F.

x = 'sample'
? islo(x)
.T.
```

SAMPLE PROGRAM

ISTEST.SRC

ISNUM(1)

PURPOSE

This function will determine whether the first character of a field is a numeric character (0-9), or something else.

PARTS

1 f/c/e Field to check, must be of type A.

RETURN TYPE

L If the first character is from 0-9, the function will return .T., otherwise .F.

EXAMPLE

```
x = ‘x10’
? isnum(x)
.F.

x = ’10x’
? isnum(x)
.T.
```

SAMPLE PROGRAM

ISTEST.SRC

ISUP(1)

PURPOSE

This function will determine whether the first character of a field is a uppercase alphanumeric character (A-Z), or something else.

PARTS

1 f/c/e Field to check, must be of type A.

RETURN TYPE

L If the first character is from A-Z (upper case), the function will return .T., otherwise .F.

EXAMPLE

```
x = ‘SAMPLE’
? isup(x)
.T.

x = ‘sample’
? isup(x)
.F.
```

SAMPLE PROGRAM

ISTEST.SRC

JUST(1,2)

PURPOSE

This function will return the field with the non-space characters moved to the left, right, or center.

PARTS

1 f/c/e The field to be justified, must be A type.

2 f/c/e Which direction: 'L' - Left, 'R' - Right, or 'C' - Center.

RETURN TYPE

A The original field remains unchanged.

EXAMPLE

```
x = 'ABCD    '
? just(x,'r')
'    ABCD'

? just(x,'C')
'  ABCD  '
```

SAMPLE PROGRAM

JUSTEST.SRC

LBLNME(1)

PURPOSE

This function will return the name of the label for a given value.

PARTS

1 f/c/e The number of the label name to be found.

RETURN TYPE

A The name of the label.

COMMENTS

This function will work properly only if the program has been compiled with the -D (debug) flag.

LBL_LNE(1)

PURPOSE

This function will return the internal (program) location for a given label number.

PARTS

1 f/c/e The number of the label location to be found.

RETURN TYPE

I The internal location of the label.

LCHR(1,2)

PURPOSE

This function will return the last displayable (non-space) character in an A type field or the position value of that character.

PARTS

1 f/c/e The field to be checked. Must be an A type field.

2 f/c/e If 'C' the function will return the character, if 'L' the location of the last character.

RETURN TYPE

A If 'C' the function returns a single A type character.
I If 'L' the function returns an I type value.

EXAMPLE

```
x = 'ABCD    '
? last_chr(x,'c')
D

? last_chr('ABCD    ','l')
4
```

SAMPLE PROGRAMS

ETST.SRC, LSTCTEST.SRC

LCKD(1,2)

PURPOSE

This function will allow you to check if a record is locked by another user for a specific file/key value without having to actually read the record.

PARTS

1 f/c/e The **file_number** of the file to be checked.

2 *key_expr* The key number to be used.

RETURN TYPE

L If the record is locked the function will return .T., otherwise .F.

LIKE(1,2)

PURPOSE

This function will return a logical value indicating whether the two fields are alike. You may use the skeleton characters **?** and ***** in the first field.

PARTS

1 f/c/e Comparison field 1. If you are going to use the skeleton characters (variable character specifiers) **?** or *****, they must be in this field. Must be an A type value.

2 f/c/e Comparison field 2. Must be an A type field.

RETURN TYPE

L If the fields match the function will return .T., otherwise .F.

COMMENTS

The character **?** will match any other single character in that same position in field 2. The character ***** will match all following characters from that position on. If comparison field 1 is shorter than field 2 and the last character in field 1 is NOT ***** the fields will not match and .F. will be returned. Differences in upper and lower case are treated as differences in characters.

EXAMPLE

```
? like('ABC','abc')          && same characters, different case
.F.

? like('this*','this is a test')      && last character in
.T.                                   && comp fld1 is '*' and
                                      && first chrs match

? like('th?s','this')                 && ? will match any single chr.
.T.
```

LISTF_CHRS()

PURPOSE

This function will return the characters entered by the user when searching for records during a LISTF command.

NO OTHER PARTS

RETURN TYPE

A The characters entered by the user using Fast Search.

COMMENTS

This function can be used at the end of a command to put the characters entered by the user into the ENTER field if a record wasn't found. For example:

```
LISTF bkar.custcode, ' ', bkar.custname . . .
IF bkar.custcode= ' ' THEN bkar.custcode=LISTF_CHRS()
RET
```

LOC(1,2,3,4,5,6)

PURPOSE

This is an ‘instring’ function. It will return the location of a string of characters (A type field) within another A type field.

PARTS

1 f/c/e Field to check for, must be A type.

2 f/c/e Field to check within, must be A type.

3 f/c/e The character position where the function will start checking. The first position (far left) is 1. If this value isn’t provided the default value is 1.

4 f/c/e The number of characters to check. If this value isn’t provided the function will check the entire field (part 2).

5 f/c/e If this is set to **Y**, the function will ignore differences in case (upper or lower). Default is **N**.

6 f/c/e The memory area number. This may be from 1 through 4. If this is set properly, you do not have to specify part 2, the field to check within. However, you do still need to put the comma in for place keeping, i.e., =loc('abc',,10,300,,4)

RETURN TYPE

I The first character position of the matched field. If no match is found the return is 0.

COMMENTS

If you are using a field as the search for value (part 1) make sure that you TRIM() the value first. This can be done within the LOC(). If you don't, the function will try to match all trailing spaces also. Obviously, if you are looking for the exact match, spaces and all, don't TRIM() it.

EXAMPLE

```
x = 'ABCDEFGHIJ'
? loc('CD',x)
3

? loc('CD',x,4,5)
0

define srch_fld type a size 10
srch_fld='CD'
? loc(trim(srch_fld,'t'),x)
3
```

SAMPLE PROGRAM

LOCTEST.SRC

LOG(1)

PURPOSE

This function will return the log base e of the numeric value specified.

PARTS

1 f/c/e The numeric value to use.

RETURN TYPE

N

EXAMPLE

```
? log(9000.000)
9.105
```

SAMPLE PROGRAM

LOGTEST.SRC

LOG10(1) (*DOS only*)

PURPOSE

This function will return the log base 10 of the numeric value specified.

PARTS

1 f/c/e The numeric value to use.

RETURN TYPE

N

EXAMPLE

```
? log10(9000.000)
3.954
```

SAMPLE PROGRAM

LOGTEST.SRC

LOW(1)

PURPOSE

This function returns the A type field specified in lower case characters.

PARTS

1 f/c/e The value to use. The original value is left unchanged.

RETURN TYPE

A If the receiving field is too short, the remaining characters will be truncated.

EXAMPLE

```
? low('ABcdEFG')
abcdefg
```

SAMPLE PROGRAM

LUPTEST.SRC

LROW()

PURPOSE

This function will return the current row from within a **LISTF** (List File) or **LISTM** (List Array) command.

NO OTHER PARTS

RETURN TYPE

I

COMMENTS

This function will return an accurate value only during the execution of the **enter** option in a **LISTF** or **LISTM** command. The value will be offset from the beginning of the window that was set up previous to the command. This means if the cursor is on the 4th item in the window the **LROW()** value will be 4.

Windows Note: In Windows the LROW() offset is from the beginning of the active window, whereas in DOS it is from the top of the main screen. The same applies to the current column value and this will change the values you might use in any command that would require column and row coordinates.

LSTCHR()

PURPOSE

This function will return the ASCII value of the last character entered by the user.

NO OTHER PARTS

RETURN TYPE

I Returns an integer value.

LTOC(1)

PURPOSE

This function converts a L type value to an A type value.

PARTS

1 f/c/e The logical value to convert.

RETURN TYPE

A

MAKE_DIR(1) (*Windows Only*)

PURPOSE

This function creates a new directory.

PARTS

1 f/c/e The name of the directory to create.

RETURN TYPE

L If the directory is created properly the function returns .T., otherwise it returns .F.

COMMENTS

This is the way you create a new directory in Windows. You cannot execute the DOS command MD.

MAX(1,2)

PURPOSE

This function returns the maximum value of the two specified.

PARTS

1 f/c/e The first value to check, may be of any type other than P or F.

2 f/c/e The second value to check, may be of any type other than P or F. Part 2 must be the same type as part 1.

RETURN TYPE

? This depends on the type of fields specified. The value returned will be of the same type.

EXAMPLE

```
? max(100,200)
200

? max('ABC','DEF')
DEF
```

MAX_COLS() (*Windows Only*)

PURPOSE

This function returns the maximum number of columns in the base window. Since the user can increase the size of the base window, by changing the appropriate value in TP5WIN.INI, stretching the window or maximizing it, this will give the programmer an easy way of always knowing what the maximum width of the window is.

NO OTHER PARTS

RETURN TYPE

I The maximum column value.

MAX_ROWS() (*Windows Only*)

PURPOSE

This function returns the maximum number of rows in the base window. Since the user can increase the size of the base window, by changing the appropriate value in TP5WIN.INI, stretching the window or maximizing it, this will give the programmer an easy way of always knowing what the maximum length of the window is.

NO OTHER PARTS

RETURN TYPE

I The maximum row value.

MCOL()

PURPOSE

This function returns the column number for the mouse's current location.

NO OTHER PARTS

RETURN TYPE

I The column number relative to the current window.

MDY(1,2)

PURPOSE

This function returns a date value in the form of Month, Day, Year.

PARTS

1 f/c/e Date value to use.

2 f/c/e If set to 'S', only the first three characters of the month value are used (e.g., Jan = January).

RETURN TYPE

A The size of the string returned depends on the month.

EXAMPLE

```
? mdy(date(),'s')
Jun 23, 94
```

MEM()

PURPOSE

This function returns the number of kbytes (each kbyte is 1024 bytes) of memory not currently being used and available.

NO OTHER PARTS

RETURN TYPE

I

EXAMPLE

```
? mem()
264             && 264k or 270,336 bytes
```

MID(1,2,3,4)

PURPOSE

This function will return a portion of a specified A type field.

PARTS

1 f/c/e The value to be parsed.

2 f/c/e The starting character position. The first character of an A type field is position 1.

3 f/c/e The number of characters to return. If the value given would go beyond the end of the original field, only those characters up to the end of the field will be passed.

4 f/c/e The memory area number. This may be from 1 through 4. If this is set properly, you do not have to specify part 1, the field to be parsed. However, you do still need to put the comma in for place keeping, i.e., =mid(,1,10,3)

RETURN TYPE

A If the receiving field is too short, the remaining characters will be truncated.

EXAMPLE

```
x = 'ABCDEFG'
? mid(x,2,2)
BC

? mid(x,6,10)
FG
```

MID_REC(1,2,3)

PURPOSE

This function will return a portion of an active record in a specified file.

PARTS

1 f/c/e The **file_number** of the file to use.

2 f/c/e The starting character position. The first character of a record is position 1.

3 f/c/e The number of characters to return. If the value given would go beyond the end of the record, only those characters up to the end of the record will be passed.

RETURN TYPE

A If the receiving field is too short, the remaining characters will be truncated.

MIN(1,2)

PURPOSE

This function returns the minimum value of the two specified.

PARTS

1 f/c/e The first value to check, may be of any type other than P or F.

2 f/c/e The second value to check, may be of any type other than P or F. Part 2 must be the same type as part 1.

RETURN TYPE

? This depends on the type of fields specified. The returned value will be of the same type.

EXAMPLE

```
? min(100,200)
100

? min('ABC','DEF')
ABC
```

MNTH(1)

PURPOSE

This function will return the month of a date as a number.

PARTS

1 f/c/e The date value to use, must be of D type.

RETURN TYPE

I The number range returned is from 1 to 12, January thru December.

SAMPLE PROGRAM

DATETEST.SRC

MOD(1,2)

PURPOSE

This function returns the division remainder of two specified numbers.

PARTS

1 f/c/e The dividend, where quotient = dividend / divisor. This must be a type N value.

2 f/c/e The divisor. This must be a type N value.

RETURN TYPE

N

EXAMPLE

```
? mod(100,33)
1                          && 100/33 = 3 with a remainder of 1

? mod(100.00,32.33)
3.01
```

MOUSE_ACT() (*DOS only*)

PURPOSE

This function returns information about mouse actions. Once you test for a specific action that flag is cleared until it happens again.

PARTS

1 f/c/e Mouse action to check.
- 0 - movement
- 1 - Left button down
- 2 - Left button up
- 3 - Right button down
- 4 - Right button up

RETURN TYPE

L If the action you are testing for has happened since the last time you checked the function will return .T. Once you check it clears that flag, so if you checked immediately again and no action had occurred since your last check the function would return .F.

COMMENTS

There are many situations where use of the mouse traps won't be effective and you need a different method for determining actions. You would use this function in those cases. A good example of this use is in the programming utilities for TAS Professional 5.1, i.e, TASEDPRG.SRC (editor), TASEDSCR.SRC (screen painter), TASEDRPT.SRC (report writer) and TASRPT.SRC (report/2 writer).

MOUSE_ON()

PURPOSE

This function returns whether or not the mouse is turned on in a program.

NO OTHER PARTS

RETURN TYPE

L If the mouse is on this function returns .T.

MROW()

PURPOSE

This function returns the row number for the mouse's current location.

NO OTHER PARTS

RETURN TYPE

I The row number relative to the current window.

NULL(1)

PURPOSE

This function checks for a null pointer value. Will work with both P and F type fields.

PARTS

1 f/c/e Pointer value to check.

RETURN TYPE

L If null the function will return .T., otherwise .F.

COMMENTS

This is the only way to check for a pointer value that doesn't point at anything.

NUMFLDS()

PURPOSE

This function will return the actual number of fields in a program.

NO OTHER PARTS

RETURN TYPE

I

NUMLBLS()

PURPOSE

This function will return the actual number of labels in a program.

NO OTHER PARTS

RETURN TYPE

I

OFST(1,2) *(DOS Only)*

PURPOSE

This function returns the offset for a field or memory area. This is the position of the first character in memory.

PARTS

1 fn/v Field to use.

2 f/c/e The memory area number. If this is set properly, you do not have to specify part 1. However, you do still need to put the comma in for place keeping, i.e., =loc(,1)

RETURN TYPE

R This returns the absolute (flat) memory location.

COMMENTS

Use this function, along with the **SEG()** function, to get the absolute location of any field in memory.

OPEN()

PURPOSE

This function will return the appropriate TAS Professional 5.1 error message number for the last **OPEN** (or **OPENV**) command.

NO OTHER PARTS

RETURN TYPE

I

COMMENTS

If the file was opened properly this function will return 0. If the file was a Btrieve type file the function **FLERR()** will return the Btrieve error number.

OS()

PURPOSE

This function returns the operating system name and version number.

NO OTHER PARTS

RETURN TYPE

A The size of the returned value depends on the name of the operating system and the version number.

EXAMPLE

```
? OS()
DOS 5.1
```

PCOL()

PURPOSE

This function returns the current column from the printer.

NO OTHER PARTS

RETURN TYPE

I

EXAMPLE

```
? '' print_to printer              && This will do a cr.
? 'abcd' print_to printer no_cr    && This will not.
? pcol()
5
```

PERR()

PURPOSE

This function returns the error number of the last program error.

NO OTHER PARTS

RETURN TYPE

I

COMMENTS

Once a program error occurs the appropriate error number will remain until it is cleared by the **CLEAR PRG_ERR** command or the current program terminates. The error number returned will correspond to the standard TAS Professional runtime error numbers. For more information please refer to **Chapter 10, Runtime Errors**.

NOTE: The value 999 is returned if the user has quit a subsequent program by pressing the Esc key.

PI()

PURPOSE

This function returns the value of the constant PI.

NO OTHER PARTS

RETURN TYPE

N The constant is maintained internally up to 8 decimal characters.

EXAMPLE

```
? pi()
3.14159265
```

PRGLNE()

PURPOSE

This function will return the actual number within the source file (or library routine) for the line that will be executed next.

NO OTHER PARTS

RETURN TYPE

A

COMMENTS

This function will work properly only if the program has been compiled with the -D (debug) flag.

PRGNME()

PURPOSE

This function returns the name of the source file (or library routine) that contains the line that will be executed next.

NO OTHER PARTS

RETURN TYPE

A

COMMENTS

This function will work properly only if the program has been compiled with the -D (debug) flag.

PRINT_CANCEL() (*Windows Only*)

PURPOSE

When the user is given the choices of where to print under Windows one of those choices is Cancel. If they choose to cancel the output then this function will return .T. By itself, this function will NOT cancel the report. It is up to the programmer to add the 1 or 2 lines of code necessary to actually exit the report. This will just notify the programmer of the user's choice.

NO OTHER PARTS

RETURN TYPE

L If the user chooses Cancel in the Choose Output options this function will return .T.

COMMENTS

For more information please see **Chapter 11 - Windows Programming**..

PRINTER_NAME() (*Windows Only*)

PURPOSE

Every printer in Windows has a distinct name. This function will return the name of the currently chosen printer..

NO OTHER PARTS

RETURN TYPE

A The name of the currently active or default printer.

PROP(1)

PURPOSE

This function is used to return a person’s name in a more proper format.

PARTS

1 f/c/e The value to be reformatted. The original value is not changed.

RETURN TYPE

A

EXAMPLE

```
? proper('JOHN Q. PUBLIC')
John Q. Public

? proper ('aBnEr Q. aNyDaY iii')
Abner Q. Anyday III
```

SAMPLE PROGRAM

LUPTEST.SRC

PROW()

PURPOSE

This function will return the current row from the printer.

NO OTHER PARTS

RETURN TYPE

I

EXAMPLE

```
top_of_form print_to printer
? prow()
1
```

PSET(1)

PURPOSE

This function will return the currently set value for printer width, printable lines, or maximum lines.

PARTS

1 f/c/e What to get: 'W' - printer width, 'P' - printable lines, or 'M' - maximum lines.

RETURN TYPE

I The current value depending on the option.

EXAMPLE

```
? pset('w')
80

? pset('m')
66
```

PSTAT() *(DOS Only)*

PURPOSE

This function returns the printer status, indicating whether or not it is ready for printing.

NO OTHER PARTS

RETURN TYPE

I 0 - Printer is ready for output.
1 - Printer is out of paper.
2 - Printer is busy.
3 - There is an i/o error between the computer and the printer.
4 - There has been some other kind of printer error.

SAMPLE PROGRAM

PSTATUS.SRC

PWHR()

PURPOSE

This function will return a character representing the current default output device.

NO OTHER PARTS

RETURN TYPE

A 'A' - Ask at runtime.
'S' - Send output to screen.
'P' - Send output to printer.
'D' - Send output to disk file.
'W' - Send output to Windows List routine.

RCN(1)

PURPOSE

This function returns the record number for a specified file.

PARTS

1 f/c/e The **file_number** of the file to use.

RETURN TYPE

R

COMMENTS

If there is not an active record for the specified file, the function will return 0.

This function can be used for both TAS and non-TAS files.

NOTE: The number returned by this function is not the true 'record number,' but the offset of the record within the file.

SAMPLE PROGRAM

RCNTEST.SRC

RECORD_CHR() (*DOS only*)

PURPOSE

This function returns the number of characters recorded with the **AUTO_RUN** command when the **RECORD** option is set.

NO OTHER PARTS

RETURN TYPE

I

REC_PTR(1)

PURPOSE

This function returns a pointer (type P) to a record buffer. You can then use that pointer to get values from that record buffer rather than having to use the **ADD** command.

PARTS

1 f/c/e The **file_number** of the file to use.

RETURN TYPE

P

RENF(1,2)

PURPOSE

This function can be used to rename a file. It will return a value indicating whether or not the renaming was successful.

PARTS

1 f/c/e The current file name, must include full path and extension.

2 f/c/e The new file name, must include full path and extension.

RETURN TYPE

L If the rename operation is successful the function will return .T., if not, .F..

COMMENTS

You may 'move' the file from one path to another by using the appropriate path values.

RNDM()

PURPOSE

This function will generate a random number.

NO OTHER PARTS

RETURN TYPE

I

COMMENTS

The first time this function is called in any program the random number generator is 'seeded' with the time value. This makes sure that the value returned will always be different for multiple groups of random numbers generated.

SAMPLE PROGRAM

RANDTEST.SRC

ROUND(1,2)

PURPOSE

This function will return an N type value rounded up to a specified number of decimal characters.

PARTS

1 f/c/e The N type field value. The original value is unchanged.

2 f/c/e The number of significant decimal characters to keep. This cannot be more than 8. If the number specified is more than the defined value for the field being rounded, there will be no effect.

RETURN TYPE

N The number of decimal characters displayed is unchanged; only the value is affected.

EXAMPLE

```
? round(29.3251,2)
29.33
? round(29.3247,2)
29.32
? round(29.775,0)
30.000
```

ROW()

PURPOSE

This function will return the current row from the screen.

NO OTHER PARTS

RETURN TYPE

I

EXAMPLE

```
clear screen
? row()
1
```

RSIZE(1)

PURPOSE

This function will return the record size for a specified file.

PARTS

1 f/c/e The **file_number** for the specified file.

RETURN TYPE

I This is the maximum record size.

EXAMPLE

```
define hndl type i
openv 'errmsg' fnum hndl
? rsize(hndl)
324
```

RTOD(1)

PURPOSE

This function will convert a value in an R type field to a date (D type) value.

PARTS

1 f/c/e The R type value to convert. The original field is not changed.

RETURN TYPE

D

SAMPLE PROGRAM

DATETEST.SRC

RTOT(1)

PURPOSE

This function will convert a value in an R type field to a time (T type) value.

PARTS

1 f/c/e The R type value to convert. The original field is not changed.

RETURN TYPE

T

SAMPLE PROGRAM

TIMETEST.SRC

RTP(1)

PURPOSE

This function will convert a value in an R type field to a pointer (P type) value.

PARTS

1 f/c/e The R type value to convert. The original field is not changed.

RETURN TYPE

P

COMMENTS

A P type pointer normally contains information about the segment and location values as well as the field type, display size, etc. The result of this expression will only contain the segment and location values.

SEG(1)

PURPOSE

This function no longer applies! It is only used internally by TAS Professional 5.1. You can use the OFST() function to get the absolute location of any field in memory.

SIGN(1)

PURPOSE

This function will return an N type value indicating the sign of the field value specified.

PARTS

1 f/c/e N type field to check.

RETURN TYPE

N If the value being checked is < 0 the returned value is -1.00
If the value being checked is = 0 the returned value is 0.00
If the value being checked is > 0 the returned value is 1.00

EXAMPLE

```
? sign(-25.29)
-1.00
? sign(0.00)
 0.00
?sign(25.29)
 1.00
```

SIN(1)

PURPOSE

This function returns the sine of the N type field specified.

PARTS

1 f/c/e The N value.

RETURN TYPE

N The value returned is in radians.

EXAMPLE

```
? sin(1.539)
0.9994954
```

SIZE(1,2,3)

PURPOSE

This function will return the size of a specified field.

PARTS

1 f/c/e The field being checked.

2 f/c/e 'I' - Return the internal size.
'D' - Return the display size.
'A' - Return the actual size only if the field being checked is type A.

3 f/c/e The memory area number. This may be from 1 through 4. If this is set properly, you do not have to specify part 1, the field to check. However, you do still need to put the comma in for place keeping, i.e., =size(,'a',3)

RETURN TYPE

I

EXAMPLE

```
define x type n size 12 dec 2
x = 100.10
? size(x,'i')
8
? size(x,'d')
12
? size(x,'a')
0

define x type a size 20
x = 'ABCD'
? size(x,'i')
20
? size(x,'d')
20
? size(x,'a')
4
```

SNDX(1)

PURPOSE

This function returns a 4 character value representing the 'sound' of the field. These values can be used in comparing fields (see the **DIFF()** function also).

PARTS

1 f/c/e The A type field to be converted. The original value is unchanged.

RETURN TYPE

A A 4 character string is returned.

EXAMPLE

```
? sndx('translate')
T652
? sndx('trnslt')
T652
? sndx('John')
J500
? sndx('Paul')
P400
```

SQRT(1)

PURPOSE

This function will return the square root of the value specified.

PARTS

1 f/c/e The value to use. Must be of N type.

RETURN TYPE

N

EXAMPLE

```
? sqrt(4)
2

? sqrt(99.325)
9.966
```

STR(1,2,3)

PURPOSE

This function will return a numeric (B, I or N type) value as a string (A type).

PARTS

1 f/c/e The numeric field value. The original field is unchanged.

2 f/c/e The number of characters in the final field.

3 f/c/e The number of decimal characters in the final field.

RETURN TYPE

A The size of this field depends on part 2.

EXAMPLE

```
? str(195.32596,6,2)
195.33
```

TAN(1)

PURPOSE

This function returns the tangent of the N type field specified.

PARTS

1 f/c/e The N value.

RETURN TYPE

N The return value is in radians.

EXAMPLE

```
? tan(1.539)
31.43957419
```

TEST(1,2)

PURPOSE

This function will determine whether a bit or bits are set in a byte.

PARTS

1 fn/v Field to check. The function will only test the first byte/character.

2 f/c/e This value determines which bits to check. To set the proper value use the following guide:

Bit#	7	6	5	4	3	2	1	0
Value	128	64	32	16	8	4	2	1

To test for bits 7 and 4 (to see if the bits are set to 1 instead of 0) the test value would be 144 (128+16). The function will ignore the other 6 bits.

RETURN TYPE

L If the bit(s) is (are) set (1 not 0), the function will return .T. Otherwise it will return .F.

COMMENTS

Use this function with the **PEEK** and **INT** commands to check the results when you need to know the settings of individual bits.

SAMPLE PROGRAM

CAPSLOCK.SRC

TIME()

PURPOSE

This function will return the current system time.

NO OTHER PARTS

RETURN TYPE

T

COMMENTS

This is the same time that is returned if the user enters TIME at the DOS prompt.

SAMPLE PROGRAM

TIMETEST.SRC

TPATH()

PURPOSE

This function will return the default TAS Professional 5.1 path value as set in the DOS SET TAS50= process.

NO OTHER PARTS

RETURN TYPE

A

COMMENTS

If the SET TAS50= command has not been executed at the DOS prompt (meaning there is no default path), the function will return a null string.

TRC(1)

PURPOSE

This function returns the number of records in a specified file.

PARTS

1 f/c/e The **file_number** of the file to use.

RETURN TYPE

R

SAMPLE PROGRAM

RCNTEST.SRC

TRIM(1,2)

PURPOSE

This function will return an A type field with the leading or trailing spaces deleted.

PARTS

1 f/c/e The value to be converted. The original field is not changed.

2 f/c/e 'T' - trim trailing spaces; this is the default value.
'L' - trim leading spaces.

RETURN TYPE

A If the receiving field is not long enough, the remaining characters will be truncated.

COMMENTS

You would use this function when concantenating (adding) A type fields together. If the spaces aren't trimmed, there will probably be extra spaces in the resulting value. This function can get rid of them. You can also use the '*' when concatenating the fields. For more information please see **Chapter 1, Installation and General Information.**

TTOC(1)

PURPOSE

This function will convert a time (type T) value to a string (type A).

PARTS

1 f/c/e The time value to use. The original field is left unchanged.

RETURN TYPE

A If the receiving field is not long enough the remaining characters will be truncated.

SAMPLE PROGRAM

TIMETEST.SRC

TTOF(1)

PURPOSE

This function will convert a time (type T) value to a numeric (type N) value.

PARTS

1 f/c/e The time value to use. The original field is left unchanged.

RETURN TYPE

N

SAMPLE PROGRAM

TIMETEST.SRC

TTOR(1)

PURPOSE

This function will convert a time (type T) value to a record/long integer (type R) value.

PARTS

1 f/c/e The time value to use. The original field is left unchanged.

RETURN TYPE

R

SAMPLE PROGRAM

TIMETEST.SRC

UP(1)

PURPOSE

This function returns the string value (type A) specified in upper case characters.

PARTS

1 f/c/e The value to use. The original value remains unchanged.

RETURN TYPE

A If the receiving field is too short, the remaining characters will be truncated.

EXAMPLE

```
? up('abcdEFG')
ABCDEFG
```

SAMPLE PROGRAM

LUPTEST.SRC

VAL(1)

PURPOSE

This function converts a string value (type A) to a numeric value (type N).

PARTS

1 f/c/e The value to be converted. The original value remains unchanged.

RETURN TYPE

N

EXAMPLE

```
? val('100.2')
   100.20

? val('abcd')
         0.00
```

SAMPLE PROGRAM

TRANSTST.SRC

VARREAD()

PURPOSE

This function will return the name of the last, or current **ENTER** field.

NO OTHER PARTS

RETURN TYPE

A A 15 character field is returned.

COMMENTS

If the field being entered is an array, only the major part (the field name of the array) will be displayed.

VER()

PURPOSE

This function will return the current TAS Professional 5.1 name and version number.

NO OTHER PARTS

RETURN TYPE

A

EXAMPLE

```
? ver()
TAS Professional 5.1
```

WIN_LASER_PRT() (*Windows Only*)

PURPOSE

In some programs it is important for the report to know whether it is being printed on a laser printer or not. Since laser printers will only print 60 lines normally you may need to tighten up the output. This function will return the value set for this printer in the TP5WIN.INI file.

NO OTHER PARTS

RETURN TYPE

L If the current (active) printer is set as type laser this will return .T.

COMMENTS

For more information about printing in Windows please refer to **Chapter 11 - Windows Programming**.

WINDOW_PTR() (*Windows Only*)

PURPOSE

This function will return the handle or pointer to the currently active window.

NO OTHER PARTS

RETURN TYPE

R

COMMENTS

The value returned by this function can be used in conjuction with the WINACT command. By getting the window handle you can use the WINACT command even though you didn't set the window up with the WINDEF command. If you are manipulating several windows on the screen at the same time you may find this more convenient. For more information about windows in Windows please refer to **Chapter 11 - Windows Programming**.

WINDOWS() (*Windows Only*)

PURPOSE

This function allows you to write programs that will run under Windows and DOS at the same time by letting the program know when it's running in Windows.

NO OTHER PARTS

RETURN TYPE

L If the program is running in the Windows environment this will return .T.

WRAP(1,2,3)

PURPOSE

This function will wrap a long string value (type A) so that it may be output to the screen or printer as a block of characters. As opposed to the **WRAP** command this function would be used in a report format or in a **PRINT MESSAGE** command.

PARTS

1 f/c/e The string value to use. The original field remains unchanged.

2 f/c/e How many columns (characters) wide to make the block.

3 f/c/e Whether or not to trim the trailing spaces at the end of the block. This will allow the next line to be output immediately after the end of the last non-space line.

'Y' - trim trailing spaces. This is the default value.
'N' - don't trim spaces.

RETURN TYPE

A If the receiving field is too short, the remaining characters will be truncated.

COMMENTS

You may have up to 10 **WRAP()** functions on the same print line. Each one will print the appropriate number of lines.

In the default mode the program will trim any trailing blanks in the field so that blank lines are not printed. However, a minimum number of characters will be printed within the space for the block even if there are not enough characters to fill a single line. If you want all the lines to print, blank or not, set option 3 (trim trailing spaces) to 'N'.

Also, note that any subsequent lines will start at the same starting column number as the first line.

EXAMPLE

```
field1 = 'abcd'
field2 = 'now is the time for all good men to come to the aid of their party.'
field3 = 100.00
? field1,'  ',wrap(field2,20),' ',field3

abcd   now is the time for   100.00
       all good men to come
        to the aid of their
        party.
```

SAMPLE PROGRAMS

WRAPTEST.SRC, WRPTEST2.SRC

WRAPL(1)

PURPOSE

This function returns a line from a field that has been previously set up with the **WRAP** command.

PARTS

1 f/c/e The line number.

RETURN TYPE

A If the receiving field is not long enough, the remaining characters will be truncated.

WRAPO(1)

PURPOSE

This function returns the offset of the beginning of a line from a field that has been previously set up with the **WRAP** command.

PARTS

1 f/c/e The line number.

RETURN TYPE

I The first character of any field is position 1.

WRAPS(1)

PURPOSE

This function returns the size of a line from a field that has been previously set up with the **WRAP** command.

PARTS

1 f/c/e The line number.

RETURN TYPE

I

YEAR(1)

PURPOSE

This function returns the year of a date value (type D) as a string (type A).

PARTS

1 f/c/e The date value to use.

RETURN TYPE

A This function returns a 4 character string.

EXAMPLE

```
? year(date())
1994
```

TAS
PROFESSIONAL
VERSION 5.1

Chapter 7

Structured Programming Commands

INTENTIONALLY BLANK

INTRODUCTION

There are five different structures that can be used within a program. Each one is a little different and would be used in specific situations. Often the decision to use a specific structure is strictly due to your likes or dislikes.

These structures are very solid. By this we mean that you could do a GOTO from one section of a structure into a totally different section or even GOTO a different structure, and the program would know just what to do. Of course, we're not suggesting you do this! We only want you to know that if you set the structure up properly, it should always work as expected.

In the examples given below, the beginning and ending lines of the structure are given, and the possible loop options are shown within the structure. There is no requirement to use any of the 'intrastructure' options; however, the beginning and ending commands are required to form the structure properly. Each structure type can be nested up to 20 deep. For example:

```
IF x = y
  IF z = 1
    IF q = 1 .A. test = 'abc'
          ... etc. till there are 20 IF/ENDIF pairs
    ENDIF
  ENDIF
ENDIF
```

You can insert other structure types, and those other structures will not count towards the 20 total. The indentations above are just for example only and are not required.

FOR/NEXT

This is a loop that will continue until the stop level is reached. The general layout of the structure is :

FOR(*counter;start_value;stop_value;step_value*)

```
FEXIT
FEXIT_IF lexpr
FLOOP
FLOOP_IF lexpr
```

NEXT

FOR *etc.*

There are four different sections to this command. They are the counter field, a starting value, a stop value and a step value.

counter

The name of the field used as a loop counter. The **counter** can be I or R type.

start_value

The first value to be used in the loop. The **counter** field is initialized to this value the first time through.

stop_value

When the **counter** reaches this value the program knows to exit the loop at the **NEXT** command.

step_value

How much to add or subtract from **counter** each time through the loop. If you wish to subtract from **counter** (i.e., decrement), precede the value with the minus sign. For example, a **step_value** of -1 will reduce the **counter** value by 1 each time the loop is executed.

Each time through the loop (except the first time) the program adds (or subtracts if **step_value** is negative) the **step_value**. If the result is greater than the **stop_value** (in the case of a positive **step_value**) or less than the **stop_value** (in the case of a negative **step_value**), the program will transfer control to the line after the appropriate **NEXT** command.

In TAS Professional 3.0, if you pressed the UP ARROW key (assuming no traps), and the program encountered a FOR/NEXT loop, the **counter** would be decremented or incremented as appropriate. This feature is also available in TAS Professional 5.1. You do not need to set the #PRO3 compiler directive to activate this feature. Please be aware that in TAS Professional 5.1, the **counter** is always +1 (or -1) greater (or less) than the **stop_value** at the end of the loop. If the #PRO3 directive is set, the **counter** will be equal to the **stop_value** when the loop is finished.

FEXIT
FEXIT_IF *lexpr*

These elements of the **FOR/NEXT** loop will allow you to exit the loop before the **counter** reaches the **stop_value**. The **FEXIT** option exits the loop immediately. The **FEXIT_IF** *lexpr* will exit the loop if the expression resolves to .T.; otherwise nothing will happen. The *lexpr* must be in the form of a comparison expression. For example:

```
FEXIT_IF x = 1
```

The **FEXIT** and **FEXIT_IF** (assuming the *lexpr* resolves to .T.) will transfer control to the next line after the **NEXT** command.

As many **FEXIT** and **FEXIT_IF** commands may be included in a single **FOR/NEXT** loop as the programmer desires.

FLOOP
FLOOP_IF *lexpr*

These elements of the **FOR/NEXT** loop will allow the program to transfer control to the beginning of the loop (**FOR** command) without having to reach the **NEXT** command. The **FLOOP** option transfers control immediately. The **FLOOP_IF** *lexpr* will transfer control if the expression resolves to .T.; otherwise nothing will happen. The *lexpr* must be in the form of a comparison expression. For example:

```
FLOOP_IF x = 1
```

As many **FLOOP** and **FLOOP_IF** commands may be included in a single **FOR/NEXT** loop as the programmer desires.

NEXT

This is the final step in the **FOR/NEXT** loop. When this command is executed control is returned to the **FOR** command line. Each **FOR** must have a **NEXT** to finish the loop. If the **NEXT** is not included an error message will be displayed during compilation.

IF/ENDIF

This is a non-loop structure. Each item is only tested once. The general layout of the structure is:

IF *lexpr*

ELSE_IF *lexpr*
ELSE

ENDIF

IF *lexpr*

This is the beginning of the **IF/ENDIF** structure. The *lexpr* must be in the form of a comparison expression. For example:

IF x = 1

If the *lexpr* returns .T., the lines after the **IF** command and continuing until the next **ELSE_IF**, **ELSE** or **ENDIF** command will be executed. Otherwise control will be passed to the next subsequent **ELSE_IF, ELSE,** or **ENDIF** command.

ELSE_IF *lexpr*

If the original **IF** or a previous **ELSE_IF** comparison expression returned .F., the program will keep testing each subsequent **ELSE_IF** command until one is found that returns .T. or until an **ELSE** command is found, or the appropriate **ENDIF** command is found. If the **ELSE_IF** *lexpr* returns .T., the lines after the command will be executed until the next **ELSE_IF**, **ELSE**, or **ENDIF** command.

There can be any number of **ELSE_IF** commands within an **IF/ENDIF** structure. However, only one will be executed, and that will be the first one for which the *lexpr* resolves to .T. Once the program lines have been executed, control is transferred to the appropriate **ENDIF** command.

ELSE

If the original **IF** and the previous **ELSE_IF** commands all resolved to .F., the program will always execute the lines after this command and until the **ENDIF**. This is a way of providing for actions to be taken if all the other **IF** tests have failed.

ENDIF

This is the end of the **IF/ENDIF** structure. Each **IF** must have an **ENDIF** to finish the structure. If the **ENDIF** is not included an error message will be displayed during compilation.

The following example shows the proper way to use this structure:

```
IF X = 100
   ...
   ...     && group 1
   ...
  ELSE_IF X = 200
   ...
   ...     && group 2
   ...
  ELSE_IF X = 300
   ...
   ...     && group 3
   ...
  ELSE
   ...
   ...     && group 4
   ...
ENDIF
```

(The indentations are for clarity only.) In the example above if x = 100 the lines in group 1 will be executed, if x = 200 group 2, and if x = 300 group 3. Otherwise the command lines in group 4 will be executed.

SELECT

This is a non-loop structure. Each item is only tested once. The general layout of the structure is:

SELECT *expression*

CASE *int_value*
CASE *int_value*
...
CASE *int_value*
OTHERWISE

ENDC

SELECT *expression*

This is the beginning of the Case structure. The expression must resolve to an I type value. The value will be used in comparisons by the individual **CASE** commands.

This command is generally used in menu type situations where there is a list of options.

CASE *int_value*

This part represents the choice in the **SELECT** structure. The *int_value* must be in the form of an I type constant. For example:

```
CASE 1
```

If the *int_value* matches the value generated by the appropriate **SELECT** *expression*, the lines after the **CASE** command and continuing until the next **CASE**, **OTHERWISE** or **ENDC**

command will be executed. Otherwise control will be passed to the next subsequent **CASE, OTHERWISE,** or **ENDC** command.

There can be any number of **CASE** commands within a **SELECT/CASE/ENDC** structure. However, only one will be executed, and that will be the first one for which the *int_value* matches the value in the **SELECT** *expression.*

OTHERWISE

If none of the previous **CASE** values match that in the **SELECT** command, the program will always execute the lines between this **OTHERWISE** and the **ENDC**. This is much like the **ELSE** option within an **IF/ENDIF** structure.

ENDCASE

This is the end of the **SELECT/CASE/ENDC** structure. Each **SELECT** must have an **ENDC** to finish the structure. If the **ENDC** is not included an error message will be displayed during compilation.

The following example shows the proper way to use this structure:

```
SELECT counter     && counter might be a value received from a menu.
  CASE 1
    ...
    ...      && group 1
    ...
  CASE 2
    ...
    ...      && group 2
    ...
  CASE 3
    ...
    ...      && group 3
    ...
  OTHERWISE
    ...
    ...      && group 4
    ...
ENDC
```

(The indentations are for clarity only.) In the example above if counter = 1 the lines in group 1 will be executed, if counter = 2 group 2, and if counter = 3 group 3. Otherwise the command lines in group 4 will be executed.

If you are missing an **ENDC** (End Case) command in a **SELECT/CASE** structure you will get the If without Endif error message during the compile process.

SCAN

This structure is a specialized type of loop. The **SCAN** command combines a **FIND** command with a **WHILE** loop. You can specify the file, key, starting value, scope, and for/while filters for the file searches. The first time the **SCAN** command is executed, the first appropriate record will be read. If there is a start value, the record will be found. If there isn't the program will start with the record in memory, find the first record in the file or do whatever else the programmer has specified as determined by the settings of the various options. Then, each time through the loop, the program will find

the next record that matches the criteria of the different options. When the last record is found the program will transfer control to the line immediately after the **ENDS** command. When dealing with searches through files, **SCAN/ENDS** will probably become your most widely used command(s). With the **FOR** and **WHILE** filters you have absolute control over the records that are active within the loop. Any TAS Professional 5.1 command may be used within the loop.

The general layout of the structure is :

SCAN *options*

SEXIT
SEXIT_IF *lexpr*
SLOOP
SLOOP_IF *lexpr*

ENDS

SCAN *options*

The options in a **SCAN** command are:

filename/@file_number

Optional - The name or number of the file to be used. If you do not include this option, the program will look for a default search file set in the **SEARCH FILE** command. If a default hasn't been previously set, the program will report an error and the command will be skipped. Setting this option will override any default value set in the **SEARCH FILE** command.

keyname/@key_number

Optional - Set this option to the appropriate value to read the records in the file in a particular order. If you do not include this option, the program will look for a default key specified in the **SEARCH FILE** command. If a default hasn't been previously set, the program will report an error and the command will be skipped. Setting this option will override any default value set in the **SEARCH FILE** command.

start_value

Optional - The value to use as the beginning record. This is similar to doing a **FIND** before the command. If the index you're searching on has multiple segments you may list the proper values for each of the segments, separating them with commas. This will allow you to specify the exact type for each of the segments. For example, suppose you're searching on an index that has two segments: a 5 character alpha and an I type field. Each record you want has the same alpha but the I field will change, and is in order. In the first record the I type field has a value of 0. The following would find the first record for that group, if it exists:

start 'ABCDE',0!

Notice that we separate the values with a comma. The only requirement is that the values be of the same type (and size) as the segments in the key. In this case we put the I type constant specifier ('!') after the 0 to make sure that it is passed as an I type field.

scope

Optional - This option may help determine how many records are scanned. For more information about the **scope** specifier, please see the general information at the beginning of **Chapter 5, Command Reference**.

scope_value

Optional - If **scope** is set to **N** or **F** this is the number of records to scan.

for_filter_expression

Optional - You would use this option to restrict the records scanned in this command. If the expression did not resolve to .T., the search would continue until the program reaches the end of file and then will quit.

while_filter_expression

Optional - This option also restricts the records scanned. However, the first time the expression resolves to .F., the program stops reading records and continues to the command immediately after the corresponding **ENDS**. Thus if the program is reading records in a certain order (by a specific key) and the expression returns .F., then the program knows that all the appropriate records have been read and there is no need to continue reading. For example: Suppose you want to export all the invoice records for a specific customer. The first record for that customer is found in the invoice file either by using the **start_value** option within this command, or through the **FIND** command before executing this command. Then the **while_filter_expression** would be:

INV_CUST_CODE = CUSTOMER_CODE

This assumes that there is a field called INV_CUST_CODE in the invoice file and a similar field in the customer file called CUSTOMER_CODE. The **keyname/@key_number** option must be set to the proper value so that the records are read in CUSTOMER_CODE order, or you must have set the **SEARCH FILE** previous to this command. When the first invoice record for the next customer is read, the program will stop reading records.

NOTE: If there is no **start_value** you must set the **scope** value to *R*or *N xxx*. If this is not done, the program will find the first record in the file and the **while_filter_expression** option will probably fail the first time. However, if the **start_value** option is used you can ignore this requirement.

SEXIT
SEXIT_IF *lexpr*

These elements of the **SCAN/ENDS** loop will allow you to exit the loop. The **SEXIT** option exits the loop immediately. The **SEXIT_IF** *lexpr* will exit the loop if the expression resolves to .T.; otherwise nothing will happen. The *lexpr* must be in the form of a comparison expression. For example:

SEXIT_IF x = 1

The **SEXIT** and **SEXIT_IF** (assuming the *lexpr* resolves to .T.) will transfer control to the next line after the **ENDS** command.

As many **SEXIT** and **SEXIT_IF** commands may be included in a single **SCAN/ENDS** loop as you desire.

SLOOP
SLOOP_IF *lexpr*

These elements of the **SCAN/ENDS** loop will allow you to transfer control to the beginning of the loop (**SCAN** command) without having to reach the **ENDS** command. The **SLOOP** option transfers control immediately. The **SLOOP_IF** *lexpr* will transfer control if the expression resolves to .T.; otherwise nothing will happen. The *lexpr* must be in the form of a comparison expression. For example:

SLOOP_IF x = 1

The **SCAN** comparison expression is executed and then, if the expression resolves to .T., the loop will continue.

As many **SLOOP**, **SLOOP_IF** commands may be included in a single **SCAN/ENDS** loop as you desire.

ENDS

This is the final step in the **SCAN/ENDS** loop. When this command is executed, control is returned to the **SCAN** command line. Each **SCAN** must have an **ENDS** to finish the loop. If the **ENDS** is not included an error message will be displayed during compilation.

WHILE

This structure is a loop that will continue until the **WHILE** *lexpr* returns .F. or the loop is exited. The general layout of the structure is :

WHILE *lexpr*

EXIT
EXIT_IF *lexpr*
LOOP
LOOP_IF *lexpr*

ENDW

WHILE *lexpr*

The *lexpr* in this command must be in the form of a comparison expression. For example:

WHILE x = 1

If this expression resolves to .T., the line after the **WHILE** command will be executed; otherwise control will be transferred to the line after the **ENDW** command. This expression is checked each time the loop is executed.

The programmer can stay in the loop until an **EXIT** or **EXIT_IF** command is executed by making sure this command will always return .T. This can be accomplished easily by the following:

WHILE .T.

If an **EXIT** or **EXIT_IF** command is never executed the loop will continue on and will never quit. (Unless, of course, an error occurs.)

EXIT
EXIT_IF *lexpr*

These elements of the **WHILE/ENDW** loop will allow you to exit the loop. The **EXIT** option exits the loop immediately. The **EXIT_IF** *lexpr* will exit the loop if the expression resolves to .T.; otherwise nothing will happen. The *lexpr* must be in the form of a comparison expression. For example:

```
EXIT_IF x = 1
```

The **EXIT** and **EXIT_IF** (assuming the *lexpr* resolves to .T.) will transfer control to the next line after the **ENDW** command.

As many **EXIT** and **EXIT_IF** commands may be included in a single **WHILE/ENDW** loop as you desire.

LOOP
LOOP_IF *lexpr*

These elements of the **WHILE/ENDW** loop will allow you to transfer control to the beginning of the loop (**WHILE** command) without having to reach the **ENDW** command. The **LOOP** option transfers control immediately. The **LOOP_IF** *lexpr* will transfer control if the expression resolves to .T.; otherwise nothing will happen. The *lexpr* must be in the form of a comparison expression. For example:

```
LOOP_IF x = 1
```

The **WHILE** comparison expression is executed and then, if the expression resolves to .T., the loop will continue.

As many **LOOP**, **LOOP_IF** commands may be included in a single **WHILE/ENDW** loop as you desire.

ENDW

This is the final step in the **WHILE/END** loop. When this command is executed control is returned to the **WHILE** command line. Each **WHILE** must have a **ENDW** to finish the loop. If the **ENDW** is not included an error message will be displayed during compilation.

INTENTIONALLY BLANK

TAS
PROFESSIONAL
VERSION 5.1

Chapter 8

Conversion
TAS Professional 3.0 to 5.1
TAS Professional 4.0 to 5.1

INTENTIONALLY BLANK

INTRODUCTION

This chapter covers the process of converting from TAS Professional 3.0 or TAS Professional 4.0 to TAS Professional 5.1. The amount of data to be converted, in the case of TAS Professional 3.0, or the number of programs in both cases will determine the time involved to complete this process. In converting both data and programs automated procedures have been provided to help you with this task.

We appreciate your continued use of Business Tools products and hope you find this process easier than those in the past.

Converting from TAS Professional 3.0 to 5.1

With this new release we have reintegrated many of the commands and features of TAS Professional 3.0 to ease the conversion to this new version. These include allowing the characters #!$% in field names, continuing the old form of BCD numbers, creating the 4 external memory areas and several commands that we superseded in 4.0 are back. In general the conversion of programs and data should be straightforward and without troubles.

NOTE: This process assumes that you have created a new sub-directory separate from where TAS50 is installed. If you haven't done so yet then please do that now. The TAS50 sub-directory should only hold the programs provided by CAS. Through the use of the SET TAS50= and putting the TAS50 sub-directory in your path (both in your environment) you will be able to access the Pro 5.1 programs from any other drive or sub-directory on your computer.

Converting the 3.0 Data Dictionary

The first step in converting from 3.0 to 5.1 is to convert your data dictionary. This will bring all your old field names into the new version. The 5.1 data dictionary is considerably different than 3.0. It is now made up of 8 different files instead of 2. The new ones include two different key information files; a defined field dictionary; a file relationships file; a file that holds the Btrieve initialization information for each file so that files can be initialized at a user site without the full data dictionary; and a listing of printers available. As explained above you should create a new sub-directory for this conversion. For ease of explanation, we will assume that all programs are on drive C and that the new sub-directory you've created is C:\NEWPRGS and that the old sub-directory is C:\TAS30. For this conversion to work it doesn't matter what drive you're on, where TAS Pro 5.1 is, or what the name of the sub-directories are. Just insert your names for the ones set here. The process is as follows:

1) Create and log into the new sub-directory. Again, we're going to assume that you've created a sub-directory called NEWPRGS on drive C. When this is complete you should be at the DOS prompt in the new sub-directory:

```
C:\NEWPRGS>
```

2) Copy the data dictionary from the TAS Pro 5.1 sub-directory to C:\NEWPRGS. This would be (we will not be showing the DOS prompt anymore but unless instructed differently you should always be at DOS):

```
COPY C:\TAS50\FILE*.B
```

This will copy the 8 dictionary files into your new sub-directory.

3) Copy the OVL files:

```
COPY C:\TAS50\*.OVL
```

This will copy 2 files; TAS50.OVL and TASCOLOR.OVL.

4) Run the Set Configuration program by typing at the DOS prompt:

```
R50 C:\TAS50\TASCINFO
```

This should bring up the Set Configuration program as explained in **Chapter 3, Main Menu Programs**. Move the cursor to the Default Dictionary Path near the bottom of the screen. Do this by pressing the ENTER key until the cursor is in that field. If you haven't made any changes in the TAS50 sub-directory then this field should be blank until you reach it. Then your current sub-directory should be automatically filled in. Press the F10 key and answer Y to the save question. The program will exit and return you to the DOS prompt.

5) Now we're ready to convert the old dictionary. Type the following at the DOS prompt:

```
R50 C:\TAS50\CNVTDICT
```

The program will ask, at the top of the screen, for the location of TASFILE/TASDICT. At the bottom of the screen it should say Dict Path: C:\NEWPRGS\ (or whatever you named this sub-directory). IF IT DOESN'T STOP NOW AND RECHECK STEPS 1 THROUGH 4. Enter the location of the 3.0 dictionary, in this example you would enter:

```
C:\TAS30\
```

6) Next is will ask you for Schema name. We're going to assume that you want to convert the entire dictionary so you'll just press ENTER here and put no value in the field. However, if you want to convert just a single schema/FD you would enter the name here, or you can also use wildcards, i.e., BK*, or BKAR*, etc.

7) You will now be asked whether or convert from the old form BCD to new type fields. This will convert all your 3.0 N type fields to new 5.1 N type. If you answer N here they will be converted to 5.1 O type (old BCD) fields. In either case both T (time) and D (date) fields will be converted to the new format that is in keeping with what Btrieve expects for those type fields. The decision to convert your BCD fields hinges on two questions you have to ask yourself. First, if you are going to want to access your Btrieve data with other programs they cannot understand the BCD format. You will be forced to convert to the new floating point type. Second, if you are using overlays as keys and part of these overlays are N type fields when they are converted you will need to reset those as segmented keys instead of overlays. This is accomplished by using the Data Dictionary Manager explained in Chapter 3. This is again in keeping with what Btrieve, and other applications that access Btrieve, expect. If you aren't interested in accessing your data with other programs then you should answer N here. This will give you the most compatibility with the 3.0 version.

NOTE: In either case the program will convert your date and time fields to the new format. Internally they are still only 4 bytes, however, the data itself is different. This means that any overlay keys you have that include a date or time field will have to be converted to a segmented key. This is a relatively easy process. You would run the Data Dictionary Manager (see **Chapter 3, Main Menu Programs**), choose that schema (what we now call FD - file descriptor), press ^K (CTRL+K), and choose the offending key from the list. At the entry fields you will see the overlay as the only segment in the list. You will enter each field in the overlay in the order they appear, one to each segment. Save the key info, save the FD, reindex it and you're done.

8) The next step is to initialize the new files. Again, at the DOS prompt type:

```
R50 C:\TAS50\INITALL
```

It will ask for the files to be initialized. Just press the ENTER key for all. Next it will ask for location. At this time we'll assume that all the files will be in the current sub-directory so press ENTER again. As each file is initialized a message about this will be displayed on the screen. The process continues until all files are initialized. If for some reason a file you are creating using this method already exists in the current sub-directory you will be asked whether or not you wish to initialize it. Generally you would answer Y. If you have followed these instructions this shouldn't happen.

9) At this point you have created your new data dictionary and you can either convert the data now or the programs. Since we're already in the middle of the data we'll continue with that. However, should you want to convert programs now instead skip to the next step and then return here when you are ready for the data.

NOTE: Converting the data can take a considerable amount of time depending on the size of your files, speed of your computer, etc. We suggest you do this over a weekend or atleast overnight. The process is automatic once it starts and shouldn't require any further input on your part. Should something happen during the conversion, power failure, etc. and you wish to restart then delete all the new files (don't delete the data dictionary files), go back to step 8 and initialize again and then do this step again.

NOTE: This process changes your old data dictionary. You should make a backup of TASFILE.M and TASDICT.M so that should you need to make further changes to any 3.0 programs you will have a good data dictionary.

From the DOS prompt type the following:

```
R50 C:\TAS50\CNVTDATA
```

Again, the program will ask for the location of TASFILE/TASDICT. This should be the same path you entered in step 5. Next it will ask for the files to be converted. Again, press just the ENTER key for all files or put in the FD name as appropriate (you can use wildcards). As each file is converted the name of the file and the number of records remaining will count down to 0. The program will automatically go from one file to the next until all are converted.

NOTE: One of the biggest problems you can have here is if the old file doesn't exist. If that occurs the program will display an error message on the screen and wait for a response. If you aren't there the computer will patiently wait until you get back. This can be a problem if you did this overnight and expect all the files to be converted in the morning. So, try to be sure that there is a file that exists in the appropriate sub-directory for each you are going to convert, or, plan to check the computer several times during conversion.

Converting 3.0 Programs

10) We're now ready to convert the programs. There were two different options for source code in TAS Professional 3.0. They were .SRC or .EDT files. If you edited programs with the source code editor you have .EDT files. If you used a text editor then you probably have .SRC files, although, many people learned the codes for the .EDT format and, due to limitations of the editor, learned how to program by the numbers. In TAS Professional 5.1 there's only one format, .SRC. The editor can handle this format and it's the same file if you wish to use a text editor.

The conversion program doesn't care which you used, although, you can only convert one type at a time. To start the conversion program type:

```
R50 C:\TAS50\CNVTPALL
```

The first question it asks is what type of program you're converting. Since this is the 3.0 conversion you will enter either E (.EDT type) or S (.SRC type). Next enter the file path and name, wildcards are acceptable. So to convert all the BK*.EDT programs in the TAS30 sub-directory you would enter E for the program type and then the following for the file name(s):

```
C:\TAS30\BK*
```

The program will then ask if you wish to convert DEFINE fields that are type N from BCD to new form floating point. In this case you would answer N if you have DEFINEd overlays in which one or more of the fields is an old form BCD.

NOTE: ** VERY IMPORTANT ** This conversion routine converts your old programs line-for-line. However, if you have DEFINEd overlays you need to make sure, after conversion, that the entire block of DEFINEs is at the beginning of the program. This is due to the way the 5.1 compiler adds fields to the field list during compilation. You may think you've got a contiguous block of data in memory but in reality you may not. By putting all the defined fields in that block at the beginning of the program you can be assured, they will be contiguous in memory at runtime. This only applies to those DEFINEd fields that are also part of a DEFINEd overlay.

Once you enter the file name(s) the program will start the conversion process. During conversion the defined field are first located so that the screen/report formats can be converted properly. Next the lines are converted to the 5.1 .SRC format. As each line is converted the program displays the line number and the percentage completed. Due to differences in programs and the process required the percentage will probably never be exactly 100 upon completion and is meant to be an approximation only. This process is very fast and should take less than an hour. However, this is very dependent on the number of programs you're converting and the speed of your computer.

When this process is complete you should have a file in the new sub-directory with the same name as the program in the old. These will all have the extension .SRC.

11) The next step is to compile the converted programs. We have found 3 general problems in this conversion process and they show up here. They are:

A) Multi-defined fields. In TAS Professional 3.0 you could define a field in a program several times and never get an error. This caused many problems when running, especially if you defined the field as one type at the beginning and another further down in the program. The compiler would always use the last DEFINE command so you would think the field was alpha when it was actually numeric and might spend hours trying to figure it out. In 4.0 we solved this problem and gave a syntax error for multiple defines. In 5.1, with the push to convert 3.0 code, we have made this a warning error in the compiler, and the DEFINE command itself will be ignored. This means that only the first DEFINE will be effective. The warning error is displayed, printed or sent to the .ERR file as appropriate. You should make sure that none of these are going to effect your program. However, if these are all the errors you get, the program will compile successfully and you will be able to run the program. This happens alot in many programs.

B) Another problem that was not caught by the 3.0 compiler was extra parentheses in an expression. In 5.1 the expression compiler is much more sophisticated and catches those kinds of errors. There's no way to tell which ones are just conversion and which are actually bad so this does become a syntax error and needs to be fixed before you can

compile the program. This is generally not a big problem, and only happens occasionally.

C) The last is when your program accesses TASFILE or TASDICT fields in the dictionary. Since the data dictionary has changed completely so have the field names. If you have a program that does this you will have to go through it and change these references yourself. This should be very rare and shouldn't take you much time at all. We decided not to put this in the conversion routine due to the time it would take to check each and every field on the off-chance one of them was a TASFILE or TASDICT field. The corresponding field names for those old dictionary files are:

OLD	NEW	FILE
DICT.NAME	DICT_FIELD_NAME	FILEDICT
DICT.SCHEMA	DICT_BUFF_NAME	FILEDICT
DICT.SKEY	DICT_OVERLAY[1]	FILEDICT
DICT.OFFSET	DICT_OFFSET	FILEDICT
DICT.TYPE	DICT_TYPE	FILEDICT
DICT.SIZE	DICT_SIZE	FILEDICT
DICT.DEC	DICT_DEC	FILEDICT
DICT.UPCASE	DICT_UPCASE	FILEDICT
DICT.KEY	KEY_NAME[2]	FILEKEY
DICT.KEYN	KEY_NUM[2]	FILEKEY
DICT.OVLY	DICT_TYPE[3]	FILEDICT
DICT.SIZEH	Internal size is calculated at runtime.	
DICT.ARRAY	DICT_ARRAY_ELE	FILEDICT
DICT.DECH	DICT_HDEC	FILEDICT
DICT.SIZEHLD	DICT_HSIZE	FILEDICT
DICT.OFFH	DICT_HOFFSET	FILEDICT
BTLFNAME	LOC_FILE_NAME	FILELOC
BTLSCHEMA	LOC_BUFF_NAME	FILELOC
BLTMLOC	LOC_LOCATION	FILELOC

[1]This is actually no longer a field but just a key. If you try to set this equal to something you will get a field not found error during compilation. You need to set the DICT_FIELD_NAME and DICT_BUFF_NAME equal to their individual parts and then just use the key.

[2]With segmented keys the whole key structure has been separated from the fields. There really is no need anymore for key names to exist in the field structure so these are a little more difficult. You can either use these fields, which are 24 element arrays, or use the fields in FILEKNUM. A single record exists in this file for each key. The key name is in KNUM_KEY_NAME and the number is in KNUM_KEY_NUMB.

[3]Overlays are now just a separate type of V.

12) The conversion process is now complete. All that's left is to run the programs. If you desire you can slowly (or quickly) add the new features of TAS Professional 5.1 or you can just stick with the 3.0 code. In fact, we have converted the 3.0 editor/screen painter/report writer so that you can continue to edit program in 3.0 .EDT format. You will have to convert them to 5.1 before they can be compiled but that takes just a few moments, as you can see. The decision is yours.

Converting from TAS Professional 4.0 to 5.1

There are several new commands in 5.1 and some new features. These include larger source code sizes, virtually no limits on compiled segment sizes, full use of extended memory, and many more. We have also returned to the 3.0 method of combining screen and report formats in the source code. However, we've also given you the option of keeping some or all as separate files on the disk.

There are three commands that no longer exist. These are LOAD, CALL and RELEASE. All three dealt with .BIN files and there are better ways getting the same results. Either use the INT command or run an external program. With extended memory the programs would not have run and would have shut down your system. Also, we have eliminated the source code library management routines since we've made the other changes. There's no longer any need to limit your source files to 64k maximum nor keep all those different .SCP/.SRP/.SRB files. If you have source code in one of these maintenance files it will have to be exported before it can be converted to 5.1

NOTE: This process assumes that you have created a new sub-directory separate from where TAS50 is installed. If you haven't done so yet then please do that now. The TAS50 sub-directory should only hold the programs provided by CAS. Through the use of the SET TAS50= and putting the TAS50 sub-directory in your path (both in your environment) you will be able to access the Pro 5.1 programs from any other drive or sub-directory on your computer. This process assumes your TAS Professional 4.0 programs and data are located at C:\APP40\

The following are the steps to converting to TAS Professional 5.1:

1) Create and log into the new sub-directory. Again, we're going to assume that you've created a sub-directory called NEWPRGS on drive C. When this is complete you should be at the DOS prompt in the new sub-directory:

```
C:\NEWPRGS>
```

2) Copy the OVL files:

```
COPY C:\TAS50\*.OVL
```

This will copy 3 files; TAS.OVL, TAS50.OVL and TASCOLOR.OVL. You can delete or just ignore the TAS.OVL file. We have included it in the distribution set for the program TASEDP30.RUN only.

3) Run the Set Configuration program by typing at the DOS prompt:

```
R50 C:\TAS50\TASCINFO
```

This should bring up the Set Configuration program as explained in **Chapter 3, Main Menu Programs**. Move the cursor to the Default Dictionary Path near the bottom of the screen. Do this by pressing the ENTER key until the cursor is in that field. If you haven't made any changes in the TAS50 sub-directory then this field should be blank until you reach it. Then your current sub-directory should be automatically filled in. Press the F10 key and answer Y to the save question. The program will exit and return you to the DOS prompt.

4) Copy the data dictionary and files from the old sub-directory to the new:

```
COPY C:\APP40\*.B*
```

Your data and dictionaries are now converted!

Converting 4.0 Programs

NOTE: The conversion routine creates files with the same name and extension as those being converted. Because of this you will want to make sure that the 4.0 source files are in a different sub-directory from the sub-directory where you plan to put the converted files. If you don't you will end up with files that are 2 bytes long! The conversion routine doesn't change the original file so you can safely use the original source files in their original location and just make sure you're saving the converted files to a new sub-directory. This would already be the case if you followed the instructions at the beginning of this section.

5) We're now ready to convert the programs. To start the conversion program type:

```
R50 C:\TAS50\CNVTPALL
```

The first question it asks is what type of program you're converting. Since this is the 4.0 conversion you will enter 4. Next enter the file path and name, wildcards are acceptable. So to convert all the BK*.SRC programs in the APP40 sub-directory you would enter 4 for the program type and then the following for the file name(s):

```
C:\APP40\BK*
```

The program will then ask you if you wish to append any #INC (include) files that are referred to in the source code. It will also ask if you wish to add screen and report formats. If you answer Y here and the format is NOT found the MOUNT command in the source code will be changed to MOUNT fmt_name xxxx EXTERN. This signifies that the format for this particular MOUNT is external (or outside) of the source code. If you answer N here then the compiler directive #EXT_FMT will be added to the beginning of the program. This tells the compiler that all formats are external to the source code.

Once you enter the file name(s) the program will start the conversion process. The conversion process from 4.0 to 5.1 is actually very simple. If you don't include screen/report formats and you don't append #INC files all you need to do is copy all the *.S* files from the C:\APP40\ sub-directory and then add the #EXT_FMT compiler directive to the beginning of each main source file. **You also need to run the CHKZEROS program to make sure there are no lingering binary 0s in any of the source files. If you run the conversion routine it will do this for you.**

When this process is complete you should have a file in the new sub-directory with the same name as the program in the old. These will all have the extension .SRC. The conversion utility doesn't know whether or not the source code is part of another program or separate so all .SRC files will be converted and copied over. If you've appended the #INC files you can delete all those not necessary since they've been included already.

NOTE: This process does not effect library files. Those will still be separate.

6) The next step is to compile the converted programs. You should get no errors that didn't appear in the 4.0 programs. You may need to change the records in COMPINFO.B. We've found that just a single DEFAULT record will probably suffice since you can set the default values so high.

7) Your conversion from 4.0 to 5.1 is now complete. We hope you enjoy the new features and all that memory!

INTENTIONALLY BLANK

TAS
PROFESSIONAL
VERSION 5.1

Chapter 9

Compiler Errors

INTENTIONALLY BLANK

INTRODUCTION

All errors for the TAS Professional 5.1 compiler are maintained in the ERRMSG.B file. It is not recommended that errors be changed unless they are being translated into another language. The following is a list of all errors and a short explanation of each where needed. There are other messages within the ERRMSG.B file. These include help, and demonstration records. They can be ignored.

The compiler will, when applicable, display the offending portion of code first with the SYNTAX ERR message. Then immediately below will be displayed the full error message.

If the programmer looks at the message in the ERRMSG.B file s/he will notice that there are some extra characters that aren't displayed here. These are to force a cr/lf (\n) or to make sure that extra spaces are saved at the end of the line (\i). These extra characters are not displayed here since they are not of consequence to the message itself.

Error Num	Error message and explanation
500	`The source file above was not found. Make sure the file has the correct extension and that you have entered the drive and path correctly.` The default source extension is .SRC.
501	`The Screen/Report format was not found.` If the **EXTERN** option is set in the **MOUNT** command or if the **#EXT_FMT** compiler directive is set the compiler looks for the screen/report formats on the disk in the same sub-directory as the source code.
504	`An error has occured while reading the Screen/Report format file.` A DOS file error occured.
512	`The line is longer than 1024 characters, or the cr/lf pair was not found. This could also be caused by a binary 0 within the source line.` The maximum length of any source code line is 1024 characters.
513	`There has been an error discovered in the source file.` This is a DOS file error problem.
527	`Number of fields in program:` The compiler will return the number of named fields used in the program.
528	`Number of Line Labels:` The compiler will return the number of line labels named in the program. UDC and UDF names are treated as line labels.

529 Size of Code segment (bytes):

This is the run code buffer as explained in **Chapter 4, Compiler Information**.

530 Size of Constant segment (bytes):

This is the constant buffer as explained in **Chapter 4, Compiler Information**.

531 Size of Spec Code segment (bytes):

This is the spec line buffer as explained in **Chapter 4, Compiler Information**.

532 Size of DEFINE Data (bytes):

This is the total internal size of all fields created with the **DEFINE** command. The maximum size of defined data allowable is 262,140 bytes.

536 All file handles available from DOS were in use. Please make sure that the line FILES=20 (or more) is in your CONFIG.SYS file. If that isn't the case then add it, reboot your computer and try again.

These are files that are opened by the compiler for use during the compilation process.

537 Access has been denied to you for the file requested. This means either the file is marked read-only or that there is no space/directory entries available.

Make sure that the .RUN program you are creating is not set read-only.

538 The path entered does not exist.

539 The program is unable to create the file above. Unknown reason.

The programmer should make sure that a legal file name was entered.

540 There are no lines of code in the specified source file.

541 There was not enough space on the disk to save the run file. Please delete or move some files to other disks and try again.

542 The compiler was successful and the .RUN file has been created.

Everything went just fine and the .RUN program is ready to run.

545 Quote marks are unbalanced in the line.

You need to make sure that the same type of quote marks have been used at the beginning and end of the string constant. An easy error can occur when you use an apostrophe within the constant. For example:

'Don't do that again'

Will cause this error to appear. The correct way to use this constant is:

"Don't do that again"

547 An unknown command modifier was found.

An option has been specified which doesn't exist for this command.

548 An unknown compiler directive was found.

549 An error was made in specifying the date type. The only choices are: American/Ansi/British/Italian/French/German.

The above options are the date formats allowed. For more information on what each one looks like please see **Chapter 5, Command Reference**.

553 Too many decimal characters have been specified in a constant. The maximum is 8.

554 An ELSE command has been found without a corresponding IF.

An **ELSE** cannot be used unless an **IF** command has been used first and is still active. For more information please refer to **Chapter 7, Structured Programming Commands**.

555 An ENDIF command has been found without a corresponding IF.

An **ENDIF** cannot be used unless an **IF** command has been used first and is still active. For more information please refer to **Chapter 7, Structured Programming Commands**.

556 An ENDWHILE command has been found without a corresponding WHILE.

An **ENDW** (Endwhile) cannot be used unless a **WHILE** command has been used first and is still active. For more information please refer to **Chapter 7, Structured Programming Commands**.

557 An EXIT command has been found without a corresponding WHILE.

An **EXIT** cannot be used unless a **WHILE** command has been used first and is still active. For more information please refer to **Chapter 7, Structured Programming Commands**.

558 A NEXT command has been found without a corresponding FOR.

A **NEXT** cannot be used unless a **FOR** command has been used first and is still active. For more information please refer to **Chapter 7, Structured Programming Commands**.

559 A FLOOP command has been found without a corresponding FOR.

A **FLOOP** cannot be used unless a **FOR** command has been used first and is still active. For more information please refer to **Chapter 7, Structured Programming Commands**.

560 A FEXIT command has been found without a corresponding FOR.

A **FEXIT** cannot be used unless a **FOR** command has been used first and is still active. For more information please refer to **Chapter 7, Structured Programming Commands**.

561 A LOOP command has been found without a corresponding WHILE.

A **LOOP** cannot be used unless a **WHILE** command has been used first and is still active. For more information please refer to **Chapter 7, Structured Programming Commands**.

562 An ENDCASE cmd has been found without a corresponding SELECT.

An **ENDC** (Endcase) cannot be used unless a **SELECT** command has been used first and is still active. For more information please refer to **Chapter 7, Structured Programming Commands**.

563 There have been 1 or more IF cmds without corresponding ENDIF.

Each **IF** command must have a corresponding **ENDIF** to finish it off. For more information please refer to **Chapter 7, Structured Programming Commands**.

This can also occur if you have forgotten an **ENDC** (End Case) command at the end of a **SELECT/CASE** structure.

564 There have been 1 or more WHILE commands without corresponding ENDWHILE.

Each **WHILE** command must have a corresponding **ENDW** (Endwhile) to finish it off. For more information please refer to **Chapter 7, Structured Programming Commands**.

565 There have been 1 or more SELECT commands without corresponding ENDCASE.

Each **SELECT** command must have a corresponding **ENDC** (Endcase) to finish it off. For more information please refer to **Chapter 7, Structured Programming Commands**.

566 There have been 1 or more FOR commands without corresponding NEXT.

Each **FOR** command must have a corresponding **NEXT** to finish it off. For more information please refer to **Chapter 7, Structured Programming Commands**.

567 The expression is too complex. Either increase the number of internal fields, or split it into two operations.

569 The comparison type was not acceptable.

Please see **Chapter 1, Installation and General Information** for the comparison operators.

570 The parentheses are unbalanced in the expression.

Each expression in which parentheses are used must have an equal number of '(' and ')'. Make sure NOT to surround a single value with parentheses, i.e.:

(4) or (counter)

571 There is an error in your expression. Probably a single number/field with parentheses around it.

(1) or ('abc') or (test) are not legal expressions. These expressions can be stated without parentheses.

572 The function in the expression is not a legal TAS Professional 5.1 function.

If this is a UDF the programmer must be sure that the #UDF compiler directive has been set before the function is used in an expression.

574 You must have a receiving field in this command.

575 You must have a file name in this command.

580 You have specified too many #INC (include) files in the program. The maximum allowed is 20.

581 FILEDICT.B was not found.

Make sure that the TAS50= value is set. For more information please see **Chapter 1, Installation and General Information**.

582 FILELOC.B was not found.

Make sure that the TAS50= value is set. For more information please see **Chapter 1, Installation and General Information**.

583 FILEKNUM.B was not found.

Make sure that the TAS50= value is set. For more information please see **Chapter 1, Installation and General Information**.

585 No memory space is available to allocate the Source file buffer.

586 No memory space is available to allocate the Constant buffer.

587 No memory space is available to allocate the Field buffer

588 No memory space is available to allocate the Label buffer

589 No memory space is available to allocate the Compiled program buffer

590 No memory space is available to allocate the Specification line buffer

591 No memory space is available to allocate the Screen/Report source file buffer

592 No memory space is available to allocate the Compiled Screen/Report buffer

593 The maximum size for field name and array is 60 characters.

The field name cannot be more than 15 characters; however, the array element specifier can be an expression.

594 The option entered was not found.

595 The option entered is not allowed in this command.

596 *** Internal Error ***

Call support.

598 Error while opening error file.

599 The Constant segment has grown larger than the memory allocated. For more information please refer to the manual under Chapter 4, Compiler Information.

600 The Code segment has grown larger than the memory allocated. For more information please refer to the manual under Chapter 4, Compiler Information.

601 The Spec Code segment has grown larger than the memory allocated. For more information please refer to the manual under Chapter 4, Compiler Information.

602 You have too many different named fields. For more information please refer to the manual under Chapter 4, Compiler Information.

603 You have used too many different files.

A maximum of 32 files may be used in a program.

605 You have reached the maximum nesting level for the IF command.

The **IF/ENDIF** structure can be nested 20 deep. For more information please refer to **Chapter 7, Structured Programming Commands**.

606 You have reached the maximum nesting level for the WHILE command.

The **WHILE/ENDW** structure can be nested 20 deep. For more information please refer to **Chapter 7, Structured Programming Commands**.

607 You have reached the maximum nesting level for the SELECT/CASE command.

The **SELECT/CASE/ENDC** structure can be nested 20 deep. For more information please refer to **Chapter 7, Structured Programming Commands**.

608 You have reached the maximum nesting level for the FOR command.

The **FOR/NEXT** structure can be nested 20 deep. For more information please refer to **Chapter 7, Structured Programming Commands**.

609 ** WARNING ** The field you are trying to add has already been DEFINEd. You cannot define the same field twice. This DEFINE command will be ignored.

In TAS Professional 5.1 only the first **DEFINE** in the program will be effective. This is just a warning message and will not keep the program from running. The line number of the subsequent **DEFINE** will be displayed. You should check the program and make sure the field is being properly defined.

610 The maximum number of fields you may allocate is 2000.

611 You have tried to add more fields than you have allocated. For more information please refer to the manual under Chapter 4, Compiler Information.

612 The field name specified was too long. Maximum 15 chrs.

613 The field name is illegal. It must start with an alpha chr from A thru Z.

614 You have specified an unknown field type. The only acceptable values are: A / N / D / T / I / B / P / R / L / O

615 You must use a numeric constant for the field size.

616 You must use a numeric constant for the field decimal chrs size.

617 You must use a numeric constant for the field array size.

618 Something is wrong with the array specifier. Please check it.

619 The field display size specified is too long for the field type.

620 You have not specified a field display size.

621 You cannot specify more than 8 decimal chrs in a numeric field.

622 You have not specified a field name for this field.

624 This field was used in the program. It must be either DEFINEd or in the data dictionary.

625 You have defined a field that is also in the data dictionary.

This is just a warning error; however, the programmer must be sure that s/he isn't misusing the field.

626 You have tried to DEFINE too many fields at the same time. The limit is 10 for a single DEFINE command.

627 The #XLATE compiler directive has been set in this program and the file cannot be found in the default directory.

The TRANSLTE.B file must be in the same directory as TASCOMP.EXE and have the TAS50=path value set.

628 The key name was not found in FILEKNUM.B

Each key created for a file has a record in FILEKNUM.B. You can refer to this key in any command requiring it using the key number (preceded by the at sign '@') or the key name. If the compiler cannot find the key name used, this message will be displayed. If you add an index to a file make sure that you either initialize or reindex/restructure that file. You won't be able to use the key until you do.

629 The label name was used but was not set as a legal label.

This could also happen if you have used a UDF or UDC but the actual function or command is not a part of the program or has not been included. If you can't find the error remove the compiler directives for the UDF and/or UDC (or UDX). This will increase your errors but you will rapidly find the line giving you the problem.

630 The label name was already used.

UDF and UDC names are added to the label list. You must be careful not to duplicate those names with actual labels or to duplicate label names themselves.

631 The value in this instance must be a constant.

632 You can't specify a Pointer type constant.

633 You can't specify a Date type constant. Use =CTOD('date') instead.

If you are trying to set the value of a D type field you may also use the date as an alpha constant. For example:

'1/1/92'
'10/10/85'

634 You can't specify a Time type constant. Use =CTOT('time') instead.

If you are trying to set the value of a T type field you may also use the time as an alpha constant. For example:

'10:10'
'11:52'

635 An error has occured in Btrieve.

638 An error occurred while accessing the library file.

641 You may not use an expression or array field in a FIND

The name used in the **FIND** command as the **key_or_file_name** must be named explicitly as a key or be the name of a file.

642 You must have the FD in this command.

This is for the **SETACT** (Set Active) command.

643 There is a limit of 20 screen/report formats in a single program.

644 The Screen/Report Format above was not part of the source file but was referred t in a MOUNT command

If you don't have the **EXTERN** option set in the **MOUNT** command, or the **#EXT_FMT** compiler directive then the screen/report format must be a part of the source code. You can import a format by loading the program in the source code editor, pressing the ^I key (CTRL+I) and entering the path and name of the format file.

645 A label is required for this command.

646 You must have a format name in a MOUNT command.

647 The only legal format types are (S)creen, (R)eport and (B) Report/2

648 This error occurred during compilation of screen/report format:

The name of the screen/report format that is causing the problem will be displayed at the end of this message.

649 This error occurred while checking all Label locations.

This is the final check to make sure that all labels that were used in a program were accounted for.

650 The maximum number of nested functions is 10.

651 The screen/report format name used in the REMOUNT command hasn't been MOUNTed yet.

652 There are 1 or more SCANS without ENDS.

653 The maximum number of nested SCANs is 20.

654 SEXIT without SCAN.

655 SLOOP without SCAN.

656 ENDS without SCAN

657 You have exceeded the number of line labels specified for this program.

658 The value in the CASE command must be an integer numeric constant.

659 There is no START value for the FOR/NEXT loop and there must be.

660 There is no STOP value for this FOR/NEXT loop and there must be.

661 There is no STEP value for this FOR/NEXT loop and there must be.

663 The Defined Field Dictionary file (FILEDFLD) was not found.

This should be in the same subdirectory as the other dictionary files.

667 A Screen/Report format with this name has already been found. You can not have two formats with the same name.

The compiler found two or more different formats in the same source file with the same name. The first one was already compiled when it discovered the second.

668 The field used in the DUP option in the DEFINE command was not found in the data dictionary.

669 You have tried to initialize a Defined Filed before all the necessary info has been set. This includes Type, Size (if 'A') and Array Elements (if any). Also you can't INIT a BCD field.

The **INIT** option in the **DEFINE** command must be the last option set in the command.

TAS
PROFESSIONAL
VERSION 5.1

Chapter 10

Runtime Errors

INTENTIONALLY BLANK

INTRODUCTION

All errors for TAS Professional 5.1 are maintained in the ERRMSG.B file. It is not recommended that errors be changed unless they are being translated into another language. The following is a list of all errors and a short explanation of each. There are other messages within the ERRMSG.B file. These include help, and demonstration records. They can be ignored.

NOTE: If an error occurs during the operation of a program and you aren't sure where it took place you can recompile the program with the -D switch (debug). This will keep the line numbers and source code file names in the run program. When the error occurs again the actual line number in the source code file and the appropriate source code file name will be displayed at the top of the message box. This does add approximately 5 bytes per line to the size of the run file. It will also cause all other messages that use the same routine (**ASK** and **MESSAGE** commands) to display in the same manner.

Error Num	Error message and explanation
1	`TAS50.OVL not found or error while loading.`
3	`BTRIEVE has not been loaded.` For some reason BTRIEVE was not loaded properly. Make sure it is on your disk, preferably in your TAS50 subdirectory. You can also try loading it with the LB.BAT program. If it still doesn't load call support.
6	`The RUN program specified was not found.` The program name that was included on the command line could not be found. Don't include the .RUN extension when typing in the program name. If you have not included a program name, TAS Professional 5.1 looks for MENU.RUN. If it doesn't exist you will also get this message. You must either name a run program at the DOS command line or have MENU.RUN in your subdirectory. Also, make sure the programs are not set as read-only.
8	`An error occured while reading the RUN program.` Something has corrupted the .RUN program or it is from a previous version. Make sure it is up to date.
9	`No memory space is available.` The user needs to add more RAM memory.
10	`The temporary data space is too small to accommodate the calculations desired. Please specify larger in the source program.` Set the TDATA compiler directive to something over 64k bytes or more if necessary. You might also want to look at simplifying the expression that gave you this error.
16	`TAS Professional 5.1 internal stack overflow. Please try again with a larger stack.` This occurs when the program has executed too many nested **GOSUB**s, **PUSH**s, **UDC**s, or **UDF**s. The programmer needs to increase the internal stack value. Please see **Chapter 1, Installation and General Information** for instructions.

17 `TAS Professional 5.1 internal stack underflow. Too many POPs or RETURNs.`

The program has executed more **POP**s or **RETURN**s than **PUSH**s or **GOSUB**s.

18 `There are no more temporary fields available. Please increase initial value and try again.`

Increase the number of temporary fields using the -I compiler command line option. Default number is 30. You probably have a UDF that isn't returning properly. You might also try simplifying any expressions (split them over multiple lines if necessary).

19 `The field you are trying to use is unallocated. It may belong to a file that hasn't been opened yet, or was an array that was removed from memory - fieldname`

Make sure that the file this field belongs to has been opened. This generally happens when the programmer mounts a screen before the applicable files have been opened. The field name will be displayed at the end of the error message.

20 `The window title must be an ALPHA field.`

Can occur during the **WINDOW** command.

21 `You must have a SAVE_TO field for this command.`

Can occur during the **WINDEF** (Window Define) command.

22 `The SAVE_TO field you have specified is too small for its purpose.`

The window and attached information as defined is too large to fit in the field you have specified.

23 `This file number field must be type I or R.`

All **file_numbers** must be integers (I or R type).

24 `This field must be of an Integer type.`

Displayed in situations where you should've used an I type field.

25 `This field must be of a Record type.`

Displayed in situations where you should've used an R type field.

26 `The field in this command must be of an Alphanumeric type.`

Displayed in situations where you should've used an A type field.

27 `The field in this command must be a regular field, expressions or constants are not allowed.`

28 The field must be of type 'F' or 'P'.

You have used the redirection symbol (&) on a field that wasn't a pointer.

29 The receiving field in this command must be of type 'F' or 'P'.

The program expected a pointer as the receiving field.

30 The receiving field in this command must be of type 'F'.

In this situation the pointer must be of type F.

31 The array element specified is out of range for this field.

You have tried to access an array element that is beyond the number defined for the field.

32 File names, including paths, cannot be more than 59 chrs in length - @

39 Internal TAS Professional 5.1 error.

Call for support.

40 The VALID expression returned false. The entry must be done again.

You have specified a **valid** option in the **ENTER** command but no **valid_msg** option was set. This error message is the default.

41 The field to be entered was not in the screen buffer. Please make sure you have 'MOUNTED' a screen and that the field is on that screen.

The program has tried to **ENTER** a field for which no **column,row** option was set and the field is not in the active screen buffer.

42 An error has occured while accessing a non-TAS file - @

A DOS error occured during reading or writing.

43 There are no more extra fields available.

Too many fields have been added thru the **ADD** command. Use the **ADD_FLDS** compiler directive to provide more slots.

44 The field you are trying to add has an unknown type.

See **Chapter 1, Installation and General Information**, section Field Types for allowable types.

45 The field being added cannot have a size of 0.

A size must be specified.

46 The maximum number of decimal chrs for the field being added is 8.

47 An error has occured while attempting to open FILELOC.

This will occur when TASPRO is first executed. Make sure that FILELOC.B is present in the same directory as TASPRO.EXE and that the DOS SET value for TAS50= is correct, if applicable.

48 A record was not found in FILELOC for file - @1 . There must be a record for each file opened.

When a file is opened you need to either specify the **fd** and **path** values in the **OPENV** command, or a record for that file must be in the FILELOC.B file.

51 An error has occured while trying to open file - @1

The file either isn't where it is supposed to be or something has happened to it. The file name will be displayed as part of the error message.

52 The OWNER command will work with TAS Professional 5.1 files only.

You have set the **owner** option in the **OPENV** (Open Variable) command for a non-TAS file.

53 You have specified too many RELATEDs. The limit is 32 for all programs currently being run.

54 You have specified too many menu choices, max is 25.

55 You have specified a key number for which there is no key.

Check the keys for the file in the Maintain Data Dictionary program (TASDMGR.RUN). If you add a key to the FD you must either initialize the file or reindex/restructure it before the key can be used.

56 You have not specified either a SEARCH FILE value, nor a file_number this particular command. You must do one or the other.

57 You have not specified either a SEARCH FILE key number, nor a key value in this particular command. You must have one or the other.

58 You must have a legal file_number for this command.

The value being passed as the **file_number** is incorrect.

59 You can't delete records in non-TAS files - @

Records can be deleted only in TAS Professional 5.1 files.

60 The RAP buffer number specified was either 0 or more than the max number of buffers available.

The maximum number of RAP buffers is 10.

61 This file was not opened in a previous program - @1

You have used the **ROPEN** (Reopen file) command and the file wasn't opened in a previous program.

62 The file being used has not been opened yet.

65 In this command you must specify the NUMBER of elements.

66 You have used an illegal type spec in the REDEFINE command.

67 An Alpha field must have a size specified in the REDEFINE command.

68 You have specified a handle number and there is no corresponding file.

69 The program does not allow adding new items to this list, however, there are no lines to list at this time.

This will occur in a LISTM command when there are no lines to list and **NOADD** option is set.

70 You are trying to change this list and there are not enough unused lines to do so.

The program needs at least one extra line to use as a holder during any change, add or insert operation in a **LISTM** command.

72 You can't convert a TIME field to an INTEGER field.

73 You can't convert a DATE field to an INTEGER field.

74 You can't convert a POINTER field to an INTEGER field.

75 You can't convert a TIME field to a BYTE field.

76 You can't convert a DATE field to a BYTE field.

77 You can't convert a POINTER field to a BYTE field.

78 Internal error CRF

Call for support and give error number and CRF code.

79 An error has occured in BTRIEVE.

The error occured while trying to access a file in some manner.

82 BTRIEVE error - @

This will return the file name and the error number.

105 *** BAD DATE - REINPUT ***

The date entered by the user was incorrect. It will have to be reentered.

111 You must have a Buffer name for every non-TAS file - @

The **BUFFER** option in the **OPENV** (Open Variable) command was not set and the file was not defined in the File Data Dictionary.

112 You must specify a record size of every non-TAS file.

The **SIZE** option in the **OPENV** (Open Variable) command was not set and the file was not defined in the File Data Dictionary.

113 The maximum number of elements in an array to be sorted must be at least 1 greater than the number of elements to be sorted.

The program needs an extra element to use as temporary storage.

114 The field in the FFLD() function was not found in the field list.

116 The field in this command/expression must be of 'N' type.

117 You must MOUNT a Report Format before you can use the PRINT FORMAT command.

118 The Format number specified in the PRINT FORMAT LINE command is beyond the end of the Report Format that is currently mounted.

119 The major field in an UPDATE ARRAY command must be in array format, i.e., X[Y], even if it is the first element, or X[1].

120 You have tried to use a field in an UPDATE ARRAY ADD/SUB command that isn't of type N, I, R, or P. Those are the only ones allowed.

121 The only math allowed on ptrs is add and subtract.

123 An error has occured while trying to open the print to file. Please try with a different file name: -

Occurs during the **PON** (Print On) command when specifying a Disk file name.

125 The READ/WRITE commands may not be used with TAS Pro 5.1 files - @

These commands were meant for non-TAS files only.

126 An error occured while trying to set the position as required in a READ/WRITE command - @

The position specified was beyond the end of the file.

128 The program has tried to open more files than are allowed.

You may have 100 files open at any one time in all programs currently active.

129 An error has occured while opening or reading the printer driver file.

This file contains the printer control codes and should be in the TAS50 path. This can also occur if you don't have enough files specified in your CONFIG.SYS file.

130 The printer control command was not found.

You have specified a code name in the **PCHR** (Print Chr) command that could not be found in the active printer driver file.

131 The pointer field in this command must be of 'P' type.

135 An error has occured while creating file - @

This message could occur while creating a new file or initializing a current one in the Initialize file program (TASINIT.RUN).

136 **** BAD TIME - REINPUT ****

The time value entered by the user was incorrect. It will have to be reentered.

137 There is no space available in the Screen Field Buffer and another field cannot be added with the SAY command.

You should increase the number of extra screen fields in the buffer using the SFLDS compiler directive.

143 You can't export to or import from a TAS Professional 5.1 file. You must use standard OPEN/FIND/SAVE commands.

144 The program cannot find the import file specified in the command.

Make sure you have the proper path and file name. Don't forget: you need to include the applicable file extension, if any.

145 An error has occurred while attempting to open or write to the Export file.

146 The maximum size for an IMPORT/EXPORT record is 64k and you have exceeded that size.

147 An error has occurred while trying to read a record from the Import file.

148 The field being passed and the associated receiving field in the PARAMETER command must be of the same type.

154 An error occurred while opening or reading the ACS file as specified in the TAS50.OVL file.

155 The field provided is not large enough to hold the entire TRAP list.

This occurs during an **XTRAP** (Trap Operations) command when the field specified to hold the traps is not big enough. You need 1000 bytes to hold the entire trap list. We recommend that you don't specify a field at all. When the field is not specified the program will create a buffer for the TRAP list and will delete automatically when you **RSTR**.

156 The search and array field types in the ALOC function must be of the same type.

You have tried to search for an element in an array field using a value of a different type than that of the array field itself.

157 The second field in the ALOC function must be an array field.

200 Error during disk read/write. Btrieve - @

201 Must open file before accessing it. Btrieve - @

202 Tried to save a duplicate key when not allowed. Btrieve - @

203 Changed keys and tried to do Next, Previous, etc. Btrieve - @

204 Tried to modify a key value set as non-modifiable. Btrieve - @

205 An error has occurred while trying to access/create/open the pre-image file. Btrieve - @

206 The disk is full. Btrieve - @

207 A major (unrecoverable) error has occurred. Use your backups. Btrieve - @

208 The file you have tried to open is not a standard TAS Professional 5.1 (Btrieve) file. Please set the TYPE in the Open command. Btrieve - @

209 Need to add the /T to the Btrieve trailer before the Transactions commands can be used. Please refer to **Chapter 3, Main Menu - Utilties - Set Configuration**.

Change the value in the TAS50.OVL file for the Btrieve Load String. Add ‘ /T’ just past the last character in the current load string.

210 You cannot nest Transactions. One has to be COMMITed before another can BEGIN.

211 An error occurred during the Transaction process. You will need to restore the appropriate files from your backups. The disk is probably full. Btrieve - @

212 You must have a Transaction Begin before a Transaction Rollback. Btrieve - @

213 The maximum number of files that may be included in a Transaction update is 12. You have exceeded that number. Btrieve - @

214 File is read-only, you cannot write or delete any records. Btrieve - @

215 The number of buffers available has exceeded the number of files opened. Please reset the /M option. For more information please see **Chapter 3, Main Menu - Utilities - Set Configuration**.

Increase the value of the number after the /M in the Btrieve load string.

216 The Owner is already set for the file. Btrieve - @

217 The correct Owner name has not been provided for this file and it cannot be opened. Btrieve - @

218 An error has been received by Btrieve from the Expanded Memory Manager. Btrieve - @

This is a common error with some expanded memory managers. You need to add a /E to the end of the Btrieve load string. This will keep Btrieve from using expanded memory. You can also use 386Max™ from Qualitas. They know what the problem is and have worked around it.

219 This record has been changed by another user since you read it. It cannot be written back until you read it again. Another option is to lock the record when it is read. Btrieve - @

In this situation you have read the record with a **FIND** or **RCN** (Record Number Get/Set) command and the **no_lock** option was set. You then tried to write the record back to the file. It is best, in this type of situation where you know you're going to be saving the record back, to avoid using the **no_lock** option.

220 The lock table is full. Reset the /L option in the Btrieve load string. Please refer to **Chapter 3, Main Menu - Utilities - Set Configuration** for more information.

Increase the value of the number after the /L in the Btrieve load string. Or, if it doesn't exist, add it.

221 The record you have tried to access has been deleted previously by another user. If you use the locking process this shouldn't occur. Btrieve - @

This is the same situation as 219 in that you read the record with **no_lock**.

222 All records that are updated/deleted within a Transaction must be read during that transaction and not before. Btrieve - @

You can't read a record before a **TRANSX begin** command and then try to delete it before the **TRANSX commit**.

223 The number of files you have tried to open is greater than the FILES= parameter in CONFIG.SYS. It needs to be increased. Btrieve - @

225 DOS is restricting access to this file/record. Btrieve - @

226 This is not a TAS Professional 5.1 Program.

You have tried to **CHAIN/RUN** a program that was not compiled with TAS Professional 5.1.

227 This field is not indexed and cannot be used for searching within a file.

The user tried to use the search keys while the cursor was in a field that is not also named as an index in the file definition. If you have named the field as an index make sure that the file has been initialized or reindexed/restructured.

228 The file was not found in the list of opened files.

229 You have exceeded the maximum number of lines that can be wrapped, this is 2000.

230 The maximum number of WRAP'd fields in any one print line is 10.

231 You have specified a buffer for a non-TAS Btrieve file that is smaller than that specified by the file itself.

232 The maximum ENTER field size is 255 characters.

233 An error has occurred while attempting to compile an expression.

This will occur while trying to compile an expression as a filter. Make sure all fields in the expression are named in the program and that you haven't used a UDF, a function that doesn't exist, or an operator that doesn't exist.

234 You have used too many internal screen buffers in the program.

The maximum is 20. This means you may have up to 20 different **MOUNT** format commands.

235 You have tried to REDISPLAY a screen from an internal buffer and it wasn't previously saved.

You have a **REDSP** command without the corresponding previous **SAVES** command.

237 You have tried to load a program that cannot fit in the space allocated for it. Please allocate a larger space for the General Buffer in TAS50.OVL and try again.

Please refer to **Chapter 3, Main Menu - Utilities - Set Configuration**.

238 The Thru value in the PRINT FORMAT command is less than the last From value.

239 The printer is not operating. Output will default to the screen.

The program tried to access the printer and couldn't. It will send the report to the screen. If you wish, press the ESC key and the program will quit.

240 The printer number must be from 1 thru 3, program will default to 1.

This corresponds to LPT1, LPT2 or LPT3.

241 You are trying to close a file that was opened in a previous program.

Only the program that opens a file may close it. If the program doesn't close the file it is done automatically when the program quits.

242 Only F type pointers are allowed in the FLIST option.

243 You have specified a file number value that is not legal.

261 You have specified an array field for the NMENU command and the number of choices is set to 0.

When you use the array option in the **NMENU** command you must also tell the program how many choices there are with the **NCH** option.

262 An error has occured while trying to output to the printer. Do you want to try again? Y

This is the message that will appear initially if you try to print to the printer and something goes wrong. If you answer N to this question output will go to the screen.

263 You are trying to open a file without specifying the record size. You have probably specified an FD name. In this case you must also provide the record size.

264 You have tried to Execute a program and did not supply a name.

265 The record in file: xxx is locked by another user.

267 The number of active elements is greater than or equal to the number of maximum elements. In this case you can't add or change a line.

This occurs in a **LISTM** command when the number of elements that have values (**ACTV**) is equal to the maximum number of elements (**MAXA**).

273 Extra Memory Area numbers must be 1-4 only.

274 You have specified an Extra Memory Area number of 0.

275 The action you desire would exceed the size of the Extra Memory buffer area.

If you need to increase the size of the Extra Memory buffer areas see **Chapter 3, Main Menu - Utilities - Set Configuration.**

276 You may set the the special file number to 1, 2 or 3 only. You have tried to set a value outside that range.

This would occur in the **SSPCF** (Set Special File Number) command. I must be from 1 through 3.

277 The receiving offset field you have specified is not large enough. It must be of type R or, if type O the display size must be 10 chrs or more.

In TAS Professional 5.1 all memory offsets can be up to 10 characters in size due to the 32bit flat access. You need to make sure the receiving field is large enough to hold this value.

278 You have used too many internal trap buffers in the program.

You can nest up to 10 **XTRAP SAVE** commands where you haven't specified a **FLD** value.

279 You have tried to restore the TRAP table from an internal buffer and it was not previously saved.

This would occur when the program encounters a **XTRAP RSTR** command where the **FLD** value is not specified and there are no buffers to restore.

TAS
PROFESSIONAL
VERSION 5.1

Chapter 11

Windows Programming

INTENTIONALLY BLANK

Introduction

Welcome to the world of Windows programming. Many of you reading this will have been using Windows (and possibly even programming in Windows) for many years. For some of you this is your first attempt. Our aim with TAS Professional 5 for Windows is not to create the most complete programming language for Windows. It is to allow you to ability to keep one foot in DOS and the other in Windows as your customers make a slow, but methodical, transition to the Windows world. You will still be able to program in TAS for DOS and use that same code, with minor modifications, for your Windows customers. Not only that, but you'll be able to run some users on DOS and others in Windows at the same site, using the same programs and data! In fact, you'll even be able to run Windows and DOS versions of the **same** program for a single user at the same time.

As with our aim to create a first step into Windows with our language, this chapter is not meant to be the complete and encyclopedic information in actual Windows programming, but the information you need about the enhancements and differences between the DOS and Windows version of TAS Professional 5.

One major point to remember is that we have made this version of TAS Pro for Windows 16bit instead of 32bit (the DOS version). This allows you to run on both Win 3.1x and Win95. If we had created a 32bit version you would have been restricted to Win95/WinNT only. The important issue about this is that the maximum field size is 64k (65535) characters and the maximum number of array elements for any one field is 64k. What this means to you is that you can't run the development system in Windows since some of the fields are 500k+. So, you develop in DOS and run in Windows. If you have Win95 (and a 15+ inch monitor) you can put a DOS box on the screen in a small window and move easily between the development system and the Windows program. Since you can have both DOS and Windows active at the same time you don't even have to exit out of the development system when running your program under Windows. The limitations in existing commands and functions, as well as the new commands and functions, are listed below and in **Chapter 5 - Command Reference**, and **Chapter 6 - Function Reference**.

TAS Professional for Windows can run very well on a 4mb machine running Windows 3.1x. It also runs fine on a Windows 95 but has not been tested thoroughly on Windows NT.

We hope that this program works as well for you as it has for us. Now on to the rest of the information.

The Programs That Make Up TAS Professional 5 for Windows

1) TP5WIN.EXE - This is the actual Windows executable that will run your TAS Professional for Windows programs. This program does not need to be in the sam sub-directory as the programs you are running, however, you should be sure it's in a sub-directory specified in your PATH option in your AUTOEXEC.BAT file. For example, this could be your TAS50 sub-directory. If the TP5WIN.EXE file is not in the same sub-directory as your .RUN programs then you can either specify the initial run program in your command line or in the TP5WIN.INI file. In either case be sure to use the entire path or TAS won't be able to find the correct .RUN program. The path for the initial run program is also used for the default location of TAS50.OVL and it's where TAS will look for TP5WIN.INI first.

2) TPC50.EXE - This is the standard TAS Professional 5 for DOS programs. Make sure you're running version 5.1 or newer. Any version prior to this will not have the Windows extensions as part of the language.

3) TP5WIN.INI - More about this later, however, this file needs to be in your default sub-directory (as described above) or in your \WINDOWS sub-directory. There is a new program TASINI.RUN that will maintain this file.

4) WBTRTAS5.DLL - This is the Windows equivalent of BTRIEVE.EXE. This must be in your \WINDOWS sub-directory. An entry needs to be made in your WIN.INI file for the appropriate options for this program. This entry should look like the following:

```
[BTRIEVE]
options=/m:64 /p:4096 /b:16 /l:40 /n:12 /f:150
```

This entry can be made by the installation program or you can make it yourself. If you get error #24 while opening a file you more than likely have a problem in this. You should also make sure you don't have multiple Btrieve sections in your WIN.INI file if you enter it yourself. For more information on installing TAS Professional for Windows please refer to the general installation information in the front of this manual.

Now let's look at the changes between DOS and Windows.

Changes, Enhancements and Limitations

The most important limitation is that this is a 16bit product. This returns us to the 64k limits set for field and array sizes. Also, since we have not included the compiler in this version you cannot use the **FILTER** command or any alpha macro fields, i.e.:

```
AlphaField = 'X * 2'
Y = &AlphaField
```

If you include the above in a program you will get an error during runtime. The **FILTER** command is just ignored (as are all other commands and functions that are not part of Windows) and does not give an error. The **FOR** and **WHILE** options (used in many commands) are still availabled and work just like you would expect them to. This restriction on filters applies just to expressions that would be compiled 'on the fly' as is shown above.

Some commands and functions are not applicable to Windows and are not implemented. This can be for various reasons but mostly because they don't have any use. All the color commands in DOS are significantly different in Windows (see section on Color in Windows below). So, many of the standard DOS color commands are just ignored. All of the commands and functions that have not been implemented either have new forms in Windows or were not used much, if at all, in DOS. Following is a list of commands and functions that are not implemented in Windows at all, changed, or new to the Windows version:

Commands Not Implemented
AUTORUN, BKG, COLOR (colors are much different in Windows), COMPRG, DISPMEM, FRG, INT, LIST (not to be confused with LISTF and LISTM), MEMPTR, MEMSPC, MOUSE (it's always on), PAINT, PEEK, POKE, PORT, PRTNUM, REV, TRACE.

Commands Changed or Enhanced
CHAIN, CURSOR, ENTER, LISTF, LISTM, MSG, NMENU, SOUND, UPAR, WINDEF, WINDOW.

Commands New for Windows
{} (Process controls - These also work in DOS), BUTTON, CAPTION, CLIK_SRCH_LIMIT , GRAY, HOT_SPOT,PICTURE, ROW_COLOR,WIN_COLOR.

Functions Not Implemented
ATAN2, CC, CCE, CCF, CCR

Functions New for Windows
CLICKED_ON, COPY_FILE, EDIT, GET_ELEM_NUM, GET_WIN_COLOR, MAKE_DIR, MAX_COLS, MAX_ROWS, PRINT_CANCEL, PRINTER_NAME, WIN_LASER_PRT, WINDOW_PTR, WINDOWS

Details about changes, and new commands/functions, are in **Chapter 5 - Command Reference** and **Chapter 6 - Function Reference**.

In DOS you could not 'click' on a field that was part of an array, in Windows you can. You can use the 'standard' Windows editing commands in any field. This includes selecting a group of characters with the mouse (holding down the left button and dragging the mouse pointer over the characters), ^X (ctrl+X) to delete those chosen characters, ^C (ctrl+C) to copy those characters, or put the mouse pointer at a certain location and press ^V (ctrl+V) to insert characters copied previously either in a TAS program or some other that were saved to the clipboard.

The Windows version allows for multiple instances. This means you can bring up TAS Professional for Windows once and then do it again, and again, etc. until you run out of memory. If you attempt to access the same record in the same file in two different instances the program will act as though the record were locked by another user (just what it should do).

One of the major changes in Windows is a window itself. The next section explains the differences between the DOS and Windows screens.

Windows in Windows

Obviously, Windows is a much different beast than DOS. However, the most important difference is how it keeps track of what's on your computer screen. Probably the easiest way to explain this is that in DOS you have a flat screen. What we mean by this is that only one thing can be on the screen at a time. When you create a window and display it the characters under that window are gone forever, unless you have previously saved the screen with the SAVES command. But what you're doing, with the SAVES command, is putting each character and color attribute on the screen into a buffer. In DOS there are 2000 characters and color attributes in a 25 row by 80 column screen. In Windows, things are measured by the pixel instead of a character and in TAS it takes 128 pixels to make up a character. And by the way, that's just the text. Each pixel can have its own color attribute also, each color attribute is an R type field (internally it's a longint if you know what that is), so you can see that the number of bytes to hold the screen starts becoming very large. Plus, it would take forever, to constantly move those bytes from buffer to screen, to buffer, etc., even on fast computers. Luckily for us, Windows takes care of most of the work. So, instead of having a flat screen, we now have things on the screen (I hate to call them objects). And, even though you can't see it, they are actually piled on top of each other. So, when you add a new window it doesn't obliterate the characters underneath it, it just lays on top of them. What we've done here, it to try to make it look as much like a DOS screen as possible without losing the Windows touch and feel, thereby making your job as easy as possible.

There are some things that are much different, however. When you first run a TAS program, we create what we call a base form. The size comes from values you set in the TP5WIN.INI file (more about this later). No window you create can be larger than that base form. Each time you run a program TAS creates a new base window that is the same size as the client area of the base form (unless you use the **NOBASEWIND** option in the **CHAIN** command). All the windows, entry fields, etc. you create in that program are then stacked on top of the base window for that program. Each window you create is a separate entity until you get rid of it, either by exiting the program or using the **REDSP** command. In fact, the **SAVES** and **REDSP** command work much differently in Windows than in DOS. In DOS when you run the **SAVES** command it will save the current screen characters and attributes either to an internal buffer or to a buffer you supply in the command. In Windows all it does is put a bookmark in the list of windows that have been created. When you use the **REDSP** command in DOS it puts whatever is in the appropriate buffer back on the screen. In Windows it removes all the windows currently on the screen that have been created since the corresponding **SAVES**. You can see in just this one example how different the two systems are. There are many benefits to the process in Windows.

One of the most important is the ability to get a handle for any window currently on the screen. All you have to do is use the **WINDOW_PTR** function to get the handle or pointer for the currently active window. Then, when you want to make that window active again, later in the program, you just use the **WINACT** command and pass the value received earlier. This assumes that you have not removed the window previously with a **REDSP** command. Once it has been removed it cannot be called back without using the **WINDOW** or **WINDEF** commands.

Another thing that works differently in Windows is shadows. In DOS you can put a shadow to the right or left of the current window (this also works for menus, etc.). In Windows we actually create a raised border around the window if you specify a shadow of either right or left. This again, gives your program more of a Windows look and feel.

One command that has changed significantly in Windows is the **RSCR** (Reset the Screen) command. In DOS this command will undo the effect of any windows without removing those windows from the screen. In Windows this command removes all windows (except the base window) and may have unexpected results. If you find that you are losing windows on the screen when running a program, look for an **RSCR** command. In DOS you could alway reactivate a previously defined window after this command, in Windows you can't.

Hopefully, this will give you some insight into how we run under Windows. The next subject is logically connected, and that's about color.

Color in Windows

Probably, the thing you hear the most from new computer users is that Windows is much prettier than DOS. It certainly has more colors! Of course, in most systems today the typical user never even sees DOS so they might not even know what it used to be like.

In DOS colors are very easy, but also very limited. In Windows things become much more complex. In DOS a single value determines both the background and text color. In Windows there are two different values. Further, the values in DOS are a 1 byte value with a range of 0 through 255. In Windows this becomes a 4 byte value with a range of 0 through 4,294,976,295. Most computers don't allow close to this many options but many today allow more than 256.

In Windows a color value is split into 4 different pieces or bytes. Three bytes cover the colors red (R), green (G) and (B) which we all remember as primary colors from our high school science classes. The fourth byte determines the palette which we won't get into here. For right now, the palette byte (or the highest byte) will always be 0. The RGB colors can range from 0 to 255 each. The higher the color value the more intense the color becomes, so a blue value of 255 would be the most intense blue possible and 0 would be no blue at all. If all three color values are 0 you have the color black (no color at all) and if all three are at full intensity you would have the color value of 16,777,215 or white. To determine the color value use the following equation:

```
Color value = (BLUE*65536)+(GREEN*256)+RED
```

Or, you can use the standard colors provided by Windows. These are the colors you (or your system) have determined as appropriate for the different parts of the screen, or standard colors. You can also use colors that have been setup in the TP5WIN.INI file. You should note that you can't use these colors directly, you must use them as the parameter in the **GET_WIN_COLOR** function. Also, due to the major difference in colors between DOS and Windows all commands that would normally allow you to set specific colors ignore any values unless you use the **USE_COLORS** option that is a part of that command. This applies to **NMENU**, **LISTM, LISTF, WINDOW** and **WINDEF**. If you do not use the **USE_COLORS** option these commands will get the appropriate colors from the TP5WIN.INI file or from the defaults for those colors (see below). A typical command might be:

```
Window at .... wcolor get_win_color('red') bcolor
       get_win_color('blue') ...
```

The following are the different Windows color names:

Standard Windows Part Names (these colors are set by the system)
ScrollBar, Background, ActiveCaption, InactiveCaption, Menu, Window, WindowFrame, MenuText, WindowText, CaptionText, ActiveBorder, InactiveBorder, AppWorkSpace, Highlight, HighlightText, BtnFace, BtnShadow, GrayText, BtnText, InactiveCaptionText, BtnHighlight.

Standard Windows Color Names
Black, Maroon, Green, Olive, Navy, Purple, Teal, Gray, Silver, Red, Lime, Yellow, Blue, Fuchsia, Aqua, LtGray, DkGray, White.

Color Names in TP5WIN.INI
NormalBkg, NormalText, BoxBkg, BoxText, EnterBkg, EnterText, MsgBkg, MsgText, WindowBkg, WindowTextSet, MenuBkg, MenuTextSet, ButtonBkg, ButtonText, ChoiceBkg, ChoiceText, EcolorBkg, EcolorText.

Another process that can be significantly different in Windows is printing. Let's look at that now.

Printing in Windows

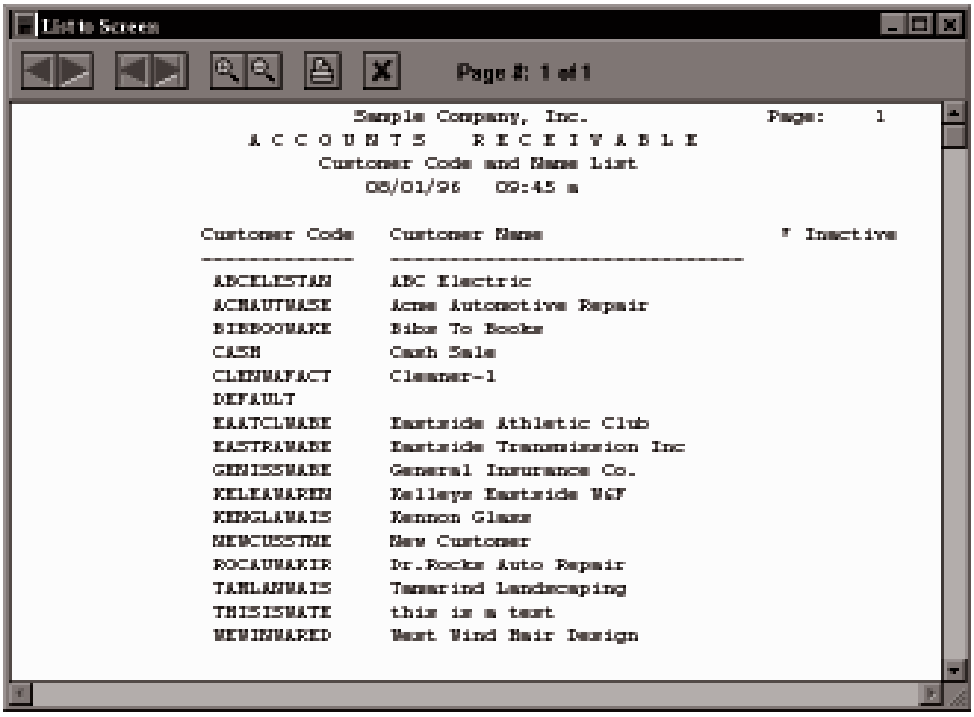
For those of you who have had the joys of trying to print under Windows you may know that it's not quite as simple as they would like you to believe. Also, no matter how hard you try, your output is never going to match up to what you had under DOS. So, we here at Business Tools, trying to always give you the best of both worlds, believe we've really done it this time. Not only can you use the appropriate Windows printer driver, where necessary, but you can bypass the driver and print direct where possible.

When printing to non-HP compatible laser or IBM dot matrix compatible printers you are forced to use the Windows printer drivers. This includes Postscript type printers, fax machines, etc. However, if you can print to the printer using TAS Professional 5 for DOS you can print direct to the printer in Windows. Why would you want to do this? By printing direct you get the **exact** same output you would under DOS. This means that all the formatting you have already done will still work. Further, if you want to use control characters to create boxes, shading, etc. all that will still work. When printing through the Windows drivers the only option, at this time, is to print in condensed format.

If you can print direct you will want to. You lose no capabililties; you still choose your printer from the list provided by Windows and you can print across the network without having to capture a printer port. Any printer you can access in Windows in other programs you can now access the same way in TAS Professional for Windows. The only difference will be whether your program is using the Windows printer driver to communicate with the printer or uses the internal controls built into the printer.

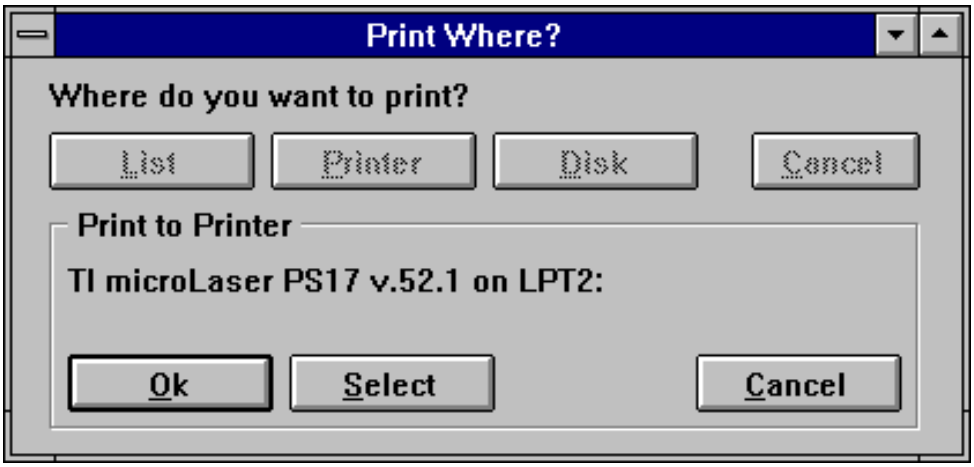
The way TAS knows whether it is to print direct or not is by the entry in the TP5WIN.INI file for the printer. That is described below. Right now we want to show you what the printing dialog looks like.

The print dialog box above is the first one that will appear if your program allows the user to choose where to send the output from the report. If the user chooses List (the default button) the report will be printed to disk first. Then a screen similar to the one below will be displayed.

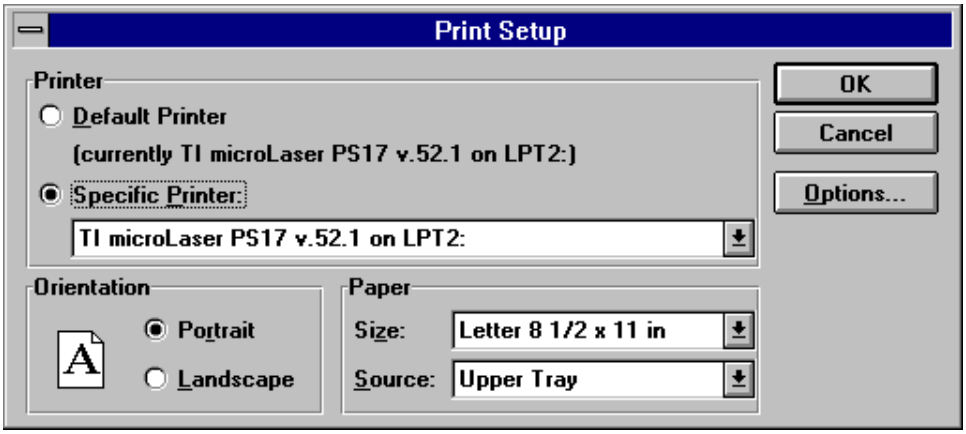


At the top of the screen are 8 buttons and a line of text that tells you how many pages there are in the report and what page you are on. The first two buttons let you move towards the beginning of the report (left button) or the end (right button) one page at a time. The next two buttons will take you to the first or last page of the report. The following two buttons will increase or decrease the size of the font. By decreasing the size of the font you will get more characters to fit in the window, however, it may get so small that you won't be able to read it. The next button will let you print the report to a printer. You will get the standard choose print dialog, as described below, however you will also be able to choose the page range to be printed. The last button lets you exit from the report list. You can also press the ESC key and get the same result.

If you press the PRINTER button the following screen will be displayed.



You first option is to choose the default printer, which is displayed in the box. You can also choose the SELECT option which will allow you to display the standard Windows printer dialog below.



You can then click on the drop box for Specific Printer to get a list of printers for your system. Once you choose a printer you will return to the choose printer dialog. Press the OK button or enter O and the program will start printing. When you choose a printer the program will get the appropriate information from the TP5WIN.INI file for that printer. This information will tell it whether to print directly to the printer or use the Windows driver (more about this later).

If you want to print the report to disk just click on the DISK button (or press the D key) from the original dialog box and the dialog below will be displayed.

You can either enter a file name directly, or click on the SEARCH button. If you enter a name for a file that already exists the program will alert you to that and you will be given the chance to choose whether or not to overwrite the existing file (if any). When you click on the OK button (or enter O) the report will be written out to disk.

All of the choices, including the original where to print choices, offer a cancel option. If you click on the CANCEL button once you are in the Printer or Disk option you will return to the original options. If you choose CANCEL again the program will exit from the print where dialog. You can tell, in your program, that the CANCEL option has been chosen by checking the **CANCEL_PRT()** function. For more information on this function please refer to **Chapter 6 - Function Reference**.

TP5WIN.INI and How to Access It.

Most Windows programs have what's called an INI file (because of the .INI extension) and TAS Professional for Windows is no exception. This file plays a role similar to the TAS50.OVL file but it has different values. TAS Professional for Windows still uses the standard TAS50.OVL file to get information such as the default data dictionary, date and time type, etc.

This file can be in two different locations. The program first looks in the user's default path. This is where the first program is executed. If it doesn't exist there the program looks in the user's \WINDOWS sub-directory. If the file doesn't exist there it is created in that sub-directory. By using this procedure you can create different TP5WIN.INI files for different applications, users, etc. If you ever feel that you've messed up the beyond repair, just delete it. You can always specify the initial run program in the command line for TP5WIN.EXE. If the INI file is not found and there is no program in the command tail TAS will look for the programs BKMENU.RUN and MENU.RUN. If neither of those are found you will get an error message. However, once the program is running you can then go back and reset the values, as appropriate, using the maintenance program described below. You can also edit the file directly with a text editor if you wish.

The installation program automatically puts a copy of TP5WIN.INI into the same subdirectory where you install TP5WIN.EXE. You may end up installing TAS Professional for Windows several times in separate subdirectories. Each time you will put a new copy of TP5WIN.INI in that same subdirectory. Since you may have already changed colors, added printers, etc. and don't want to go through that work again, you can copy previously edited versions of the TP5WIN.INI file to the new subdirectory. Also, you can keep a single copy in the \WINDOWS subdirectory. Then when you install new copies of TAS Professional for Windows just delete the copy of TP5WIN.INI in the new subdirectory. This will force the program to look for the file in the \WINDOWS subdirectory. NOTE: This will also work if you are loading Windows from a network drive and have all the programs in a single directory on the same or different network drive. By setting the InitialProgramName value in TP5WIN.INI you can control where the default drive is for each user.

TP5WIN.INI contains, at a minimum, three different categories. If you aren't familiar with Windows INI files each category starts with a name surrounded by square brackets. These three categories are [Color], [Video] and [Misc]. Following each category is one or more names for different values used within TAS. When TAS is initially loaded (or when a printer is chosen) the program looks at the INI file for the values for each of the options. If a value is not provided the internal value is set to a default. The lists below give each of the options, their default value a short description of the item. Please note, when a value is given as True or False it is actually 1 or 0 in the TP5WIN.INI file.

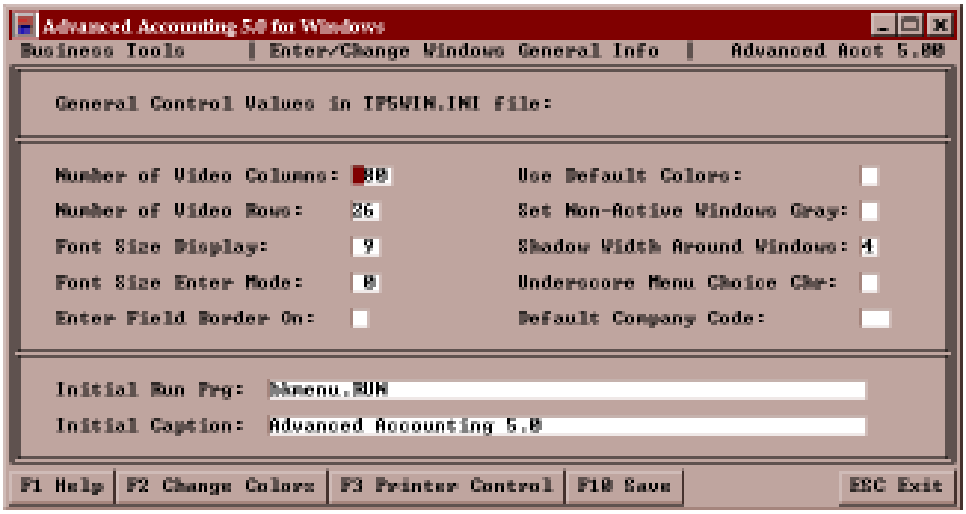
Option Name	**Default Value**	**Description**
[Color] category		
NormalBkg	BtnFace	Base window color.
NormalText	BtnText	Base window text color.
EnterBkg	Window	Entry field background color.
EnterText	Black	Entry field text color.
WindowBkg	Window	Created window (WINDOW, WINDEF) background color.
WindowText	Black	Created window text color.
MenuBkg	Window	Menu (MENU, NMENU) background color.
MenuText	Black	Menu text color.
ButtonBkg	BtnFace	Button background color.
ButtonText	BtnText	Button text color.
ChoiceBkg	Highlight	Choice bar background color (Menus and LISTF/LISTM).
ChoiceText	HightlightText	Choice bar text color.
EColorBkg	Yellow	Entered characters during a LISTF/LISTM command background.

EColorText	Black	Entered characters during a LISTF/LISTM command text.
ShadowBkg	WindowFrame	Background color for raised frame around windows and menus as described above.
ShadowText	WindowText	Text (or border) color for raised frame around windows and menus as described above.
[Video] category		
VideoRows	26	Number of rows in base form.
VideoCols	80	Number of columns in base form.
FontSizeReg	-9	Regular display (non-entry mode) font size.
FontSizeInp	-9	Entry mode (during ENTER command) font size.
BorderOn	False	Whether there is a border around the entry field during display mode.
ShadowWidth	4	Size in pixels of the raised border around windows and menus as described above.
[Misc] category		
UnderScoreMenuChr	True	If this is set to True then in the NMENU command if TAS finds a & before a character in the menu line it will remove the & and put an underscore _ under the character immediately to the right of the &.
InitRunPrg	MENU.RUN	This is the initial program that will be run if there is no program in the executable command line.
InitCaption	blank	This is the caption that will appear in the caption block at the top of the default form.
DefaultCompanyCode	blank	This is the company code that will be used as the default value when the files are opened. If no value is here TAS will use a blank value which is the normal default. If you save a value here it must be two characters. This option follows the same rules as standard company codes.
AllowGray	False	If this is set to True then inactive windows will be "grayed out" unless the command **GRAY OFF** has been executed. For more information please refer to the command **GRAY** in **Chapter 5 - Command Reference**.

Along with the categories listed above there will be a category created for each printer you add to the file. The category name for each printer will be the printer name, for example, the printer HP LaserJet Series II on LPT1 would be set to: [HP LASERJET SERIES II]. The entries are different if you are printing direct to the printer or if you are using the Windows driver. These entries are:

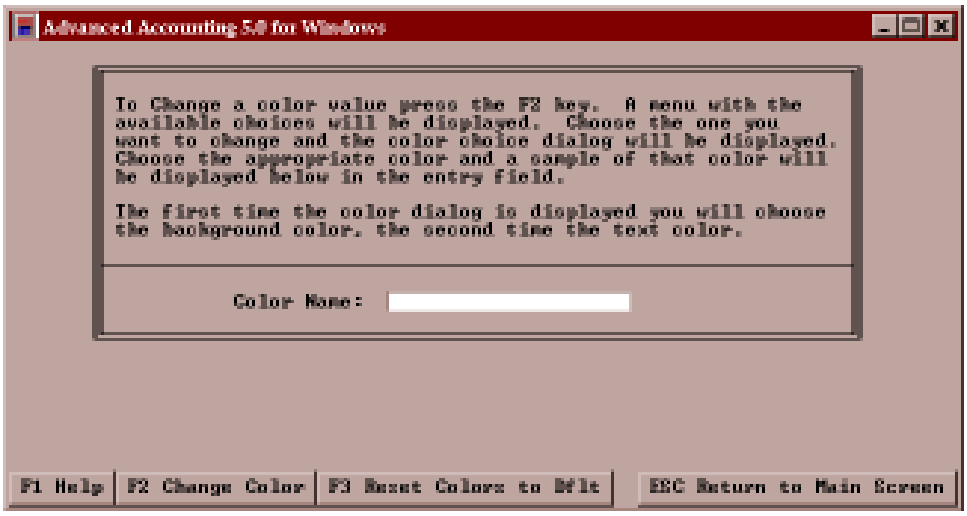
Option Name	Default Value	Description
When printing direct		
PrintDriver	HP2	The name of the .CTL (printer control) file on the disk.
PrintWidth	80	The number of columns wide you can print.
PrintLines	60	The number of lines you can print on the page.
PrintPaper	66	The total number of lines possible to print from top to bottom.
PrintLaser	True	Whether or not this is a laser printer. This means that PrintPaper for this type of printer would normally be 60 lines also. To make it 66 you have to send special control characters before starting to print. At the end of the report you'll need to send special control characters to change back to 60 lines.
When printing through a Windows Driver		
FontName	Courier New	The name of the font to be used in the report. You should probably use a fixed pitch font, although you don't have to. However, if you don't, you may not be able to line up the columns as you expect. Also, Courier New is a True Type font and should be available on all printers.
RegularSize	12	The size of the regular pitch font.
CompressedSize	7	The size of the compressed pitch font.
Bold	False	Whether to use bold option or not.
MarginSide	0	The number of columns to offset the report from the left side. In general this will be 0, however, you may find you need to set this value when printing to a fax machine.
MarginTop	0	The number of rows to offset the report from the top of the page. In general this will be 0, however, you may find you need to set this value when printing to a fax machine.

We provide a program to manage this file called TASINI.RUN. This program must be run in Windows and can be run directly in the command line, i.e., TP5WIN.EXE TASINI. When you execute the program the first screen displayed is below:



The entry fields on the screen correspond to the TP5WIN.INI listings previous. Two issues that need to be dicussed here are Display/Entry field sizes and the Enter Field border. Due to the font type we use on the screens (to maintain compatibility with DOS) there are only a few display sizes available. The one that fits the screen the best for normal size is 9. The next size down is 6. The default mode of displaying a field is without a border. However, you can choose to put a border around the display field by enterying Y at the Enter Field Border On option. This will do the following: When the field is in normal display mode (not being entered) the program will put a border around the field. This will also reduce the size of the field so that it will fit within the border to 6 (this can be very small depending on your monitor and age). When the field is in entry mode the border will be removed and the font will be increased to the Enter Mode Size value. You should be aware that your only options in size for this font are 5, 6, 9, 12 and 14. Even if you change these values you will not change the size of the font used for general text on the screen.

At the bottom of the initial screen are the buttons for entering colors and printers. To enter new colors click on the Change Colors button or press the F2 key. When you do the following screen will be displayed.



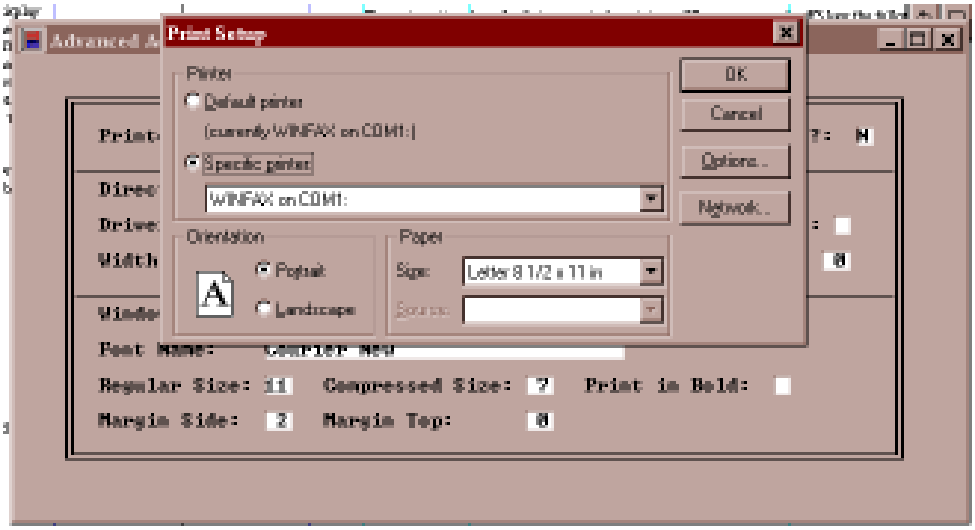
Next press the F2 key or click on Change Colors and a list of the color options will be displayed. Choose the appropriate color. When you do the following color dialog will be displayed.

You can then choose the appropriate color, first for the background and then the text. Note that even though you can create custom colors that can be used those values will not be saved from one session to another. Also, the color dialog that is displayed may be different for your system depending on your video card and your setup.

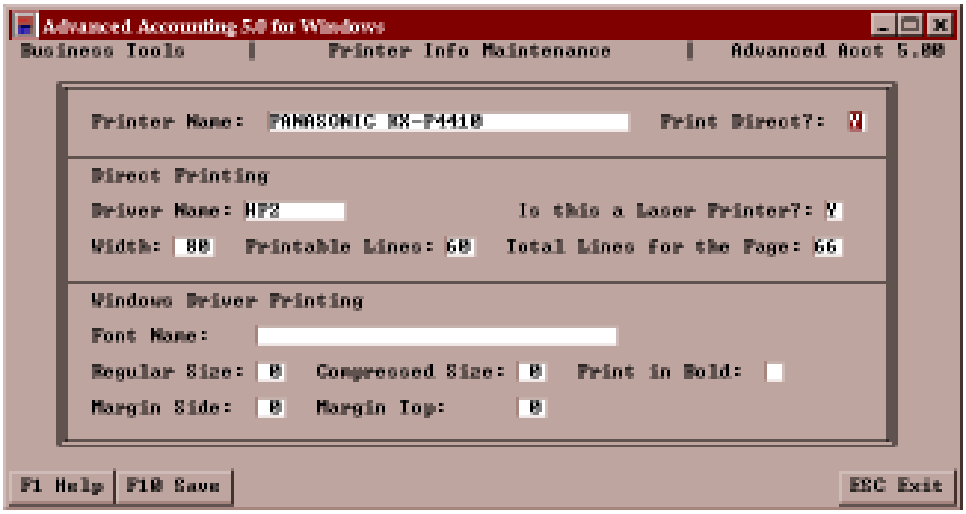
If you wish to reset all colors back to the default values then press the F3 key or click on the Reset button. When you're finished choosing colors press the ESC key or click on the EXIT button.

NOTE: When you make changes to either the first screen options or color values you **MUST** save the new value to TP5WIN by pressing the F10 key or clicking on the SAVE button when you're on the first screen. If you don't any changes you have made will be discarded. This, however, does not apply to printer entries as described below. Also, you have to exit completely out of the current program and then run TAS Professional for Windows again before the changes will take effect. This also does not apply to changes made to printer information.

The main option from the first screen is for printers. When you press the F3 key the following screen is displayed.



Your first task is to choose a printer or just press the ENTER key for the default. Once you have chosen a printer click the OK button or press the ENTER key. When you do the printer dialog box will be removed and the following screen will be displayed.



The first entry is whether or not you're printing direct. If you are then enter Y here. If you enter N then you will be using the Windows printer driver. If you answer Y to Print Direct you will be entering values in the middle box. If you answer N you will enter values in the lower box. Each entry field will default to the appropriate value if a value didn't already exist for it. Further, if you have entered information for this particular printer previously, those values will appear on the screen. You can also change from printing direct to printing through the driver and vice-versa without any ill effect. You can also try printing with your settings and then come back and adjust them any number of times. Each time you save the printer information the appropriate block in TP5WIN.INI is deleted and the new information is saved.

You can print to a printer without having this information entered for it, however, the program will assume that you are printing through the Windows driver and will use all appropriate default values for that printer. If you want to print direct you must have it added to the TP5WIN.INI file. The easiest way to do that is to enter the information here.

NOTE: As opposed to the main screen information and colors when you save the printer information from this screen it is saved immediately to the TP5WIN.INI file. Even if you press the ESC key at the main screen and answer No to saving changes, any changes you saved about printers will be in the TP5WIN.INI file.

Field Search Process

When you or your user clicks on a field with the mouse TAS Professional for Windows has a specific process in deciding not only whether to allow you to make that field active, but which line of code in your program that field is attached to. This is called the Field Search Process. The rules that apply are as follows:

1) If the user is not on a field currently they will not be allowed to move to a field by clicking on it. This means they can't move from a menu to a field just by clicking on the field.

2) The program then checks for a **NOCLICKOFF** option in the ENTER command. If this exists they won't be allowed to click on to another field.

3) Next it checks for a for an **ENTER** command in your code that matches the field. This is probably the most complicated part of the process and there are some twists here also. First, normally the process will start checking from the beginning of the program for the appropriate line. However, if an **UPAR** command has been executed then the search process will start from that line instead. Each time an **UPAR** command is executed (in the normal course of operating, not during the field search process) this starting point is reset. The search process will continue from that line until the end of the program or the first **CLIK_SRCH_LIMIT** command. Through the use of both of these commands you can easily control where the search process starts and ends for different parts of your program. This might be necessary if you have the same field on two or more screens, but you only enter it in one location.

   For example: Let's say you have two different entry screens in a program. The first is a general entry screen, the second for a particular sub-group of fields. You would normally put an **UPAR** command after the second **MOUNT** and before the first **ENTER** command so that the user couldn't move towards the beginning of the program by pressing the UP ARROW key when in the first entry field. If you didn't do this, and the user pressed the UP ARROW key as they are apt to, your user would get a 'Field Not Mounted' error message and you would get a support call. In Windows you or your user are more likely to use the mouse to click on a field you want to move to and the field search process is what needs to be controlled. This time, if you don't put in the proper controls, the **ENTER** command for the field being clicked on might work properly, but when the user presses the ENTER key to move to the next field the lines being executed are not the ones supposed to be next and then, you'll get the 'Field Not Mounted' error or worse. The sample code below shows a situation that may occur.

```
start:
   mount screen1 type s
   upar
   enter fld1
   enter fld2
   enter fld3
   ... other code ...
   clik_srch_limit     ;put this here to keep the click
                       ;process from checking too far down
                       ;the code also!

start2:
   mount screen2 type s
   upar
   enter fld4
   enter fld5
   enter fld6
   ... other code ...
   clik_srch_limit
   goto start
```

In this example above assume that that FLD1 is on both screens, however, it's only being entered in the START group. If the user clicked on the field when SCREEN2 was mounted, the process would start at the beginning of the program (assuming there were no **UPAR** commands) and would find the **ENTER** that applied to SCREEN1. Then when the user pressed the ENTER key while in FLD1 the next line to be executed would be **ENTER** FLD2 which would give the 'Field Not Mounted' error assuming that FLD2 wasn't on the second screen also. By using the **UPAR** and **CLIK_SRCH_LIMIT** commands you can make sure this won't happen to you or your users.

Another change in the Windows version is that the user can click on the array element of a field and the program can correctly determine which field this is on the screen. So, you could have, for example, all twelve months on the screen and have code allowing the user to enter these values with a single ENTER command and a FOR/NEXT loop as follows:

```
for(cntr;1;12;1)
   enter monthly_amounts[cntr]
next
```

The field search process will set the array element to the proper value internally for that ENTER command. However, it is up to you to reset the CNTR value before you leave the ENTER command (generally in a POST option) so that the user continues from the next field on the screen if they press the ENTER key. If you don't update the value the entry process will continue with the field that matches the next CNTR value. For example, if the user is on element 2 and clicks into element 5. If they press the ENTER key only the entry process will continue with element 3. However, if you use something similar to the code below, the entry process will continue with element 6 and everything will look proper to the user:

```
for(cntr;1;12;1)
   enter monthly_amounts[cntr] post reset_cntr()
         {
         func reset_cntr
               if windows() then cntr = get_elem_num()
               ret .t.
         }
next
```

Notice how we use the **WINDOWS()** function above to make sure we running in Windows. This allows us to use this code in either DOS or Windows. Because the **GET_ELEM_NUM()** function always returns 0 in DOS, if the **WINDOWS()** function had not been used to prevent this section of code from being run under DOS, your user would have been permanently stuck on the first element! This works for any element in the array since it will reset CNTR to the current element number. CNTR will get incremented properly by the FOR command when the loop continues. Finally, notice how we used the new code control braces. This allows us to keep the code that applies to the **ENTER** command very close. And, since this works in both DOS and Windows we don't have to worry about those issues when we use them.

4) If the field search process finds the proper **ENTER** command it now looks for a PRE option in the new command. If a PRE option exists, the program will evaluate the expression and if it returns false (.F.) the user will not be able to move to that field.

   Don't forget, you can use the **GET_ELEM_NUM()** function in the PRE option also. The only extra code you need here is to make sure that the field is being clicked on. If it isn't then the function will not return the proper element number. So, using similar code to above, you would do the following:

```
for(cntr;1;12;1)
    enter monthly_amounts[cntr] pre check_fld() post reset_cntr()
            {
            func check_fld
                    if clicked_on()
                        if monthly_amounts[get_elem_num()]=0 then ret .f.
                    else
                        if monthly_amounts[cntr]=0 then ret .f.
                    endif
                    ret .t.
            func reset_cntr
                    if windows() then cntr = get_elem_num()
                    ret .t.
            }
next
```

   Some things to note about the code above: First, we didn't have to check both **WINDOWS()** and **CLICKED_ON()** functions. In DOS the **CLICKED_ON()** function will always return false (.F.). Second, we didn't change the CNTR value if **CLICKED_ON()** was true. That's because we're just checking the PRE option and if it returns false we don't want to change where we are in the array. Finally, notice that we have two different UDFs in the code control block. There is no limit the number of UDFs or other code lines you have within the braces.

5) Finally, if the PRE option of the clicked on field returns true or if there is no PRE option, the program checks the VLD (valid) for the current field. If it returns true, or if there is no VLD for this field, the program will move to the new field.

Sample Program

There is a sample program that demonstrates some of the features of TAS Professional for Windows. This program is installed in the SAMPLE subdirectory. It is a standard .RUN program, however, it is meant to be run in Windows only.

TAS

PROFESSIONAL

VERSION 5.1

Appendix A - Telecommunications

INTENTIONALLY BLANK

Communications routines have been removed in TAS Professional 5.1 due to unreliability of library used. They will be returned in the 6.0 (Windows) version.

INTENTIONALLY BLANK

TAS
PROFESSIONAL
VERSION 5.1

Index

INTENTIONALLY BLANK

A .....

B

C

D .....

E

F

G .....

H

I

J

K

L

M .....

N .....

O .....

P

Q .....

R .....

S .....

T

UV

W

XY