Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Legal Notes

Information in this document is subject to change without notice and does not represent a commitment on the part of Parker Software, Incorporated. The software described in this document is furnished under a license agreement (see below). The software may be used or copied only in accordance with the terms of this agreement. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose other than the purchaser's personal use without the express written permission of Parker Software, Inc.

> **License Agreement:** This product is provided to the user to operate on one computer only. Copies of this program and its related data files may be made for archival/back-up purposes only, and not for use on another computer. It is against the law to copy the software for use on more than the licensed number of machines.
>
> **Liability Disclaimer:** This product is provided on an 'as is' basis, with no warranty, promise, or claim of merchantability, suitability, of fitness for a particular purpose. Parker Software, Incorporated accepts no liability for the product except that it be provided on defect-free media. If the media provided with this package is defective, return the defective media, and Parker Software will provide a replacement copy at no charge.
>
> With the previously mentioned exception, Parker Software, Incorporated, its employees, contractors and subcontractors offer no warranty for this software. This disclaimer includes, but is not limited to loss of information, time, resources and/or computer access due to the use, misuse, or inability to use this product.

Unless otherwise noted, all names of companies, products, street addresses and and persons in this manual are part of a completely fictitious scenario, and are designed to document the use of a Parker Software product.

Company Information

Parker Software, Incorporated is a software research and development company that specializes in business-oriented information systems. The company is based in Grove City, Pennsylvania, and has developed computer software solutions for businesses since 1991.

Parker Software offers a variety of programs that use VariForm, such as point-of-sale, inventory, production management, client/contact management, time scheduling, accounting, electronic data interchange, and electronic bulletin board systems.

For inquiries about products, local consultant/dealers, or technical support, contact our main office:

Parker Software, Incorporated
P.O. Box 350
Grove City, Pa. 16127-0350

Fax/BBS support line at 412-458-8120

Fax:	Group 3 or Group 2: 2,400 - 14,400 Baud
Modem:	300-14.4 KBaud 8-n-1, using PC characters, optional ANSI colors Supporting X, X1K, Y, and ZModem file transfers
CompuServe:	71155, 1470
InterNet:	@cis.com.7155,1470
Voice support:	412-458-7576 East Coast Time: 9am to 6pm, Monday through Saturday

What is VariForm?

What can it do?

VariForm is an Information Management System that allows programmers and designers to generate stand-alone and relational database systems with data entry and data management functions. The word 'VariForm', which means "comprised of or capable of assuming a variety of shapes or forms", describes the environment that is presented to the end-user by applications developed with the program. Information may be presented and manipulated in a variety of formats to provide different views of the data.

VariForm, as a complex Relational Database Management System, allows users to store, manipulate, manage, and retrieve large amounts of information stored in one or more 'databases'. Some example database applications are:

- Mailing Lists of customers, clients, and prospects
- Project Scheduling / Time Management
- Customer Purchasing (Point of Sale) / Sales Analysis
- Product Catalogs / Inventory Management
- Accounting Systems for single and multiple ledger journaling
- Automated Billing / Receipts Analysis / Dunnings notice generators

VariForm provides powerful Data Entry, Querying, Mapping, and Application Development Systems for both the developer and the end-user. Applications developed in VariForm can be produced or customized cost-effectively, quickly and with lasting value.

Sharing Queries and References reduces development costs through program and data re-use. VariForm's Multi-Dimensional Table and Relation system provides for automated sharing of data between resources without a lot of programming time. References only have to be entered one time, even if the structure changes. Queries are stored as database records, allowing forms, reports, and interfaces, as well as different fields on the same form to share or reuse queries.

VariForm provides the developer/customizer with powerful RDBMS tools to speed along development. There are over 250 database management and control options available through a menu-driven interface, a "Command Line" prompt, and through a built-in, procedural programming language.

VariForm's robust display system reduces customization time. There are over 700 pre-defined format and display control options that free the programmer to develop the application, and let the formatting take care of itself. Queries, Browsers, Forms, Sub-Forms, Functions, Procedures and Help can be attached to function keys in seconds via data entry, eliminating a time-consuming chore for the developer. The programming language is also accessed through data entry, providing user/developer assistance on programming details. **VariForm does the math, and allows the designer the freedom to design.**

It is easy to put on the premium quality 'finishing touches', with VariForm. The built-in procedural language and query system provide powerful cross-referencing tools to make sure that 'last minute change' contains proper field names and syntax. Cue lines, which tell the user

what actions are expected or available, can be filled in through the Display data entry system. Function keys can be grouped together to provide a more consistant user interface, and the resulting groups can insert consistant text into the cue lines.

What is VariForm?

What Can it Do?

The VariForm Help System is extensible, and without peer. Direct cross-references, topical lists, hierarchical categorical lists, as well as a major and minor help index make any VariForm application easier to learn, without forcing the developer to type in redundant information. Text for the help system can be entered, or imported from other user documentation. Contents of the help system can also be exported to produce supporting documentation.

VariForm contains built-in "Upsizing" capacity to accommodate future growth. By changing one internal variable, a single-user application becomes multi-user capable, with **internally managed file and record locking**, as necessary to maintain the data flow. Adding a communications package to the built-in mapping system provides **immediate expansion to Client/Server** and Wide Area Network capacity.

VariForm can interact with other programs. Users can define structural interfaces to read or write files from flat files, other database programs, spreadsheets, and word processing programs.

VariForm helps the re-developer. VariForm's modular procedural language and query system also provides room for developer notes and comments. This helps the second-round developer to understand the first-round developer's field and formula conventions, saving learning and error time when expanding an existing application.

What is VariForm?

General System Capabilities

VariForm, as a database management system, provides the developer with:

- Multidimensional Relational and Cross-Referencing Database Systems
- Data Mapping with multiple, complex relationships between Databases
- Automated, record-sensitive data encryption
- Field, Screen, Menu, and Database controlled security levels
- Data Import/Export facilities with XBase, EBCDIC & binary file support
- A built-in procedural programming language with assisted data entry
- 40 fully programmable function keys per data entry field or menu line

Applications developed in VariForm can offer the user:

- A Data Entry system that can cue the user with available options
- A multilayer, multiple-index, cross-referenced help system
- Report Generation with multifont support for an unlimited number of printers
- Internal Disk Caching
- Variable-sensitivity error logging with user, time and date stamps

All of these capabilities, including the programming language, are accessed through our unique data entry system. VariForm allows a data system to be designed in an hour or less. Interfaces can be edited on-the-fly with built-in developer notes. The language interface can also diagnose potential errors before they cause operational problems.

This is a powerful application design system that offers the ultimate in user control, modifiable help text, field-sensitive cue-lines, function key availability and system performance.

All information for the user interface is stored in database records. Providing multilingual support for international needs is quite straightforward, and can even be configured to provide a different language for each user.

The functions listed above are built with one purpose in mind: **Help the developer provide a powerful, professional, custom application to the end-user quickly**.

What is VariForm?

Why Reinvent the Wheel?

Parker Software has a variety of pre-developed modules for VariForm that are solid, business applications that the user can build upon. These modules can reduce the initial start-up time by providing a foundation of data structures, relationships, procedures and help to start with, instead of 'starting from scratch'. Currently available application modules are listed below. Planned modules, along with a tentative release schedule, are listed in the back of this manual, starting on page .

◦ Electronic Data Interchange translation system with pre-designed data entry screens and reports for a variety of EDI uses.

◦ Point-of-Sale and Inventory Control Systems for retail and catalog-style store fronts.

◦ Single and Multi-ledger Accounting Systems for bookkeepers and accountants

◦ Production line time estimation / workload scheduling and analysis for assembly-line managers.

◦ Prospect / Lead management and client progress analysis for sales force personnel and telephone solicitors.

◦ Asynchronous communications system for communications with the outside world, and Hosting to provide a dial-in site for clients.

What is VariForm?

System Philosophy

Information Systems historically have fallen into one of two broad categories: Database Managers, which call themselves DBM's, RDBMS's, and Programming Systems, which use names Like "...Environment", or "Object-Oriented System". These two categories represent philosophies about not only how information should be managed, but what skills are neccesary to modify the system. **VariForm represents a new, hybrid of these two philosophies: programs, stored as data, are accessible to the user, empowering the end-user to make detail changes. VariForm supports software reuse through open-ended, modular procedures, as well as data reuse through dynamically structured, stored queries.**

Database Managers, such as Access, dBase, and Profile, provide a rudimentary programming language in a limited environment. While editing a screen may not be difficult, writing a program to format a screen is usually beyond the scope of most basic and intermediate users. A user needs a lot of 'seasoning' to get the hang of making changes to dBase, and laying out new data systems requires a lot of program-oriented forethought. DBM's and RDBMS's provided some sophistication, but little ease-of-access needed by users in a fast-paced business environment. **The VariForm philosophy provides highly sophisticated ("assisted") data entry systems that offer complete accessibility to all application definitions.**

Programming Systems, such as Paradox, provided far more sophisticated and elegant tools for the programmer/developer, allowing programmers to develop their own data entry systems and interfaces under the Paradox environment. This made development of "user-friendly" software easier, but did not to make the application easier to maintain or modernize. Developers were forced, to a large extent, to either release their 'top secret' source code to their customers, or to provide maintenance to their clients forever. **The VariForm philosophy gives the user complete, assisted access to the application, reducing the maintenance burden on the developer.**

If the development environments are really so easy to use, why does the source code always cost so much extra? The "ease-of-use" was for initial developers, who were versed in programming languages, and who 'hand-crafted' solutions for clients. **VariForm philosophy recommends help, cue-line, and reference assistance for all levels of users and developers.**

VariForm brings the development enviroment concepts into a database manager, and provides a programming language and environment totally contained in databases. The user / developer can immediately see if the changes they make accomplish what they were intended to do.

VariForm is a database management system that can can be modified by end-users, and a programming environment that is manipulated as a set of databases. This provides sophisticated development tools to the programmer, as well as easy-to-use, and easy-to-change tools for the basic or intermediate user.

While any development task should be supervised by some sort of seasoned professional, **it is the philosophy of VariForm that an application, once it has been developed, should be understandable to and changeable by the end-user**.

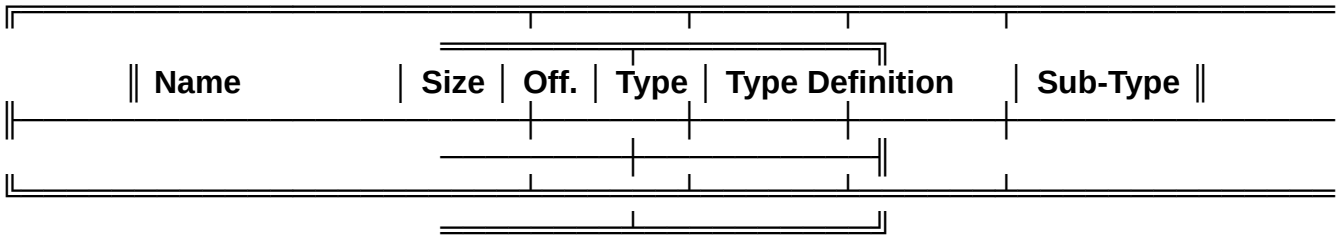
About this Manual

Preface: This manual has some conventions and abbreviations to describe keys, actions, and databases. Any abbreviations not described in this section are covered in the glossary and appendices at the end of the manual.

Database Layouts

Most sections in this manual begin with a 'data structure description' block. The purpose of this block is to show the reader all of a database's fields along with some basic information about each field's contents. All systems in VariForm are described and controlled by one or more databases.

Below is a sample layout, followed by a description of what the columns in the structure refer to.



Column Explanations

Name contains the names of the fields in the database. The field name is used in designing reports, data entry screens, and formulas that use or fill-in these areas in a database record.

Size is the size of the field in (8-bit) bytes. The size indicates how much space this field consumes in each record. In the case of Character Fields, the size also indicates how many 'typable characters' will fit in the field (actually, size - 1 characters will fit in a Character field.) For example, a field containing city names might be 16 bytes in size, allowing the user 15 typable characters, such as "East Pittsburgh".

Off. or "offset", refers to the exact physical location of the field in a database record. This valuer becomes important when discussing "unioned" fields, which may share the same physical locations within a record.

Type is the type of information contained in the database field, such as 'Character', 'Numeric', 'Date', and so forth. This number tells VariForm what specific kind of information is stored within the field. For more information on Types and Sub-Types, see the section on the 'Database of Database Fields', which begins on page .

Type definition is the word description of the database field type for human reference. This

'spells out' the meaning of the type number.

Sub-Type is a number that describes the field contents in more detail, as a supplement to the "Type" field. This tells VariForm how to use a field, such as permitting spaces, or enforcing the uniqueness of this field's contents.

About This Manual

Keyboard Designations

General

Depending on the specific style of keyboard being used, these generic descriptions may vary slightly from the exact user's keyboard:

◦ The 'main typing area' is the section of the keyboard where the letters are. Numbers, enter, tab, shift, control, and alternate keys are also in this area of the keyboard.

◦ The 'numeric keypad' is right of the main typing area, and consists of the numbers 0-9. These keys serve a double purpose: if the 'Num Lock' key is active (usually a light on the keyboard indicates this), the keypad keys are treated as a spare set of number keys. If the 'Num Lock' key is off (light is off), the keypad keys present a set of 'cursor movement' keys, such as arrows, home and end. VariForm will respond to these keys according to the status of the Num-Lock key.

◦ The 'function keys' are either in a line across the top of the keyboard, above the main typing area, or in two columns to the left of the typing area. These keys are marked with an F, and a number from 1 to 10 or 12.

◦ The 'enhanced keys' are only available on 'AT-style' keyboards. These are a spare set of cursor movement keys between the main typing area and the numeric keypad, and are not affected by the NumLock key.

Alphanumeric keys refer to the common typing keys on the keyboard that are letters or numbers, such as 'Capital A' (A) or 'The digit one' (1).

Punctuation keys refer to the keys that represent single, typable characters that are not alphanumeric, such as 'Percent' (%), 'Semicolon' (;), 'Slash' (/), and so forth.

The Shift key refers to one or two keys that are marked "Shift", located in the lower corners of the main typing area. The shift key affects all other keys on the keyboard. Fox example, Shift-A produces an uppercase "A", while the A key by itself produces a lowercase "a". This is accomplished by pressing and holding down the shift key while typing the other key.

The Control key refers to one or two keys that are marked either "Ctrl" or "Ctl". They are usually located in the lower corners of the main typing area. This key affects all other keys on the keyboard, and is used in the same way the shift key is used.

The Alternate key refers to one or two keys that are marked "Alt". They are usually located near the Control Key. This key affects all other keys on the keyboard, and is used in the same way the shift key is used.

The Escape key refers to a key in the upper, left-hand corner key that is marked with either "Esc" or "Escape".

About This Manual

Keyboard Designations
Cursor Movement

Cursor Movement keys refer to the keys that describe movement: Page Up, Page Down, the Arrow Keys (Up, Down, Left, Right), Home and End. Where this manual describes cursor movement keys 'on the numeric keypad', this means when the NumLock key is off.

<Pg Up> is the Page Up key, and is also the '9' key on the numeric keypad.

<Pg Dn> or <Pg Down> is the Page Down key, and is also the '3' key on the numeric keypad.

<Up> is the 'Up Arrow' key, which is the arrow key pointing away from the user, and is also the '8' key on the numeric keypad.

<Down> is the 'Down Arrow' key, which is the arrow key pointing towards the user, and is also the '2' key on the numeric keypad.

<Right> is the 'Right Arrow' key, which is the arrow key pointing to the right, and is also the '6' key on the numeric keypad.

<Left> is the 'Left Arrow' key, which is the arrow key pointing to the left, and is also the '4' key on the numeric keypad.

<Tab> is the Tab key, usually on the left edge of the main typing area.

<BackTab> or <sTab> is the 'BackTab', which is a 'shifted' Tab Key. This key is

<Enter> is the 'Enter Key'. On most keyboards, this key has the word 'Enter' on the top; some keyboards use a return symbol (←┘).

About This Manual

Keyboard Designations

Special Keys

<Special keys> are always enclose in angles (< or >), and refer to other keys on the keyboard. Where this manual describes special keys 'on the numeric keypad', this means when the NumLock key is off.

<F1> is Function Key 1, which is the key with 'F1' on the top.

<sF1> or <Shift F1> is a 'shifted' Function Key 1. This means the user will press and hold the shift key down, and then press the <F1> key.

<aF1> or <Alt F1> is an 'alternate' Function Key 1. This is similar to a shifted Function Key 1, except that the user presses the Alt key instead of the shift key.

<cF1> or <Ctl F1> or <Ctrl F1> is a 'control' Function Key 1. This is similar to a shifted Function Key 1, except that the user presses the Control (Ctrl) key instead of the shift key.

<Ins> is the Insert Key, usually right of the main typing area on the keyboard, which has either (INS) or (Insert) on top. This is also the '0' on the numeric keypad.

<Del> is the Delete Key, usually right of the main typing area on the keyboard, which has either (DEL) or (Delete) on top. This is also the '.' on the numeric keypad.

<Esc> or <Escape> is the Escape Key, usually marked 'Esc' on top, and almost always located in the upper left-hand corner of the keyboard.

Installation

First-Time users should read this entire section before attempting to install VariForm.

VariForm has only three **System Requirements**:

+ The computer has enough RAM (chip memory) to run the program (400,000 bytes)

+ The storage device (hard drive, network drive, et cetera) has enough space for the files (6,000,000 bytes)

+ DOS (3.0 or higher) has enough "File Handles" for the database system (150-230)

Step 1 : RAM:

To check available RAM: At the DOS prompt, type:

DOS 3.0 to 5: CHKDSK
DOS 5.0 / higher: MEM

And press <Enter>
Look for the number next to the words:

DOS 3.0 to 5: Free Memory
DOS 5.0/higher: Largest executable Size

If the number is 400,000 or higher, there is enough RAM for VariForm to run.
If the number is below 396,216, there will not be enough 'conventional' memory to run VariForm.

Solving Out of Memory Problems:

Reboot the computer, and check available RAM again. If there is still not enough RAM, there must be some TSR (Terminate and Stay Resident) programs in the computer's startup files.

Check the config.sys, and autoexec.bat files for lines that contain:

"DEVICE = ..."
"INSTALL = "

If you are using DOS 5.0 or higher, and have available Extended or Expanded Memory, change these lines to DEVICEHIGH and INSTALLHIGH statements, and load other residents with LOADHIGH or LH statements.

If you are using dos 4 or 3, you must remove some of the resident programs before continuing with the installation.

For more information on memory management, see appendix 7: Making Room, which begins on page .

Step 2: Disk Space :

VariForm must be run on a 'hard drive', either locally, or through a network. For users with more than one hard-drive, the drive letter where VariForm will be installed is the name of the 'Target' drive.

If you are installing VariForm on a standalone pc, run CHKDSK on the drive that will contain VariForm (CHKDSK C: or D:). If there are at least 6,000,000 bytes free on the target drive, there is enough hard drive space to install VariForm.

If you are installing this on a network drive, you should logout of the network, log back in, and run a "DIR" statement on the target drive. Some networks, such as Novell v2, and old v3, do not update the free disk space listings for a user while they have logged in to the network.

Step 3: File Handles :

Standalone: check the "CONFIG.SYS" file on your boot disk. It should contain at least:

FILES = ###
BUFFERS = ###
FCBS = ##,##

For VariForm, FILES should be at least 200, and preferably 235. VariForm does not use Buffers, so the line can be absent, or any value. FCB settings have been known to cause startup problems on some clone systems, if the values were less than 16 and 8, respectively. After changing these values with a text editor, such as edlin or edit, you must save the file and reboot the computer for the values to take effect.

Network: if VariForm is going to be installed on the terminal's local hard drive, use the standalone settings.

Novell Network: Warning to Novell users: there is a second file, called config.net, or net.cfg in your network directory, that may contain a line such as "File_handles=###". If so, the number after File_handles, plus the number after the FILES statement of config.sys must total less than 235. For best results:

Change config.sys to FILES = 165
Change net.cfg to FILE_HANDLES = 70

You will need to logout and reboot for these values to take effect.

Network hard drive: Reverse the network settings from above, if you intend to use VariForm on the Network Hard Drive:

Change config.sys FILES = 70
Change the net.cfg FILE_HANDLES = 165

You will need to logout and reboot for these values to take effect

LanTastic Network: in some circumstances, the Server is configured to 'use config.sys' for file handles. If the Server's hard drive is going to act as a platform for multiuser VariForm, run net_mgr, and change the Server Startup settings for file handles to a minimum of 200 files per concurrent user. This is in addition to the network setting changes for each terminal that needs access. Any terminal that also runs the SHARE program will need its parameters increased as well.

Recommended values for share: 200 file/locks, an 24000 bytes or more of filespace, such as "LOADHIGH SHARE /L:200 /F:24000" in the autoexec or startnet batch file.

Excluding LanTastic v6.0, you will need to logout all users and reboot the server for this change to take effect.

Step 4: Install VariForm:

After the necessary changes have been made, insert the 'VariForm CORE' disk in a floppy drive (A or B), make that drive current (A:<Enter> or B:<Enter>), and type "INSTALL" and press <Enter>. The installation program will ask for a target drive and directory, and place the VariForm System in the specified location.

VariForm will also place a 'Batch' file called "Variform.bat" on the root directory of the computer's boot drive. This allows users to run VariForm in one step from the dos command prompt.

If there is not enough hard drive space, RAM, or file handles available when the installation program is run, the program will provide a warning, and offer to make corrections for you.

Step 5: Add VariForm to run-time options

If, after boot-up, the computer provides a dos prompt, such as "C:\>", running VariForm is accomplished in one of two ways:

1: Run the batch file in the root directory (type "\Variform"<enter>)

2: Change to the VariForm Directory (type "CD \(directoryname)"<enter>)
Run the main program (type "CORE"<enter>)

If, after boot-up, the computer enters a dos-shell program, you may have to consult your shell manual for instructions on how to add a new program to the menu. Here is an example procedure for the DOS 5/DOS6 DOSSHELL program:

Select the main menu (Keep selecting parent until the top line reads "top level" or "root")

Select File (Alt-F), New (N), and a requestor will appear

The program name is VariForm

The program's current/working directory is whatever was specified during

installation (our default is c:\VariForm)

The program's command is CORE

No parameters or passwords are necessary

You do not need to pause on exit (press spacebar to remove checkmark)

Select "OK" to save the option, and it should appear and be accessible from the Main Menu of your Dos Shell.

The procedure in Step 5 will also work for Windows 3.1, but the program name must include the full path, such as "C:\VariForm\core.exe", and the program must be flagged as DOS, not windows program. If the requestor asks for memory requirements, the minimum is 420K conventional, preferably 500+, and 0K EMS, preferably 380+.

Getting Started

Basic Term Definitions

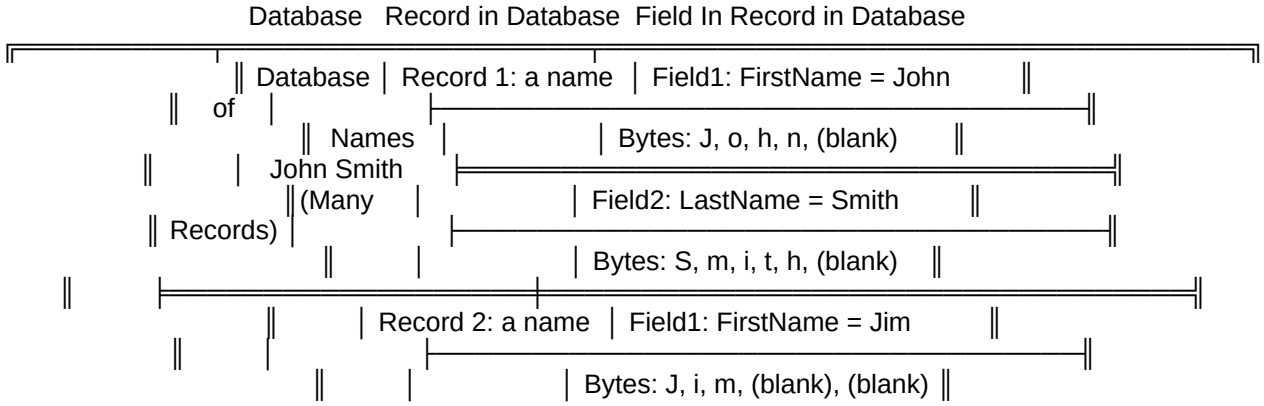
When the computer is turned on, at the beginning of the day, it runs a set of start-up procedures and programs. This set of start-up routines make up the operating system, and possibly establish connections to a network or some other computer(s). This initial processing is called Booting.

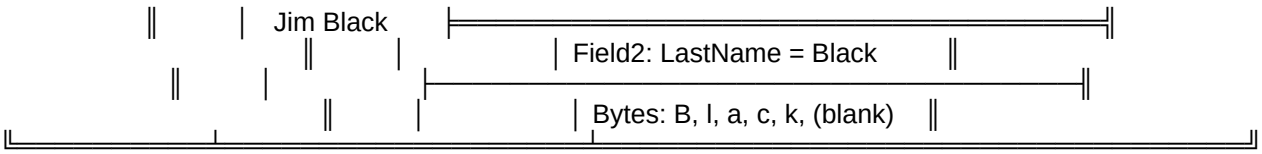
Once the computer has booted, the user has either a dos-prompt, such as "C:\>", or a menu system, such as "Dos Shell", that allows the user to tell the computer what program to run. In either case, calling the VariForm batch program or selecting the VariForm icon from the shell or menu will Start VariForm.

Before getting into the nuts and bolts of operations, there are some critical terms and concepts used by VariForm, as well as most other information systems, that must be defined. First, the definition of a database:

> **Database:** a collection of information. It may encompass one file or many, one computer or many, but in its most basic sense, a database is a collection of structured information called 'records'. A database can be compared to a box of filing cards: all of the filing cards contain similar information about a subject, they are in the same box, hence the same database. A database can be a mailing list of clients or prospects, for example, with one client on each card.
>
> **Record:** a single unit of information. Continuing the illustration above, a record is a single filing card, with room for only one client/prospect. It is the smallest unit of information a database will deal with.
>
> **Field:** a piece of information within a record. On the filing card, a field would be the address line, another field would be the city name, and another field would be the zip code. A field is the smallest unity of information a record deals with.
>
> **Byte:** a single character of information, such as the letter 'A'. Each field is comprised of one or more bytes. On the filing card's zip-code field, 11 bytes might be used for a zipcode (5 digits, a dash, 4 digits, and a terminator).

These four definitions comprise the elemental definition of a physical database. With this, users have been generating mass-mailing lists and basic sales information for years.





Getting Started

Basic VariForm Tools

There are several ways to view and edit database records, and VariForm provides an assortment of 'tools' to make data entry and data reviewing easier.

Menu: A list of selection/options/choices presented to the user. These lists allow the user to tell VariForm what to do: what database to view, what report to run, et cetera. For database viewing and editing, menus will generally call browsers.

Browser: A window that allows users to look at several records in a database at one time. Visually, the database is displayed as a vertical list of records, with one or more 'lines' or rows on the screen used for each record. Changing information on a record requires the use of either an editing key, a procedure key, or a Data Entry Screen.

Editing Keys: function keys, usually within a browser, that allow the user to edit, delete, copy, or add new records. Editing keys often call data entry screens.

Procedure Keys: function keys, on Data Entry Screens or Browsers, that call programs or procedures, such as "update the current record with today's date", or "recalculate the value of the current field".

Cursor Bar: the 'bar' of different color on a browser, which indicates the 'current record'. When using editing or procedure keys, the record highlighted by the cursor bar will be affected.

Data Entry Screen: a window that allows users to edit the contents of a record. A data entry screen usually has the information for only one record at a time, along with prompts for entering information.

Cursor: The flashing line or block on Data Entry Screens that indicates where the next character typed will be placed. Pressing the <Insert> key will change the thickness of the cursor. If the cursor is "thick", the characters being typed will "Insert", pushing existing text to the right. If the cursor is "thin", the characters will overwrite existing text.

Report: A field, a record, a group of records, or a whole association of databases, can be sent to a text file as a 'report'. That file can be viewed from within VariForm with our text editor, sent to a printer, and/or discarded.

Data Entry Browser: A hybrid interface between a data entry screen and a browser. A data entry browser shows many records at one time, like a browser, and allows the user to modify the individual fields on the current record.

Requestor: A window that asks the user for more information.

Index: A manner in which records are sorted, such as "Client Name", or "Zip Code Order". Indexes may contain 1 or several fields. For those familiar with the phrase "Key Field", an index is a named association of 1 or more Fields that generate a composite Key Field. The composite key fields do not have to be unique: in the event of duplication, the physical record number will determine the final order of appearance.

Getting Started

Lookups, Data Dictionaries, and Lists

Every person has been exposed to some form of abbreviation: a two-letter state abbreviation used in mailing addresses, for example. It is much more convenient to put 'PA' on an envelope than to spell out 'Pennsylvania'. Database systems lend themselves to abbreviations: data entry is quicker, simpler, and often more accurate with abbreviations or codes in place of repetitive, lengthy typing.

Abbreviations must be used consistently to be effective, and the user must either know or be able to find out what abbreviations are available. This is accomplished through some database features known as **lookups**, **data dictionaries**, and **lists**.

A **lookup** tells the user the meaning of an abbreviation. When a user types 'PA' in a state abbreviation field, for example, a lookup will display the word 'Pennsylvania' near the field, indicating to the user the meaning of the abbreviation. Lookups confirm the meaning of an abbreviation to the user, which reduces the chances of data entry errors.

A **data dictionary** is a database with records that contain both the abbreviations and their meanings. For this example, the data dictionary would contain all necessary state names, 1 per record, along with all corresponding abbreviations.

A **list** shows the user all possible abbreviations within the system. When a user asks for a list of state codes, for example, all state names will appear in a tool called a **PickList**. The user can then 'pick' a particular state and have the abbreviation appear in the appropriate field. PickLists shorten the 'learning curve', by giving the user instant access to the full list of abbreviations.

Another type of list is the **BrowseList**, which allows the user to add new abbreviations and meanings to an existing data dictionary. BrowseLists empower the user to expand the abbreviation lists while doing data entry.

VariForm Improves on Data Dictionaries

In VariForm, any database may be used as a data dictionary. This reduces the amount of specialized or redundant data-entry: Reference information only needs to be entered one time, and all databases can have access to it. This concept also permits data dictionary expansion while the system is in use, eliminating the manpower requirements and delays involved in 'updating internal tables' during normal business operations.

Getting Started

Beyond Flat Files: Relationships

Information about a topic may be kept in more than one physical database, either for user convenience or for economy of space. Database relationships provide a definition of how information is retrieved 'as a whole' from separate physical databases. The difference between a 'flat file' database and a 'relational' database is the ability to support and enforce inter-database relationships.

There are two general types of relationships: automatic and manual. An Automatic relationship is enforced by VariForm, and always exists. A Manual relationship is enforced by data entry, and is enforced by a separate procedure called an "Active Query".

Databases within a relationship may be termed a 'parent', 'child', or 'brother'. These terms refer to the database's position within the relationship.

Parent Database: The database that has control. In an automatic relationship, if a parent record is deleted, all associated child records are deleted as well.

Child Database: The database that is controlled. If a child record is deleted, usually the associated record in the parent database is unaffected.

Brother/Peer: A non-parent, non-child database, where the records are generally not affected by operations on other databases, and which do not affect other databases in the relationship. Data dictionaries, for example, can be considered as brothers to all databases that refer to them.

Records within a parent, child, or brother database are organized by specialized fields within each record.

Record Group: An association of records that have something in common, such as "all client records in Pennsylvania". Groups are useful for assembling information into useful categories for manipulation or reporting.

Group Field: A field (or group of fields) within a record that define the record's group or cluster identity. All records in the same 'group' will have the same contents in their group field(s).

Binding Field: Classification of a field (or group of fields) within a record that identify the record's identity within the relationship. The record may be either an individual, or part of a group of records. All fields used in the definition of a relationship are classified as 'binding fields'.

Unique Field: A field that contains a totally unique value within either all records in a database, such as "client id number", or all records within a group, such as "Pennsylvania client id number". Unique fields can consist of any field type(s). The only requirement is that the database containing the unique field has an index that sorts the unique field or its group.

Getting Started

Linear Relationship Types

There are four kinds of basic relationships: one to one (direct), one to many (parent/children), many to one (inverse or aggregate), and many to many (mushy).

One to One Records in two databases have a direct and unique correspondence, such as "Client Address", and "Miscellaneous Client Information", where there is 1 record of information for each client address. Usually both databases have a binding field that is unique within the database, and equivalent to a similar binding field in the other database.

One to Many A record in one (parent) database has a direct and unique correspondence to one or more records in another (child) database, such as "Client Address", and "Client Purchasing History", where there may be several records of purchases for a particular client. Usually the parent database has a unique binding field that corresponds to a non-unique "group field" in the child database.

Many to One Several records in one (parent/brother) database have a direct but non-unique correspondence to one record in another (child/family) database. This is another expression of Record Grouping, where the parent database would be "Client Address", and the child/family database would be "States with clients in them". This is an upside-down relationship, where the parent database has a non-unique group field that corresponds to a unique binding field in the child database.

Many to Many Also called a 'mushy' relationship: several records (a record group) in one database have a direct but non-uniqe correspondence to several records (record group) in another database. A better name for this style of relationship would be peer-group related records. In many to many relationships, there is no 'parent' or 'child', but only related associations of records between peer-databases. One brother could be "Client Address", and the other might be "Counties in a state with clients in them", where the common bond between the two databases is state name. These are cases where relationships are designed to connect to non-unique group fields between two databases. In VariForm, many to many relationships are often called 'unenforced references', or 'groups'.

Getting Started

MultiDimensional Relationships

Flat File databases which treat records as computerized "filing cards", which contain **static information**.

Lookup and Data Dictionary systems allow database records to be smaller and more easily entered through a system of abbreviations and corresponding definitions of meanings. Lookups make flat file **records more meaningful**.

"Relational" databases make the flat file definition **dynamic** by grouping and associating records between databases. A record in a parent database controls several records in a child database. The 'child records' may be parents to records in another database. Each relationship requires access to another database, **so the "dynamic" databases are defined in static relationships.**

A MultiDimensional Relationship is a dynamic relationship system: a database can contain both parent and child records to the same relationship. **This provides a more effective use of limited resources**, such as disk space and data entry time, **and provides more capabilities within any functioning system**. MultiDimensional Relationships three unique characteristics:

- **Self-Relating** : A database which contains as both parent and child records in a relationship. Applications can use one self-related database to do the work of hundreds of linear-relationship databases. A good example of a self-relating database can be found in the HelpTopic database description later in this manual. A single database, related to itself, provides a near-infinite depth categorization system using only 4 fields and one multidimensional relationship.

- **Recursive Referencing** : A pair of databases whose records reference each other. Applications can use recursive referencing between two databases to produce many different views of the same information without altering the information. A good example of recursive referencing can be found in the Display System definitions found later in this manual. A pair of databases, relating back and forth, define a near-infinite depth and breadth display system for multilevel data entry and report generation.

- **Complex Referencing** : A parent record may have several different relationships with groups of child records at the same time, and those relationships may partially intersect each other.

VariForm Use Terminology

Menu System

VariForm is a system of menus and data-entry screens. When the program starts, the user will be provided with a list of choices on a Menu. These choices will include running a VariForm Application, entering VariForm System, or Quitting to Dos. Please note that "Quitting to Dos" means leaving VariForm and returning to whatever the computer was doing before it started VariForm. Which may be DOS, Windows, or some other type of shell program.

It is also important to note the cue lines at the bottom of the screen. At the beginning of the program, start-up messages, such as "Loading ... database" will appear there. Once the Main Menu appears, these bottom 5 lines on the screen contain information about what menu choices do, what function keys are available at the present time, and what actions VariForm is performing at the current moment.

When looking at a menu, the user is presented with a series of choices, which may be selected in one of several ways:

All menus permit Arrow Keys to move to a selection and <Enter> to select a function:

If the menu is Horizontal (choices are all on 1 line), left and right arrows move between selections, and <Down Arrow> works the same as the <Enter> key.

If the menu is Vertical (choices are in a column), <Up Arrow> and <Down Arrow> move between selections; <Left Arrow> and <Right Arrow> will move to the next vertical menu.

Menus may also have one character of a selection in a different color from the rest of the selection text. That character is called a hot key, which means that the selection can be made simply by pressing that letter. This can save time when compared to using the arrow keys and pressing <Enter>.

Most menus also have some function keys listed at the bottom of the screen, such as "<F1>-Help <sF3>-CommandMode". This means that pressing the <F1> key will call help, and pressing <Shift-F3> will jump to the command mode.

At any point within VariForm, pressing a non-existant function key makes VariForm offer to provide a list of the currently available function keys to the user. So if the help/cue-line isn't there or isn't comprehensible, press a strange function key combination, such as <Control-F10> to get a list of function keys.

<Escape> will abandon any menu except the Main Menu, where the "Quit" option will be highlighted. You must press <Enter> on Quit to leave VariForm. This is to prevent accident program exits when the user hits <Escape> one too many times.

VariForm Use Terminology

Browsers

In general, Menu selections that access a database will call a Browser, which is a vertical list of records within a database. When first entering a browser, the topmost record in the window will be a different color than the rest of the records. This coloring, called the Cursor Bar, tells the user what record will be directly affected by a command at that time, and is simply a cursor.

Browsers allow users to look at information in a database in any order, search for particular records, and allow them to be modified. All browsers have the following basic keys built in:

<**Up Arrow**/**Down Arrow**>: move to the previous or next record in the browser. This is shown by the cursor bar moving on the screen.

<**Enter**>: edit the current record.

<**Page Up**/**Page Down**>: move one screenful previous or next in the database, if possible, or go to the first or last record in the database.

<**Home**/**End**>: go to the first or last record in the database

<**Escape**>: quit the browser: return to the prior menu or screen

(**Alphanumeric keys**): search for the first record that matches the typed-in character(s) in the first field of the current index (see below).

When searching for a particular record, typing in the appropriate field contents will cause the browser to "**Search Towards**" a particular record. The browser will find the first record that has the typed character(s) or greater as contents or the first field in the index. The search record will display on the top line of the browser with the field being searched changing color to match the number of characters typed. Any movement key, such as <Home>, <End>, <Page> or <Arrow> will 'clear the search'.

'First', 'last', 'previous', and next are relative terms, and refer to the records in a database as they are sorted according to a database index. The index order which is displaying, such as "alphabetical by name", or "numerical by zip code", is called the browser's current index order.

There are actually two modes of browser searching, which are controlled by a system Virtual Field Called "SearchExact". If SearchExact is 0 (default), the browser will select the first record with a field greater than or equal to the typed in value; If SearchExact is 1, the browser will 'beep' and not move if an exact starting match is not found.

VariForm Use Terminology

Browsers

Most browsers also have function keys to facilitate changing database information. The following list has function keys listed in parentheses. These are system defaults, and may be changed by a programmer, developer or an end user.

(**<F1>**) call **help** for the browser.

(**<F3>**) provide a data entry screen to **add** a new record.

(**<F4>**) **delete** current record, and all associated child records.

(**<F5>**) **copy** current record. If there are any unique fields, increment their values to maintain their uniqueness. If there are any child databases, copy the child records to become children of the new parent record.

(**<F6>**) **change current index order** (a database may have several indexes: the "NextIndex" key walks through the indexes in round-robin fashion.

(**<F7>**) **select index**: generate a list of indexes for the user to more easily select with index to use within the browser

(**<F8>**) **entry mode**: switch the browser to data-entry mode, allowing the user to change any displayed field on any record on the browser. See the section below for more information about the Browser Entry Mode.

(**<F9>**) **database info**: display information on the number and kind of records in the database being browsed.

(**<F10>**) **pack database**: rebuild the database, removing all deleted records, and rebuild all indexes from scratch. The original data file (*.dat) is kept and renamed to Backup (*.bak) upon successful completion of a pack.

MultiUser Warning: do not pack a database in multiuser mode if any other user has that database open! All records will be removed from the database if you do so. Also, keep in mind that automatically-opened databases in a shared directory have the same problem if more than one user is in VariForm.

(**<sF2>**) **KillSafe Toggle**: KillSafe is a system variable which toggle turns off or on. The default for VariForm is KillSafe:on (1). When KillSafe is on, and a user presses the 'delete' key, the record will appear in a data entry screen, and the user will be asked to enter 'y' to confirm the deletion of that record. With KillSafe off, the deletion happens immediately, with no confirmation or display of the target record.

(**<cF5>**) **Flush Files**: For single-user mode (OnNetwork = 0), Flushing files forces VariForm to save all changes made to all Open Indexes to the disk. This is especially useful just before testing a new feature: if the system locks or needs 're-booted', the indexes will not be corrupted if they were flushed before the reboot. When OnNetwork = 1, Flushing occurs automatically, so that other users on the system can see changes made to any shared database immediately.

(**<cF6>**) **Flush Memory**: VariForm tries to keep as much information in memory as possible. This includes browsers, data entry screens, keys, colors, searches and procedures recently used, so they do not have to be constantly reloaded from the disk. Flushing memory will take any currently unused elements out of memory, making room for more data entry screens and/or procedures. Generally, this key is used as a reaction to "Out of Memory" errors (message 320).

Browser Entry Mode

Sometimes, a user needs to see several records together to make meaningful changes. VariForm answers this need with Browser Entry Mode. The system default key (DefaultBrowser) to access entry mode is <F8>, and will work on any browser where an "EntryMode()" function key is available.

In Entry Mode, the Browser changes color, and the Browser Bar becomes a data-entry screen for the current browser record. Search Mode is disabled, and <Page Up> and <Page Down> move the cursor within the browser bar, instead of jumping up or down a browser-screenful of records. <Enter> simply advances to the next editable field on the browser bar. <Escape> aborts any changes made to the current record, and returns the user to standard Browser Mode. <Ctrl-Enter> saves any changes made to the current record and returs the user to standard Browser Mode.

The user may have a different set of function keys in Entry Mode, as compared to standard Browsing, and those keys may vary from Field to Field, just as in a regular data entry screen. The cue lines will change to reflect this.

There are two major differences between Browser Entry Mode and Data Entry Screens: "Arrowing off" the record will force changes to be saved, and fields that were shortened for the browser may be truncated if the record is saved. "Arrowing off" a record means pressing the <Up Arrow> key from the top line of the Browser Bar, or <Down Arrow> from the bottom line of the Browser Bar. On Data Entry Screens, these same actions would simply jump the user to the bottom or top of the Screen they were on.

One other suggestion about Browser Entry Mode: When putting records in order, do not edit fields in Browser's Current Index order. This will do no damage, but may cause confusion if the user attempts to scroll the browser and sees a record "still on the screen" appearing again, simply because it is next in the index, since the edits made on the screen have already taken place.

VariForm Use Terminology

Data Entry

The most common place for a VariForm user to be is the Data Entry system. Technically, all systems: Browsers, Reports, Menus, PickLists, are all simply special cases of the Display System. For convenience, "Data Entry" will refer to editing a single record's contents within a window, and is representative of both Requestors and Add/Edit mode from a standard Browser.

All Data Entry Screens contain some combination of six elements:

The Cursor The little flashing box on the screen. This tells the user where exactly the next key typed will be placed. The cursor has two different widths, which may be toggled by the <Insert> key. When the cursor is "Thick", characters will be inserted, pushing text to the right; when "Thin", characters will overwrite existing text.

Field Contents This is the information in the database, such as a client's name or address. The field that the user is on is called the "Current Field", and will be a different color from all other fields on the screen.

Prompts These are the words that describe the field, such as "Name: " or "Address: ". It helps the user to know the significance of the current field. Each field may have 2 prompts, allowing for complex, matrix-like displays to light up appropriate rows and columns to tell the user where the cursor is.

Lookups Lookups are used when the field contents are an abbreviation, so the user knows the meaning of the associated field. The lookup will appear as off-color text to the right of the field, and will change when a field is changed and left, when a picklist or browselist entry is selected, or whenever a 'redraw' function key is pressed.

Display Text This is unchanging text, such as the window title, or boxes or lines that help 'break up' a data entry window into separate pieces.

Nested Fields A field the displays Press Enter when current. Nested Fields give the user access to some other information, such as Subscreens or Child Databases.

In many cases, there is more information in a data entry screen than will display neatly on the computer screen at one time. This leads to two separate terms about data entry:

A **Data Entry Screen** is the entire data-entry area for a record, and may be hundreds of lines long.

A **Data Entry Window** is the area on the computer's display that the user can see. Often, the Window will have a marker in the lower right-hand corner showing that there is more screen above, below, or surrounding the Window's current position.

VariForm Use Terminology

Data Entry

All Data Entry Screens share certain global **Positioning and Editing Keys**:

<**Insert**> Toggle between "Insert" and "OverWrite" mode (see cursor, above)

<**Delete**> Delete the character that the cursor is on, pulling text right of the cursor left 1 character

<**Ctrl-T**> Delete all characters from the cursor to the beginning of the next word to the right of the cursor

<**Ctrl-End**> Delete all characters from the cursor to the end of the field

<**Ctrl-Home**> Delete all characters from the cursor to the beginning of the field, move cursor and all remaining text to the beginning of the field

<**Ctrl-U**> Undo all changes made since entering the field (this does not affect changes made by picklists, browselists, or procedures)

<**Home**> Move to the beginning of the field

<**End**> Move to the end of typed characters in the field

<**Left Arrow**> Move the cursor 1 character to the left within the field. <Left Arrow> will not leave the current field.

<**Right Arrow**> Move the cursor 1 character to the right within the field. <Right Arrow> will not leave the current field.

<**Ctrl-Right**> Jump to the next word beginning to the right of the cursor

<**Ctrl-Left**> Jump to the previous word beginning to the left of the cursor

<**Enter**> Move to the next field, to the right or downward, whichever comes first. This follows a 'Z' pattern, like reading a book. Exception: Nested Fields: if field displays the words 'Press Enter' when current, the <Enter> key will call up a subscreen or subdatabase browser. To leave the field, the user must use <Tab>, <Page>, or <Arrow> keys.

<**Ctrl-Enter**> Save all changes to the record and leave the data entry screen

<**Escape**> Abandon all changes to the record and leave the data entry screen. If the user made changes to the screen, or called a picklist or browselist, the user will be asked to confirm: 'Abandon All Changes', where 'No', <Enter>, or <Escape> will all return the user to the screen, preventing accidental loss of data entry. The user must respond 'Yes' to abandon the changes.

VariForm Use Terminology

Data Entry

<**Tab**> As <Enter>, move to the next field

<**Shift-Tab**> Move to the previous field, left or upward, whichever comes first. This is the reverse of the 'Z' pattern of <Tab> and <Enter>

<**Up Arrow**> Move to the field directly above the current field, if one exists, or above and to the left. <Up Arrow> on any field on the top line of the Data Entry Screen will jump the cursor to a field on the bottom line of the Data Entry Screen.

<**Down Arrow**> Move to the field directly below the current field, if one exists, or down and to the left. <Down Arrow> on any field on the bottom line of the Data Entry Screen will jump the cursor to a field on the top line of the Data Entry Screen.

<**Page Up**> Move up one 'window-full', for screens that are larger than the window. Move to the first field otherwise.

<**Page Down**> Move down one 'window-full', for screens that are larger than the window. Move to the last field otherwise.

<**Ctrl-Page Up**> Move to the first field of the screen

<**Ctrl-Page Dn**> Move to the last field of the screen

Most Data Entry Screens also have function keys to facilitate Data Entry. The following list has function keys listed parenthetically, because these are system defaults, and may be changed by a programmer, developer or an end user:

(**<F1>**) Call System **Help**: if there is help specific to the Current Field, call it, otherwise call help specific to the Data Entry Screen, otherwise call help for "Data Entry".

(**<F5>**) **Redraw** the Data Entry Screen. If several lookups are all dependant on one field, <F5> can make sure all of the lookups redisplay properly.

(**<cF6>**) **Flush Memory**: VariForm tries to keep as much information in memory as possible. This includes browsers, data entry screens, keys, colors, searches and procedures recently used, so they do not have to be constantly reloaded from the disk. Flushing memory will take any currently unused elements out of memory, making room for more data entry screens and/or procedures. Generally, this key is used as a reaction to "Out of Memory" errors (message 320).

(**<sF1>**) **Memory Window**: Toggle the display of available memory. This little window can help the user to anticipate if memory needs flushed, especially on data entry screens with several complex procedure calls.

Primary Data Structures

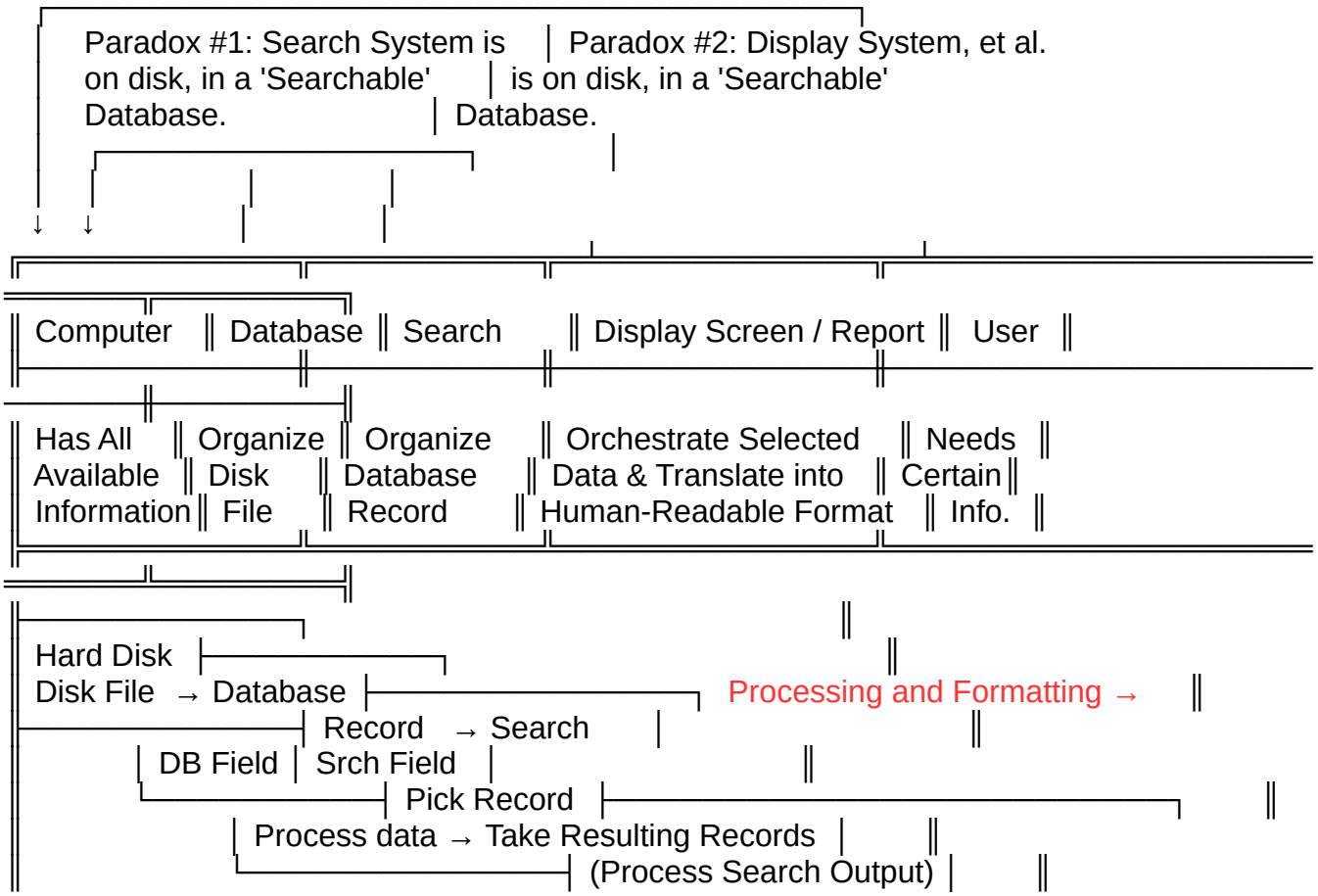
General Principle

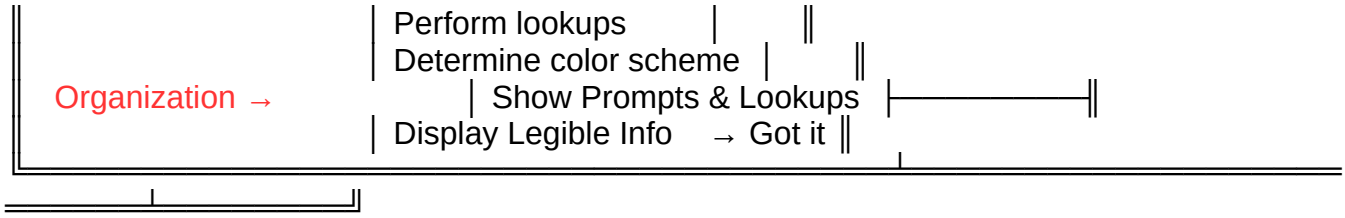
VariForm is a multi-purpose information system management environment that combines elements of CASE Tools, Relational Database Management Systems, and a procedural language to provide a unique, sophisticated Application Development Environment for both software developers and end users.

VariForm consists of a series of interactive data structures. Data is stored in databases, which describe information on computer media. These databases are accessed by Display Structures, which describe the transient presentation of the information. The Display Structures are controlled by Search Structures, that limit or expand the breadth and depth of the information to be presented.

Since the descriptions of information is also information, it too must be stored, and that storage must be described. This leads to a **Cyclical Definition of Information Systems: all descriptions must be described**. To resolve this paradox, VariForm uses three minimum data structure pairs. They are called the Database System, Display System, and the Search System. Understanding these pairs and their interactions will help the reader to succesfully explore and use the system to the best of its capability.

Information Systems become cyclical when output controls are flexible. VariForm provides Modifiable Data Entry Screens that Control Data Entry, Search Systems, and Databases.





Primary Data Structures

Structure Pairs, in brief

Major structures are generally paired, to help reduce the sheer volume of information required to make the system work. Pairs are divided into a Major and Minor element, with the minor element called 'Fields'.

Database System: The first data pair describes how the computer stores information on media, such as a floppy disc or hard drive. The major element is called a Database, and the minor elements are called Database Fields. The Database describes general information, such as the file name to use, the precise location of the file, and some basic security information. The Database Fields describe a record within the database one Field at a time. Each Field has a name, size, a location within the database's record, and some form of information in it.

Database Systems can be viewed by the Display System, and limited or modified by the Search System.

Note: The Data System also contains references to how the information is sorted and grouped. In the purest sense, Indexes and Groups are separate databases that organize structures defined by the data system.

Display System: The second data pair describes how information is presented to the user. These descriptions are used to generate menus, browsers, picklists, data entry systems and reports. The major element is called a Display Structure, and the minor elements are called Display Fields. Display Structures describe how information is presented: as a menu, data entry screen, data browser, report, et cetera. Display Fields describe what the user will see, such as a particular field on a certain location on the display screen. Display Fields should not be confused with Database Fields: Display Fields may contain transient (calculated) information, and may perform specific actions for the user, as well as display specific Database Field Information.

The Display System allows different views (Various Forms = VariForm) on Database System Information, and can be limited or expanded by the Search System.

Search System: The final data pair describes limits and interactions between databases and display systems. These descriptions can connect databases together, or limit access to only relevant portions of a database, as well as modify the information available. The major element is called a Search Structure, and the minor elements are called Search Field Records. Search Structures describe what database or databases are connected by a particular junction, and in what order to access the dependant database. Search Field Records describe what interactions take place: which records in a database to use, and what to do with them.

The Search System controls what data is allowed to be seen or modified at a given time, and provides temporary or permanent database relationships, inter-database record mapping, and calculations for reports. Searches can call other searches, can run procedures, and can be called by procedures and the Command Line Interface.

Primary Data Structures

Data System Detail

Database of Databases

Name	Size	Off.	Type	Type Definition	Sub-Type
Name	22	0	0	Char / ASCIIZ String	2
FileMask	160	22	0	Char / ASCIIZ String	0
RamBuffers	2	182	2	Integer	0
BufferSize	2	184	2	Integer	0
EncryptionStyle	2	186	2	Integer	0
EncryptionKey	22	188	0	Char / ASCIIZ String	0
Characteristics	2	210	2	Integer	0
TreeDepth	2	212	2	Integer	0
ReadSecurity	2	214	2	Integer	0
AddSecurity	2	216	2	Integer	0
EditSecurity	2	218	2	Integer	0
DeleteSecurity	2	220	2	Integer	0
StructureSecurity	2	222	2	Integer	0

Data Systems (Databases) are described in the "Database of Databases". The primary record describes the database-level security, the buffer sizes to use on the database's indexes, and some characteristic information about the database.

Name A unique name to reference this database.

File Mask A (usually) unique name that describes the file in which the information will be stored.

Encryption Style Encryption is an optional means of making the information in the database difficult to read without the VariForm program. The Encryption Style field tells VariForm what formulas to use to encrypt the databases.

Style	Name	Meaning
0	None	No Encryption. Fastest Record Access.
1	Crypt	2-key complex encryption. Sophisticated enought to keep a 'hack' style codebreaker running for 4-6 years before getting 100% of the information.

Primary Data Structures

2	DES (separate module)	Federal (NIST-defined) Data Encryption Standard. Slow. Can be certified for 'Sensitive' information for Defense Department Work, sufficient for 2-B security clearance.
3	Crypt2 (separate module)	Multi-key, multi-pass encryption. Very slow. More complex than DES encoding, and private, making 90% of all code-breakers obsolete. "Brute Force" method would require at least 6 Cray Years to cover 30% of all possibilities.

Primary Data Structures

Data System Detail

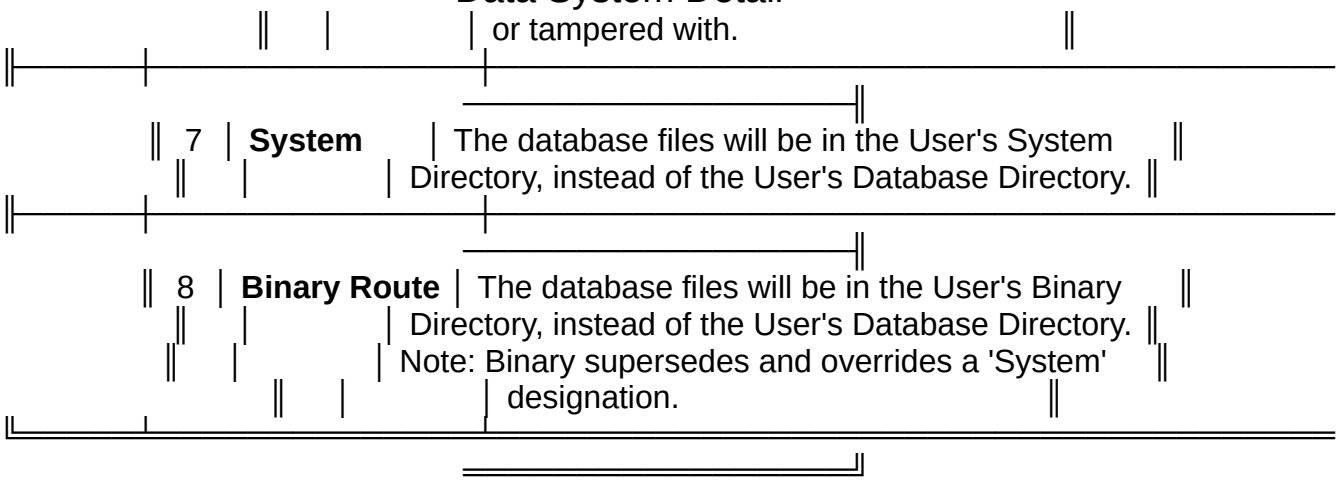
Databases

Encryption Key This is a 'keycode initializer', that helps the encryption system to encode the database records. This further secures the system, since each database may have its own unique initializer.

Character's unique features about the database, such as foreign characters, or non-standard storage methods, such as 'no trailing crc'. This is actually a bit field, broken into a series of Yes/No flags by the Data Entry Screen.

Database Characteristic Bits

Bit	Name	Meaning if Set (On. Default is off.)
0	**AutoLoad**	Database will be opened when VariForm is started.
1	**Sticky**	Database will not close, once opened, until VariForm is exiting.
2	**VariStruct** (separate module)	For use by specialized (AI) applications. 2-file data system permits Variable-Sized Records based on "Size Controller" and "Size Type" Fields in records.
3	**Forward Ints**	Numeric fields are stored as 'Motorola Format', with the most significant byte first. Default (off) stores numerics (ints, long ints, fixed dec, and dates) in 'Intel Format', with the least significant byte (and or word) first.
4	**BCD Floats** (separate module)	All 'Floating Point' fields employ a Binary Coded Decimal format, instead of the IEEE 64-bit floating point.
5	**EBCDC**	All character fields are stored in EBCDC format, instead of ASCII. This is most commonly used when working with files on or for an IBM MainFrame.
6	**No CRC**	Records will not contain a crc-16 code. CRC's add 2 bytes to each record, but provide an automatic method of detecting that records have been damaged



Primary Data Structures

Data System Detail

Databases

Security A series of numbers that can limit user access, or limit a users ability to change information on this database. These fields have no significance unless one of the Security Pack Option modules is installed. If the module is installed, there will be a field in the Main group of Virtual Fields called "SecureStyle". If SecureStyle is 1, the user must have a security number greater than or equal to the security fields listed below for permission to the associated functions. If SecureStyle is 2, the user's security is treated as 16 bit fields.

Security Fields

Name	Associated functions allowed (see above)
Read	User may read records: reporting, browsing, open-entry mode allowed, menus and data-entry screen accesses permitting.
Add	User may add records to the database. This does not require read access.
Edit	User may change existing records in the database. This requires read access.
Delete	User may delete existing records. This requires read access.
Structure	User may see the structure record in the database of databases.

Buffer Size For Indexing: size of an index buffer record. Minimum size is 128 bytes, maximum size is 16,384 bytes. While bigger is generally better, this can rapidly consume system resources (RAM).

Ram Buffers Number of Index Buffers to load for this database. This data buffering helps speed up database access by keeping information on many indexes in memory at once. Optimally, there should be at least one buffer per database index, unless in multiuser mode. Multiuser mode (OnNetwork = 1) defeats the caching system, so use the minimum value: 2. Maximum allowed value is 32, and larger numbers will be ignored, defaulting to 2 buffers per database.

Tree Depth For Indexing: VariForm indexes use a 'Binary Tree' method to efficiently sort records. This number describes the maximum limit to the tree's depth. This number limits the number of records that can be sorted within a database. Generally, the

value 11 is large enough for most purposes, but any number over 3 can be used.

Tuning the Database Buffers

There is a fairly direct trade-off between buffer sizing and response time, as the database grows larger. There is a precisely direct tradeoff between buffer sizing and free RAM. Balancing these two trade-offs (speed versus free space), which work against each other, requires a little bit of math:

◦ When first defining a database, select some standard numbers, such as 2 buffers at 1024 bytes each, that store 11 levels. Use these numbers to check the database and the indexes.

◦ After the database structures (fields and indexes) have been filled in, and the database has been opened once (to check field aligments, et cetera), then it is time for some quick analysis.

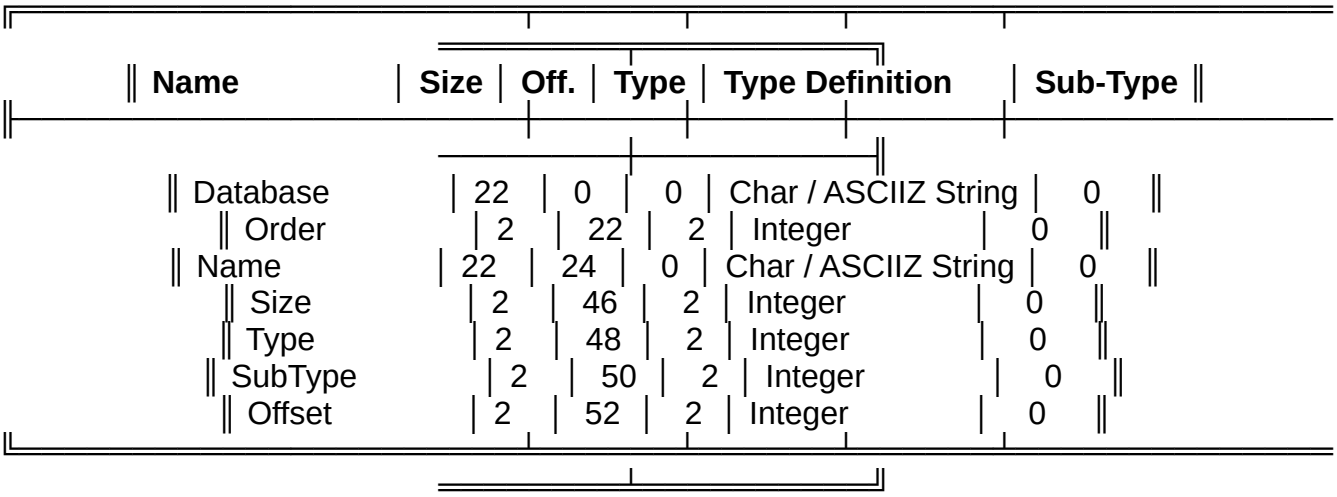
◦ Change the number of buffers to the number of indexes plus one. Open the database (browse it). If Free RAM is below 50,000 bytes, memory is getting tight, and you may want/need to consider 'shrinking the index'. If Free RAM is above 150,000 bytes and this is the lowest (most-related: youngest child) database, you should consider 'growing the index'.

To 'grow' or 'shrink' indexes safely, see 'Index Calculations' in Appendix 1, page .

Primary Data Structures

Data System Detail

Database of Fields



Name	Size	Off.	Type	Type Definition	Sub-Type
Database	22	0	0	Char / ASCIIZ String	0
Order	2	22	2	Integer	0
Name	22	24	0	Char / ASCIIZ String	0
Size	2	46	2	Integer	0
Type	2	48	2	Integer	0
SubType	2	50	2	Integer	0
Offset	2	52	2	Integer	0

The primary (mandatory) Detail information describes the field system unique to this database. This information is stored in the "Database of Fields", which contains the following information for each field:

Data Name the name of the database this field is attached to.

Field Name the unique name for this field within this database.

Type the format, or how the information is stored within a record.

Sub Type more detailed information about the field type in a bit flag. Because these are bit settings, certain combinations should be avoided, such as unique + group unique (which is redundant), and union + unique (which is inconsistant).

Size the size of the field in (8-bit) bytes. It is overridden by mandatory sizes associated with the field type (see above), so is only significant in type 0/1 (string) fields.

Offset the position of the field within the database record. The first position is always 0. This is how union fields are 'co-located'.

Primary Data Structures

Data System Detail

Database Field Types

#	Field Type	Size	Description
0	**String**	<301	Character string with a NULL (char 0) terminator, also called the ASCIIZ (ASCII-Zero) format. Size field determines length.
1	**Len. Lead.**	<301	(Length-Leader) A string with the length in first byte.
2	**Integer**	2	16-bit whole numeric: values 0-65535, or -32768-32767.
3	**Long Int**	4	32-bit whole numeric: values 0-4 / -2 - 2 billion.
4	**Float**	8	64-bit IEEE floating point number, 15 digit precision.
5	**Fixed-Len**	4	A long int using sub-type as power of 10 divisor: not as detailed or as fast as a floating point number, but smaller (4 bytes vs. 8), and accurate.
6	**BCD**	?	Binary Coded Decimal: not yet implemented.
7	**Date**	4	Date format: 4 bytes: Year int, Month, Day in bytes.
8	**Time**	4	Time field, in milliseconds, stored as a long int.

Database Field Sub Type Bits

Bit	Type	Meaning
0	**Signed**	Numerics are treated as signed, strings are case-sensitive and permit trailing spaces to have significance.

Primary Data Structures

Data System Detail

1	**Unique**	This field type may not have a duplicate within any other record of this database. To enforce this, the database must contain an index of just this field.
2	**Union**	This field type has no effect on the database for size and is not checked for positioning. Union fields should not be used in indexes, because they will produce inconsistant results. For best results, place the first field of the union in as an absolute field, to reserve the space in the record, and make all subsequent uses of the same record area union field entries.
3	**Group Unique**	This field must be unique within a group of records. This field type must have a position in an index that defines all group-significant fields before itself.

Primary Data Structures

Data System Detail

Database of Indexes

Name	Size	Off.	Type	Type Definition	Sub-Type
Database	22	0	0	Char / ASCIIZ String	0
Order	2	22	2	Integer	0
Index	22	24	0	Char / ASCIIZ String	0
Characteristics	2	46	2	Integer	0
Field0Name	22	48	0	Char / ASCIIZ String	0
Field0Size	2	70	2	Integer	0
Field0Dir	2	72	2	Integer	0
Field1Name	22	74	0	Char / ASCIIZ String	0
Field1Size	2	96	2	Integer	0
Field1Dir	2	98	2	Integer	0
Field2Name	22	100	0	Char / ASCIIZ String	0
Field2Size	2	122	2	Integer	0
Field2Dir	2	124	2	Integer	0
Field3Name	22	126	0	Char / ASCIIZ String	0
Field3Size	2	148	2	Integer	0
Field3Dir	2	150	2	Integer	0
Field4Name	22	152	0	Char / ASCIIZ String	0
Field4Size	2	174	2	Integer	0
Field4Dir	2	176	2	Integer	0
Field5Name	22	178	0	Char / ASCIIZ String	0
Field5Size	2	200	2	Integer	0
Field5Dir	2	202	2	Integer	0
Field6Name	22	204	0	Char / ASCIIZ String	0
Field6Size	2	226	2	Integer	0
Field6Dir	2	228	2	Integer	0
Field7Name	22	230	0	Char / ASCIIZ String	0
Field7Size	2	252	2	Integer	0
Field7Dir	2	254	2	Integer	0
Field8Name	22	256	0	Char / ASCIIZ String	0
Field8Size	2	278	2	Integer	0
Field8Dir	2	280	2	Integer	0
Field9Name	22	282	0	Char / ASCIIZ String	0
Field9Size	2	304	2	Integer	0
Field9Dir	2	306	2	Integer	0

The secondary (optional) Detail information describes automatic sorting that is performed within the database. The "Database of Indexes" stores this information, and facilitates reporting and organizing information by maintaining the order of records within a database. When misused, Indexes can become the #1 cause of performance degradation.

Database The name of the database this index refers to.

Order A number used to organize indexes within a database.

Index Name The unique name for this index within this database

Character The direction (ascending, descending, or complex) of the index. This field is for future use, and is ignored now.

Field#Name Name of a database field to use in the index. It is not recommended to use unioned fields as index elements, since their value may be insignificant at certain times.

Field#Size Number of significant (8-bit) bytes to use from the field when sorting. Field default sizes override this entry, except for character string (type 0, 1) fields.

Field#Dir For future use: direction (0 = ascending, 1=descending) in which to sort this field within the index.

To understand how the index sorts elements, think of an index entry as a virtual field that is made up of all of the fields listed in this record, padded with NULLs or truncated to the corresponding size, with an extra terminal field of Physical Record Number, size 4 (long int). That would describe the 'key field' that is attached to each record in the database for index-sorting purposes.

Primary Data Structures

Display System Detail

Display Database

Name	Size	Off.	Type	Type Definition	Sub-Type
Name	22	0	0	Char / ASCIIZ String	2
Database	22	22	0	Char / ASCIIZ String	0
Display	22	44	0	Char / ASCIIZ String	0
Help	22	66	0	Char / ASCIIZ String	0
Title	80	88	0	Char / ASCIIZ String	0
DEScreen	120	168	0	Char / ASCIIZ String	0
Footer	160	288	0	Char / ASCIIZ String	0
FooterOffset	2	448	2	Integer	0
FooterColors	22	450	0	Char / ASCIIZ String	0
Row	2	472	2	Integer	0
Column	2	474	2	Integer	0
Height	2	476	2	Integer	0
Width	2	478	2	Integer	0
ItemHeight	2	480	2	Integer	0
ColorScheme	22	482	0	Char / ASCIIZ String	0
TrueHeight	2	504	2	Integer	0
TrueWidth	2	506	2	Integer	0
KeyName	22	508	0	Char / ASCIIZ String	0
Usage	2	530	2	Integer	0
BrowserHelp	22	532	0	Char / ASCIIZ String	0
BrowserKeys	22	554	0	Char / ASCIIZ String	0
BrowserColors	22	576	0	Char / ASCIIZ String	0
BrowserFooter	160	598	0	Char / ASCIIZ String	0
SearchIndex	22	758	0	Char / ASCIIZ String	0
DisplayLimits	22	780	0	Char / ASCIIZ String	0
Security	2	802	2	Integer	0

The Display System performs a variety of functions to convert data from computer-usable to user-comprehensible formats. To do this, there are several different types of the Overview record. The Overview is stored in the Display Structures Database.

Name A unique name to reference this Display System

Database Name of the primary database used in this display. Name may be 'Virtual', to reference the internal variables.

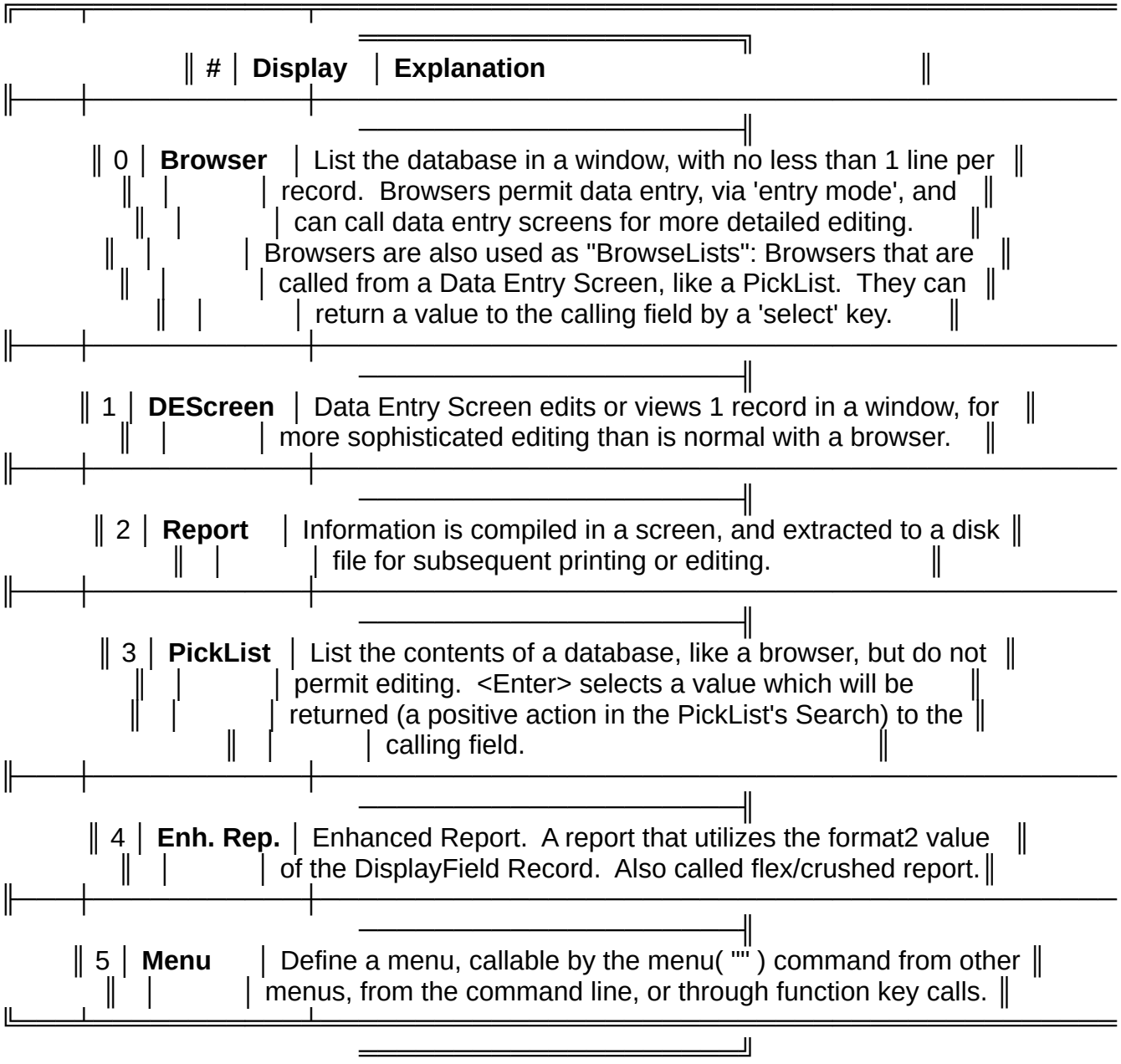
Type Type of Display System: what the output will do (see below)

Primary Data Structures

Display System Detail

Display Database

Types of Display System Records



#	Display	Explanation
0	**Browser**	List the database in a window, with no less than 1 line per record. Browsers permit data entry, via 'entry mode', and can call data entry screens for more detailed editing. Browsers are also used as "BrowseLists": Browsers that are called from a Data Entry Screen, like a PickList. They can return a value to the calling field by a 'select' key.
1	**DEScreen**	Data Entry Screen edits or views 1 record in a window, for more sophisticated editing than is normal with a browser.
2	**Report**	Information is compiled in a screen, and extracted to a disk file for subsequent printing or editing.
3	**PickList**	List the contents of a database, like a browser, but do not permit editing. <Enter> selects a value which will be returned (a positive action in the PickList's Search) to the calling field.
4	**Enh. Rep.**	Enhanced Report. A report that utilizes the format2 value of the DisplayField Record. Also called flex/crushed report.
5	**Menu**	Define a menu, callable by the menu("") command from other menus, from the command line, or through function key calls.

Group Name shared by the Display Fields associated with this Display Structure; an indirect relationship, so that many Display Structures can share the same Display Fields.

DE Help Help text to call when help key is pressed in Data Entry mode

Row, Col Define the position on the screen, or offset on the page where this

information should be displayed.

Wid, Height Define the size on the screen of the displayed information.

True "" Define the actual size of the window in memory for editing, or the maximum size of the subwindow on the report.

HelpLine 2 lines of Help text that Display on the Bottom of the screen when this Data Entry Screen is used.

Help Offset Offset from line 5 to start drawing the HelpLine

Help Colors Color system to use for the Help Area

Item Height Browsers, PickLists, subreports: maximum size of 1 record in rows.

DE Colors Color Scheme to use when in data entry mode

Func Keys Function Key set to 'attach' when in Data Entry mode

Brow Col's Color Scheme to use when in Browse/Pick mode

Brow Keys Function Key set to attach when in Browse/Pick mode

Brow Text Help text to display in Browse/Pick mode. Also pre-report commandline to call in Report mode.

Brow Help Help text to call when help key is pressed in Browse/Pick mode

SearchIdx Name of Search Record defining Limited/Mapped/Selected conditions for Browsers, BrowseLists, PickLists, Reports, and sub-report modes.

DisplayLim. Group name of Limiter Procedure list to use to determine pre/post add, edit, delete, and copy calls.

Security A numeric value indicating the minimum security required to access this display. Users with lower security will be told 'access denied' if they attempt to bring up this screen for use.

Primary Data Structures

Display System Detail

Display Fields

Name	Size	Off.	Type	Type Definition	Sub-Type
Group	22	0	0	Char / ASCIIZ String	0
MetaField	160	22	0	Char / ASCIIZ String	0
Style	2	182	2	Integer	0
Format	2	184	2	Integer	0
Format2	2	186	2	Integer	0
Decimals	2	188	2	Integer	0
Row	2	190	2	Integer	0
Column	2	192	2	Integer	0
Height	2	194	2	Integer	0
Width	2	196	2	Integer	0
Width2	2	198	2	Integer	0
Prompt1	120	200	0	Char / ASCIIZ String	0
Prompt1Row	2	320	2	Integer	0
Prompt1Column	2	322	2	Integer	0
Prompt1Height	2	324	2	Integer	0
Prompt1Width	2	326	2	Integer	0
Prompt2	120	328	0	Char / ASCIIZ String	0
Prompt2Row	2	448	2	Integer	0
Prompt2Column	2	450	2	Integer	0
Prompt2Height	2	452	2	Integer	0
Prompt2Width	2	454	2	Integer	0
Prompt2ColorSet	2	456	2	Integer	0
KeyName	22	458	0	Char / ASCIIZ String	0
Help	22	480	0	Char / ASCIIZ String	0
LookupName	22	502	0	Char / ASCIIZ String	0
LookupRow	2	524	2	Integer	0
LookupColumn	2	526	2	Integer	0
LookupWidth	2	528	2	Integer	0
LookupCount	2	530	2	Integer	0
LookupPattern	2	532	2	Integer	0
Footer	160	534	0	Char / ASCIIZ String	0
ShowSecurity	2	694	2	Integer	0
ChangeSecurity	2	696	2	Integer	0

The primary detail system for Displays is the Display Field system, which defines individual elements to be Displayed or acted upon when the Display is in use. There are some usage conventions explained after the definition.

Group Name Name used to associate this Field with Display Structures

MetaField Multi-purpose field: Name of a field in Display Structure's database, Name of a Virtual Field, or a Command for a Text-format action field.

Format A Bit Field that describes how the information in the MetaField will be displayed. There is a nested field (subscreen) in the VariForm system to provide yes/no and multibit field lookups for this field.

Format2 Extended format bits: A second bit field, exclusively for enhanced reports (type 4 displays), that describes if and when the information in the MetaField and Prompts are displayed. There is a nested field (subscreen) in the VariForm system to provide yes/no lookups for this field.

Primary Data Structures

Display System Detail

Style Information about the field's contents, and basically how it should be displayed.

Decimals Multi-purpose field: Number of fractional digits to display, true width per line of a multiline field, secondary format of a nested field, or first bit position in a bit/multibit field.

Width2 Number of bits to use in a numeric, multibit field.

Row, Col Position on the display screen for the field to be displayed, and sorting order for this field within the display group.

Height Normally 0 or 1, number of rows for field

Width Number of field characters to show, space or prompt-character padded.

Prompt1 Primary Text prompt to associate with this field: contents

P1 Row,Col Position on the display screen for the Prompt to be displayed

P1 Height Number of rows for Primary Prompt Text

P1 Width Width of area to color when Field is current

Prompt2 Secondary Text prompt to associate with this field: contents

P2 Row,Col Position on the display screen for the Prompt to be displayed

P2 Height Number of rows for Primary Prompt Text

P2 Width Width of area to color when Field is current

P2 ColorSet Values of 1-4 provide for alternative coloring of Secondary Prompt text/Secondary Prompt area

(All lookup fields will be shown on the next page)

Func Keys Function Keys specifically attached to this field.

Help Help reference to use when the system help key is pressed on this field

Help Line Help text to display on the bottom of the screen when this field is current

ShowSecur. Numeric value of minimum user security required for this field to appear on a report, browser, or display.

ChangeSec. Numeric value of minimum user security required for this field to be 'editable' on a browser or display. If the user's security falls below this value, the field may display (see above), but will be automatically set to 'uneditable'.

Primary Data Structures

Display System Detail

Lookup Name of a search structure that will automatically display reference information attached to the field's value. The location and amount of information displayed is controlled by the LookupPattern field. Alternative use: Display Structure name of the next-layer-down on Nested Fields (type 3,4).

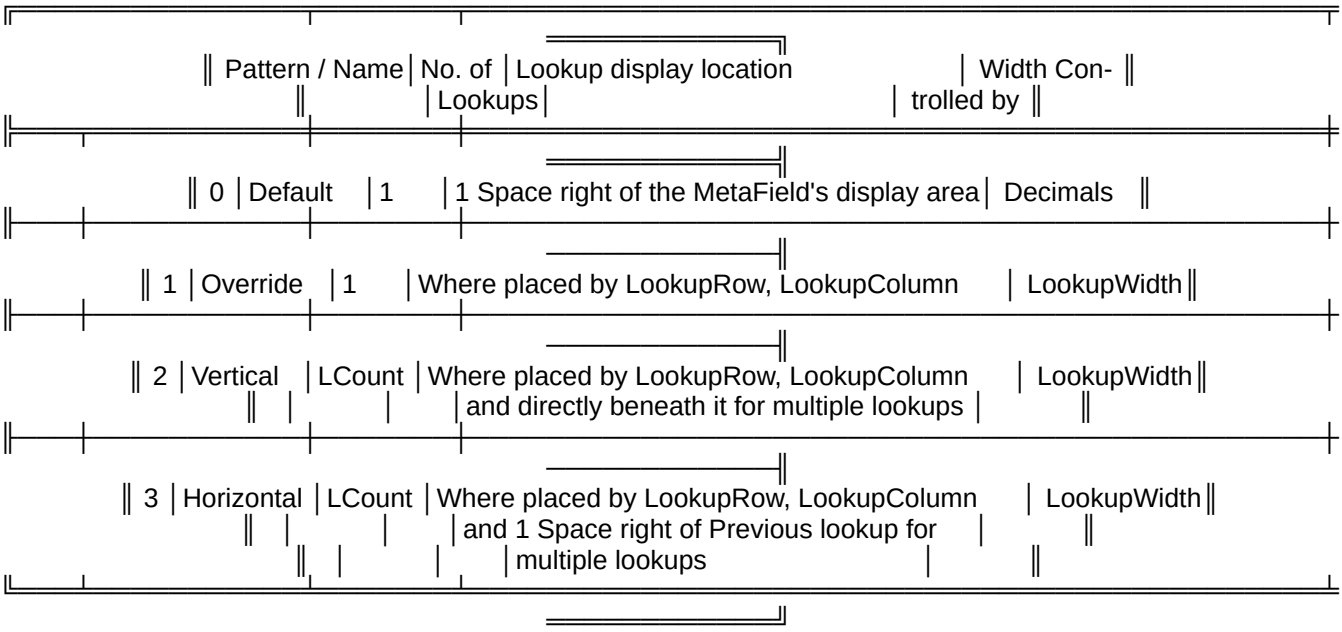
Lk Row Optional row of the display or report where the first lookup will be shown.

Lk Column Optional column of the display or report where the first lookup will be shown.

Lk Width Optional width of each display associated with this lookup.

Lk Count Optional number of displays to be shown by this lookup. (Developer's note: lookup searches fill the #Target# fields in order: #Target#, #Target2#, #Target3#, and so forth. The display or report size is the limiting factor.)

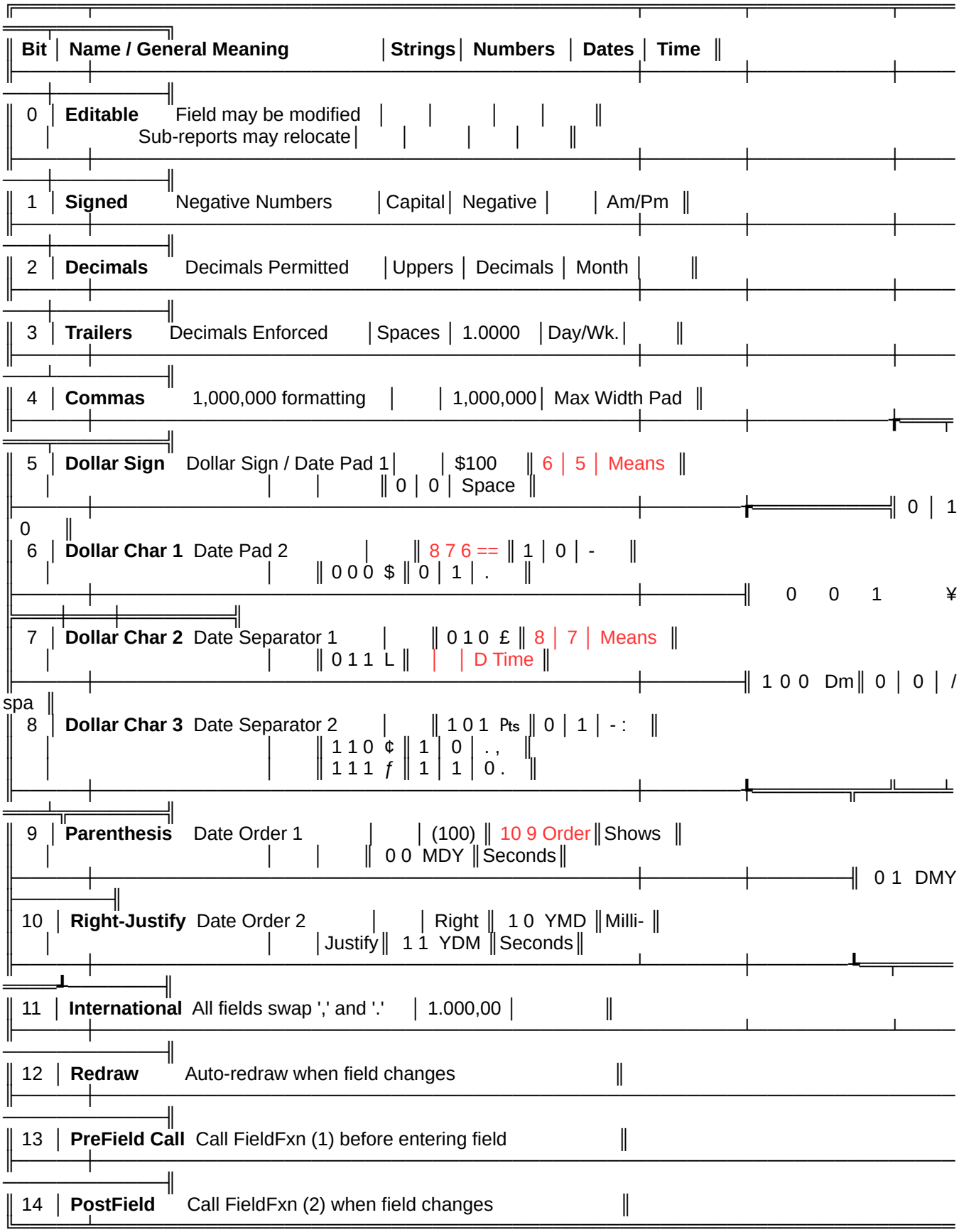
Lk Pattern A code indicating how many lookup description fields are to be displayed, and in what format.



Primary Data Structures

Display System Detail

Display Field Format Bits



Date Separators:

Date separators act differently, based on whether the display field is a date or time style. For Date Fields, bits 8 and 7 determine the appropriate separator, shown in the 'D' column. For Time Fields, bits 8 and 7 use the separator as per the 'Tim' column.

Date Order:

00: MDY, as in October 12, 1993 is the American Date Convention
01: DMY, as in 12 October, 1993 is the Military / European Date Convention
10: YMD, as in 1993, October 12 is the Clerical / Sorting Convention
11: YDM, is a seldom used format (1993, 12 October)

Primary Data Structures

Display System Detail

Display Field Styles

Value	Name	Meaning
0	**Text Only**	Menus and Displays/Reports: Prompt1 will display in text color set 1 Browsers and Sub-reports: Immobile, or un-replicated field Menus: MetaField is command performed if selected
1	**Yes/No**	Integer / Long int fields: "Yes" is a non-zero value, "No" is 0
2	**String**	Character strings to display and edit
3	**Control String**	Do not display/edit string: It may contain display control characters
4	**Standard Numeric**	Base 10 (normal) numbers
5	**NonStandard "**	Non-base-10 numbers: Decimals Field is the base: 2-36 Whole numbers only
6	**Date**	Full calendar date information: Valid from year 0-32000
7	**Time**	Time, based on format: Time of Day, or Hours:Minutes:Seconds:Mills
8	**Nested**	Subscreen or sub-browser: always uneditable Shows "Press Enter" when current Decimals Field is sub-type
9	**Bit Field**	Field is a Yes/No field that only affects 1 bit of field named by MetaField: Decimals Field is bit position: 0-n
10	**Multibit Field**	Field is a numeric that only affects a group of bits of field named by MetaField: Decimals Field is bit start position: 0-n Width2 Field is #/bits: 1-n

Primary Data Structures

Display System Detail

Display Usage Conventions

Browsers/PickLists/Sub-Reports:
These display structures are characterized with 2 sections: a 'fixed' prompt set, and a 'floating' content set. To achieve this:

Display Structure must have an ItemHeight > 0, which is the maximum size of a record in rows for the element.

Uneditable fields are considered 'static', and are only drawn at the beginning of a window.

PickLists, Sub-Browsers (Browse-Lists), and Sub-Reports need to have a search condition, if for no other reason than to generate return values.

Standard Browsers generally should not have a search condition, since this disables the user's ability to change display indexes.

Editable fields are considered 'dynamic', and are repositioned and redrawn for each occurrence of a record. There is one difference between browsers and sub-reports: Browsers do not reposition the Prompt Field(s) of active Field entries, whereas Reports do. So, for reports, put the titles as prompts in uneditable text fields; in browsers, the prompts should be part of the appropriate field records (this is how Browser/Entry mode knows which prompts to highlight when).

Of all of the 'dynamic' structures, only the reports make use of Nested Fields, which they use to call subsidiary reports. Browsers may have nested fields, but the user must be in Browser Entry Mode to make use of them.

Menus:
These display structures are characterized by no item height, and all uneditable text fields. The display contents are in prompt1, with the 'hotkey' in prompt2, with appropriate positioning and a width of 1.

For neatness, menu prompts should have a width of the Display Structure's display width, regardless of the length of the prompt text.

To make the hotkeys stand out, select an alternative color for Prompt2. Variform's system default menus use Color Set 1 for this.

Primary Report Entries:
Should be at or near page size, to accommodate page breaks and page numbering fields, with no item height. The Display Field elements are generally little more than date, time, page numbering, and a nested field or two to call sub-reports.

Secondary (sub) Report Entries:
Should be at or near Primary report size for best use of the page. They are either static (no item height), or dynamic (item height >0, like a browser). If a search calls too many Sub report elements for the size, the system will extract the primary and secondary, and re-generate them for the next 'page'.

Primary Data Structures

Search System Detail

Search Database

Name	Size	Off.	Type	Type Definition	Sub-Type
Search	22	0	0	Char / ASCIIZ String	2
Comment	60	24	0	Char / ASCIIZ String	0
Direction	2	84	2	Integer	0
Parent	22	86	0	Char / ASCIIZ String	0
Child	22	108	0	Char / ASCIIZ String	0
Index	22	130	0	Char / ASCIIZ String	0

The Search System Overview is stored in the Search Database. Fields are as follows:

Search Unique Name to reference this search.

Comment Text to appear describing what/where the Search operates.

Direction Nonzero (yes) values here force the search to function in reverse order with respect to the index, or from the last record to the first record. Any search with a nonzero direction should have a type 5 and 7 search field record; a zero direction should have a type 5 and 6 search field record.

Parent Db Name of the parent, or map-receiver database. Optional.

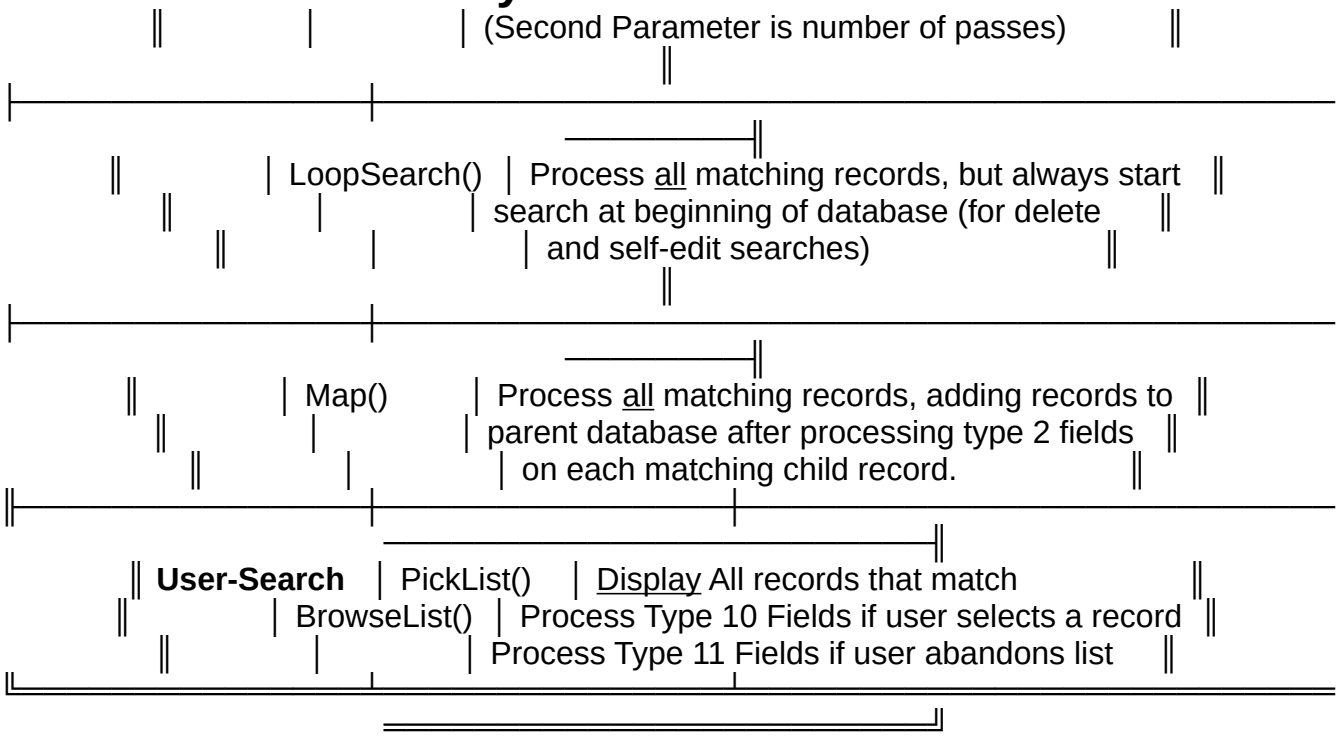
Child Db Name of the child, or map-source database. Mandatory.

Index Name of the index in the child database to use during the search. While optional, this field can dramatically affect the speed or the search.

Search Uses

Style / Name	Alternative	Uses
Single Search	DoSearch()	Process a single matching record
	Lookup Name	Type 3 Searchfields called if there is no match
Multi Search	MultiSearch()	Process all matching records in a database

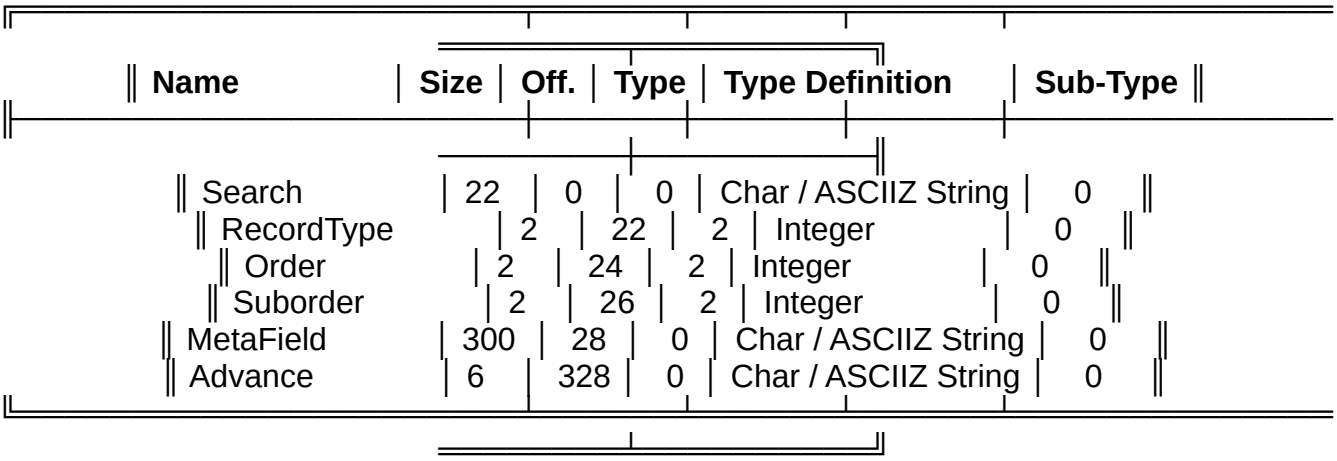
Primary Data Structures



Primary Data Structures

Search System Detail

SearchField Database



Name	Size	Off.	Type	Type Definition	Sub-Type
Search	22	0	0	Char / ASCIIZ String	0
RecordType	2	22	2	Integer	0
Order	2	24	2	Integer	0
Suborder	2	26	2	Integer	0
MetaField	300	28	0	Char / ASCIIZ String	0
Advance	6	328	0	Char / ASCIIZ String	0

The details of the search are stored in the Search Field Database. These records serve a variety of purposes, and may take various forms.

Search Name of Search Structure this MetaField is attached to

RecordType Type of MetaField: its purpose within the search (see table)

Order, Sub Numbers to keep MetaFields within a Type in consistant order.

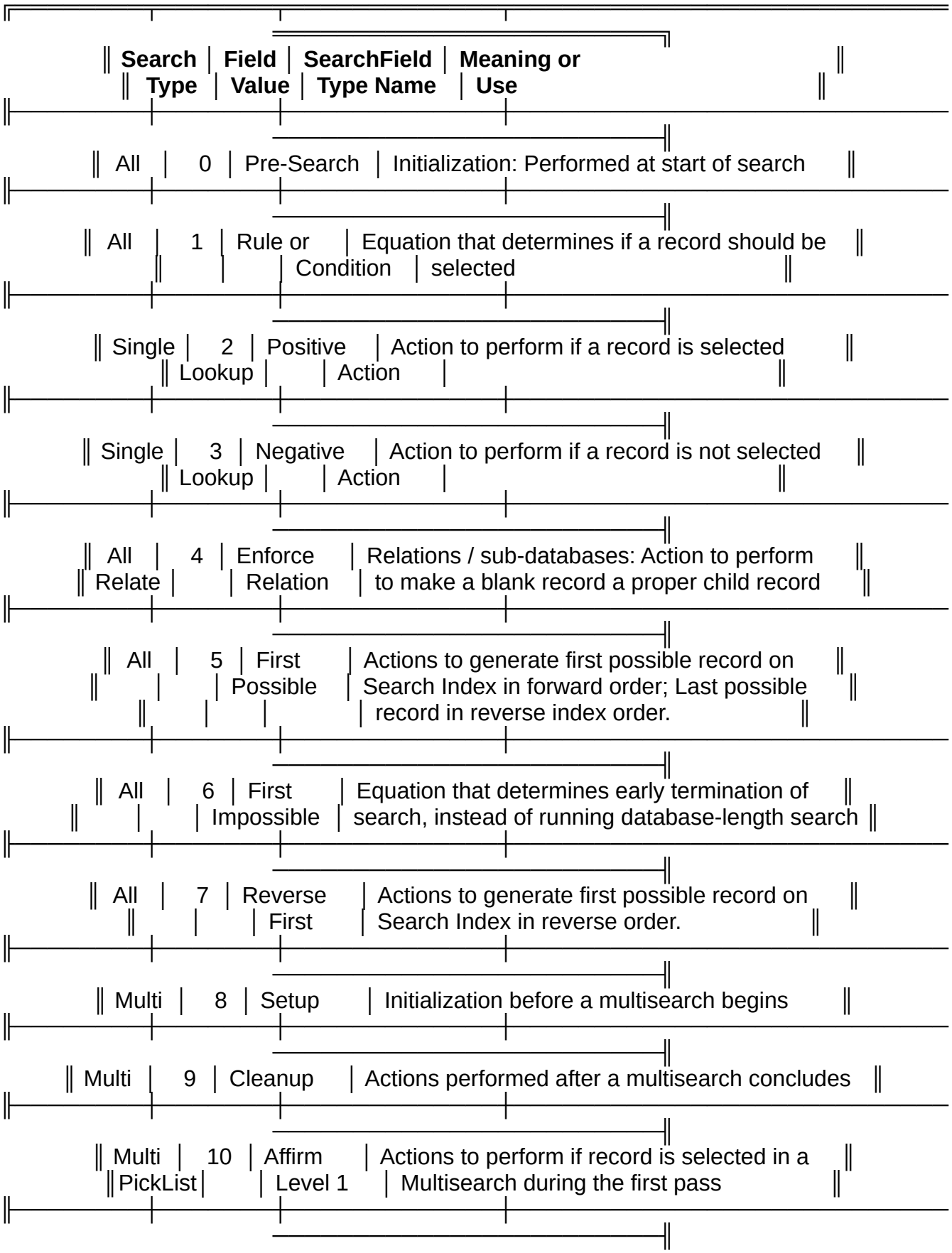
MetaField Actual Formula or equation to perform.

Advance Logical Expression to connect very long Logical Statements, primarily for Type 1 records.

Primary Data Structures

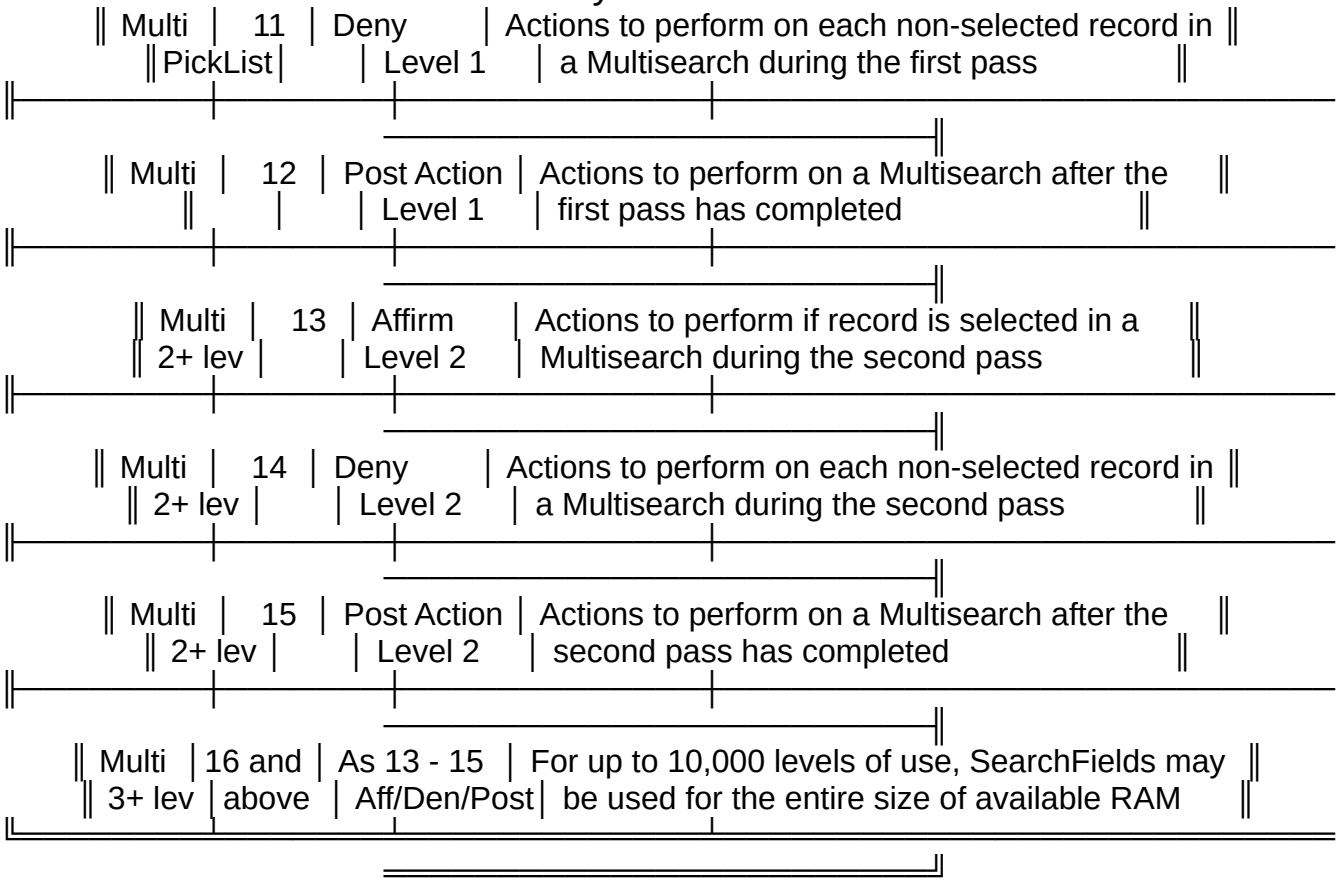
Search System Detail

SearchField Types



Primary Data Structures

Search System Detail



Procedures

VariForm does not require the Procedure system to operate as a Relational Database Management System. Procedures are groups of actions that can be called from any facet of VariForm, to perform special tasks.

- A Menu Selection can call a procedure
- Search Fields can call procedures
- Function keys can call a procedure
- The CommandLine Interface can call procedures
- Procedures can call other procedures, including themselves
- Reports can call procedures before or while running

Anywhere there is a MetaField, there can be a "Call" or "IfCall" statement. These statements call and run Procedures. The VariForm Procedure System is a structured, procedural language.

Principal Definition: a VariForm Procedure is a series of one or more records in the Procedure database which comprise one or more statements that are fed to the command processor through the procedural interpreter.

One basic definition of a programming language is "a system of symbols that allows manipulation of variables and other information in a human-readable form". VariForm procedures fill that bill:

Procedures can assign and/or remove variables from the Virtual Field List

Procedures can manipulate database records and fields directly, or through searches

Procedures can manipulate virtual fields, redraw data entry screens, pack databases, call reports, and even issue 'DOS' commands.

Another definition of a programming language asserts that "... a system of conditions, branches, and iterative loops is required...", and VariForm procedures have those as well:

Procedures can contain a variety of conditional branches, including "IfIs", a conditional equivalence statement, "If"/"ElseIf"/"Else" for conditional execution (branching), and "IfCall", a one-line conditional subprocedure call.

For iteration (things that must be done a number of times), VariForm offers "DO ... While", and "While..." loops, with optional "Break" statements for hasty retreats out of loop conditions.

Procedures

Procedures also have the ability to call 'requestors', which are data entry screens that use fields in the Virtual Database, to get user-input on what operations to perform, and how to perform them. A record deletion procedure, for example could bring up a requestor to make sure the user wanted to delete records, and then to specify what conditions must be met to delete the records. For example, "Delete all sales records with sale dates before today", could be a procedure such as follows:

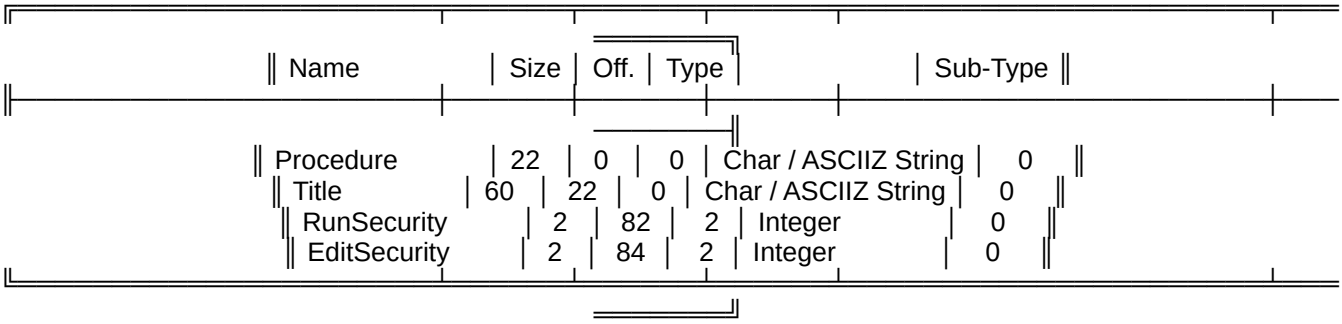
```
LoadVirtuals( "DeleteSales")
Confirm = 0
KillDate = Today
Requestor( "DeleteSalesReq" )
If( Confirm ) {
        LoopSearch( "DeleteSales")
}
Else {
        Pause( "No confirmation: No Sales records were deleted" )
}
KillVirtuals( "DeleteSales" )
```

Where the Virtual Field Group "DeleteSales" contained an int "Confirm", and a Date "KillDate" field, the requestor, "DeleteSalesReq", allowed the user to change those 'fields' in the Virtual Database. The LoopSearch "DeleteSales", would have a condition of >SalesDate < KillDate, preferably searching along an index that started with the SalesDate field, and whose positive (type 2) action was simply "KillRec()".

Technically, the "KillVirtuals" statement at the end of the procedure is redundant. All variables loaded by a procedure will be removed when the procedure has completed, but it is always smart to have explicit statments instead of assumptions.

Procedures

Procedure Database



Name	Size	Off.	Type	Sub-Type
Procedure	22	0	0	Char / ASCIIZ String	0
Title	60	22	0	Char / ASCIIZ String	0
RunSecurity	2	82	2	Integer	0
EditSecurity	2	84	2	Integer	0

The Procedure System Overview is stored in the Procedure Database.

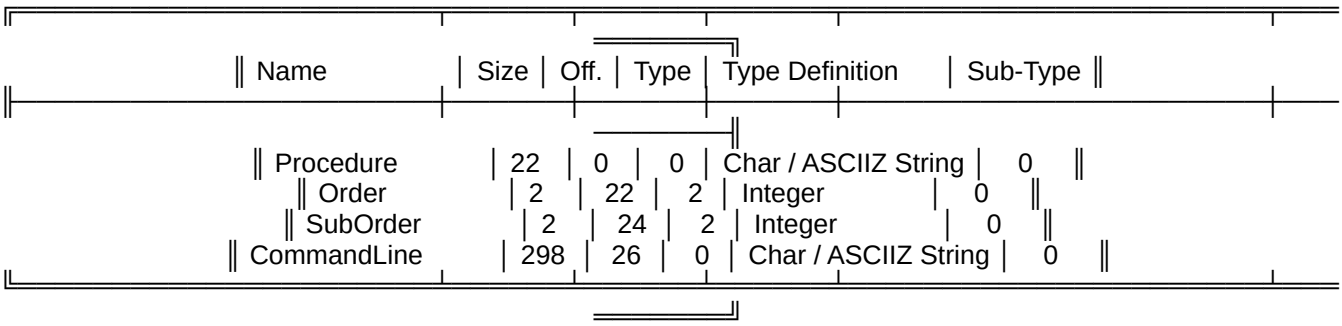
Procedure Unique Name to reference this Procedure.

Title Text description explaining what this procedure does.

RunSec'y Minimum Security Level required to Run this procedure.

EditSec'y Minimum Security Level required to perform a CommandLine() call within this procedure. Users below the minimum required security will not be sent to the commandline.

ProcedureField Database



Name	Size	Off.	Type	Type Definition	Sub-Type
Procedure	22	0	0	Char / ASCIIZ String	0
Order	2	22	2	Integer	0
SubOrder	2	24	2	Integer	0
CommandLine	298	26	0	Char / ASCIIZ String	0

There is a one-to-many relationship between the Procedure database and the Procedure Field database, throught the 'Procedure' fields. This facilitates data entry in the Procedure system two ways: the user may browse for the appropriate description in the Procedure database, and then, through a Search Type 4 mandatory relationship in the Nested Field, the appropriate Procedure field name will be inserted for the user each time they add a record.

Procedure Associates all procedure records with the same procedure field contents together, and relates them to the appropriate Procedure Title database record.

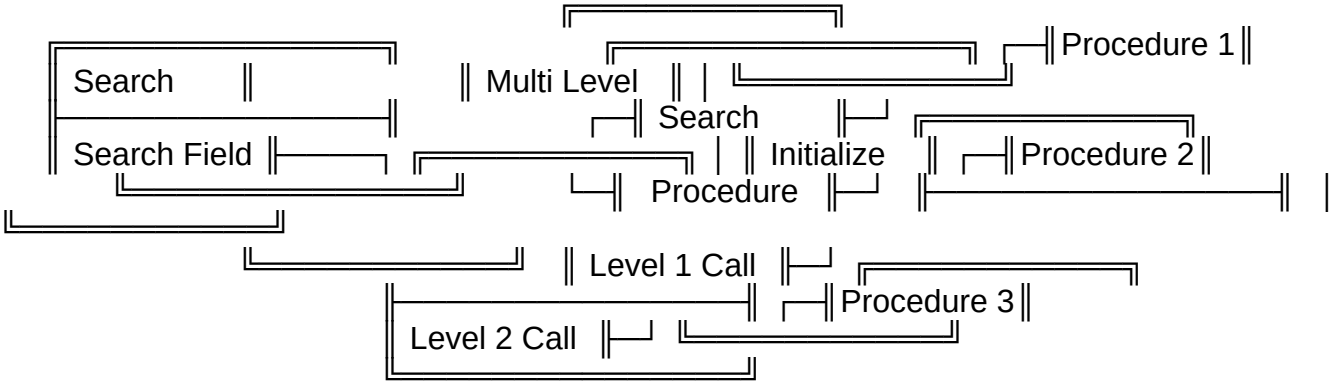
Order The primary organizational number. By convention, the first record is generally 0 or 1.

SubOrder The secondary organizational number. Records in the same procedure with the same order number are organized by suborder number, then by physical position in

the Procedure database. Most useful when inserting lines. VariForm's Procedure-renumbering routine generates a first record with Order at 1, SubOrder 0, and all subsequent records as SubOrder 0, with Order incrementing by 1 each time.

CommandLine The actual command to process. This may be 'commented', or disabled by placing a slash (/) on the first position of the line. Aside from this, the line may be blank, contain procedural references (if, elseif, else, do, while, or }), statements or function calls.

Searches versus Procedures



There are two distinct and related activities that can occur within VariForm, namely Searches and Procedure Calls. Most sophisticated development efforts will require that the two activities interact on several levels. This is where a little bit of anticipatory planning can make the difference between a 2-week and a 3-month development effort.

In general, a Search has 3 active (procedural), and 4 passive (searching) parts:

Active Parts of a Search:

Initialization :	setting up variables or tests (type 0, 5, 8, 9, 12, ...)
Success actions:	actions performed upon success (type 2, 10, 13, ...)
Failure actions:	actions performed upon failure (3, 11, 14, ...)

Passive Parts of a Search:

Defined Start:	establishes a 'first possible' record (type 5)
Conditions:	determines the success or failure for a record (type 1)
Search Limiters:	defines 'last possible' record (type 6)
Relational Maps:	define map-filling process of relational search (type 4)

Each of these parts can consist of a series of SearchFields, which can do something on its own, or run a procedure, or even perform another search. Where this can get complicated is when a procedure, called by a search field, calls another search.

A Procedure may have many active parts, or just itself; the activity of the parts is determined by procedural conditions (if, else, while), instead of the individual record 'type'.

Search Condition Advantage

The single advantage that Searches have over Procedures, in terms of processing capability, is the ability to process extremely long, complex logical expressions easily.

Search Field Type 1 (Condition) Records use the 'Advance' field to link multiple Type 1 records into one huge (greater than 300 bytes) logical statement. This allows complex expressions without the need for temporary variables, as is the case in procedures, where each 300 byte line is a distinct entity.

Searches vs. Procedures

Procedural Advantages

For all Search Field Types except 1 (Condition) and 5 (Fast Start on forward Search), Procedures have several advantages over multiple SearchFields:

◦ Procedures are not loaded with the search unless needed:
- Single Searches force type 1, 5, and 6 or 7 to load
- Multi-Searches also force types 8 and 9 to load
- Relational Searches, if called, force type 4 to load
- All other types are only loaded on success or failure conditions

◦ Procedures take less space in memory:
- The internal (RAM) structure is about 12% smaller when loaded

◦ Procedures allow complex 'if' statements:
- Search commands are limited to '1-liners', such as IfIs, or IfCall

◦ Procedures may be re-used without load time or memory penalties:
- Multi-Level Searches can often be replaced with a simpler search that uses a more sophisticated single procedure for all search levels.

- Once a procedure is loaded by a search (or browser or displaystruct), all subsidiary procedures and searches have unrestricted access to it without forcing a reload, and without additional memory consumption.

Rules of Thumb

For type 2, 3, 4, 10, 11, 13, 14, et cetera SearchField Types (i.e.: excluding Types 0, 1, 5, 6, and all pre/post multilevel SearchField Record Types):

◦ If there are six or more records of a single type, a single Call() record will certainly be more efficient.

◦ If there are three or more records of a single type, a single Call() record will probably be more efficient.

◦ If there are any IfCall() statements in addition to one other record of that type, a single Call() that accesses a procedural If(){ is not only more efficient, but is also significantly faster.

◦ **Do not use procedures for Type 1 or Type 5 records! Ever!** The Search System performs other processes based on these elements, which the procedure process will utterly defeat. At best, this can cause trouble during browser search-mode work on a limited browser. At worst, this can cause system hangs in the BrowseList/PickList system, in sub-reports, or on Nested (type 3,4) Field accesses.

Function Keys

On The Keyboard

Function keys give leverage to the user. A single keystroke can do the work of thousands of lookups, reducing drudgery and typing time. An Information Management Environment is not complete without programmable function keys. VariForm recognizes up to 40 different function keys from the keyboard at any given time: F1-F10, shift F1-F10, control F1-F10, and Alt F1-F10. VariForm does not recognize F11 or F12, since those keys are not available on every keyboard.

Function Keys are stored in groups, so that several related function key actions can be attached to a DisplayField or DisplayStructure through one reference name. The reference name, along with other information, is stored in the FunctionKeyGroup database. Groups of function keys can be shared by different screens, or by different fields on the same screen.

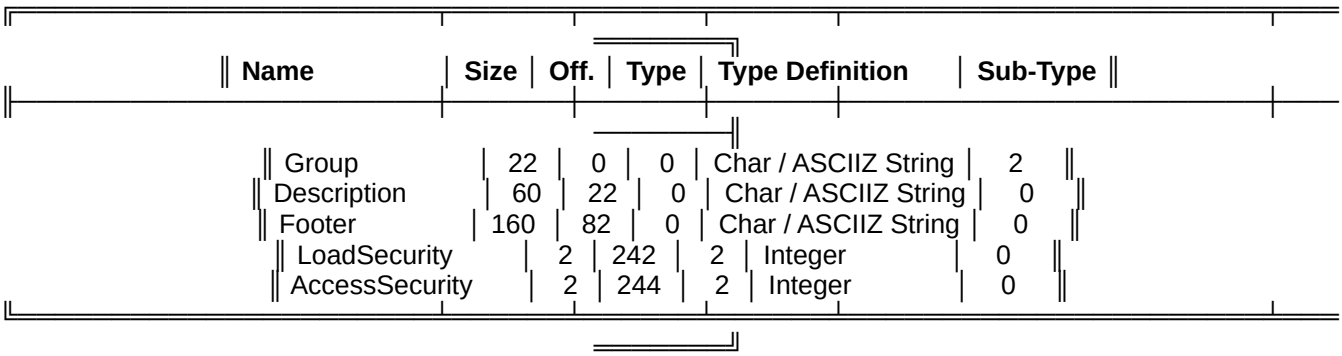
Programming function keys is accomplished by a data entry system on the FunctionKeyField Database, which provides a 300-byte CommandLine for every possible function key. Every Function Key on the keyboard is a programmable Procedure Line that can call a Procedure when pressed.

A data entry screen usually contains some generic function keys, such as help, redraw, show memory, and so forth. Any given DisplayField on the data entry screen may also have field-specific function keys, such as calling a PickList, or performing a calculation using the current field's contents.

One of the unique features of VariForm is the ability to compound function key definitions. Field-specific function keys can override the current data entry screen's function keys automatically, when the user is on the appropriate field, if the same keys are defined.

Function Keys

FunctionKeyGroup Database



Name	Size	Off.	Type	Type Definition	Sub-Type
Group	22	0	0	Char / ASCIIZ String	2
Description	60	22	0	Char / ASCIIZ String	0
Footer	160	82	0	Char / ASCIIZ String	0
LoadSecurity	2	242	2	Integer	0
AccessSecurity	2	244	2	Integer	0

Group Unique name for this set of function keys.

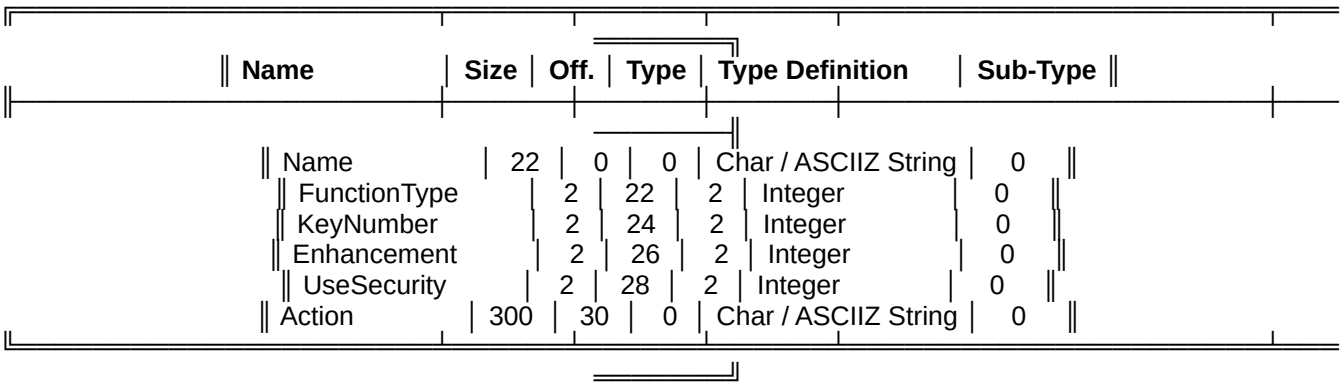
Description A text description to clarify what this group of keys does.

Footer Optional: a 'recommended' cue line for the DisplayStructure or DisplayField using this FunctionKey group. The footer can be inserted into the DisplayStructure or DisplayField's 'help line' field by pressing <F3> (load help) after selecting the FunctionKey group.

LoadSec'ty Minimum security required to load this FunctionKey group.

AccessSec Minimum security required to access this group's field records.

FunctionKeyField Database



Name	Size	Off.	Type	Type Definition	Sub-Type
Name	22	0	0	Char / ASCIIZ String	0
FunctionType	2	22	2	Integer	0
KeyNumber	2	24	2	Integer	0
Enhancement	2	26	2	Integer	0
UseSecurity	2	28	2	Integer	0
Action	300	30	0	Char / ASCIIZ String	0

Name Name of the group to which this record belongs.

FunctionType Classification of functional group (unused).

KeyNumber An integer between 1 and 10, inclusive, that indicates which function key will activate this feature.

Enhancement An integer between 0 and 3, inclusive, that indicates any needed enhancements to the function key. This allows for 40 function key definitions, instead of just

10. (Note: Other values exist: see table, below).

UseSecurity The minimum security required to activate this function. Users below this level will not have access to this function.

Action A CommandLine which is run if this specific function key is pressed when this function key group is in use.

Function Keys

Function Key Enhancement Codes

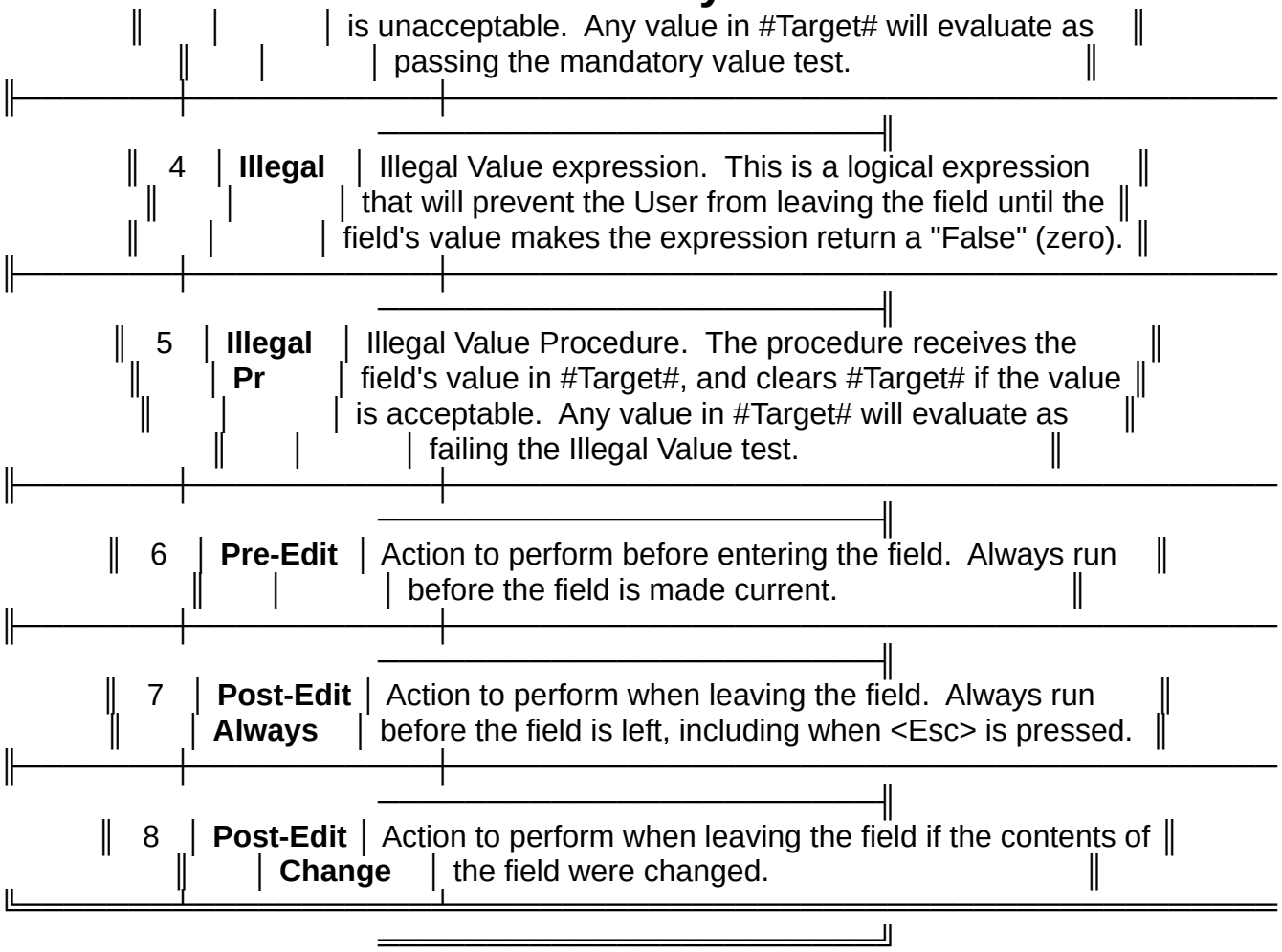
Enhance	Name	Description
0	**None**	The function key by itself
1	**Shift**	The shift key must be held down before and while pressing the function key. This is not related to CapsLock or NumLock.
2	**Control**	Similar to shift, except the <Ctrl> key is used.
3	**Alt**	Similar to shift, except the <Alt> key is used.
16	**Field**	Automatic actions attached to a DisplayField.
32	**Screen**	Automatic actions attached to a Display Structure.

Function Keys As Automatic Actions

FunctionKeyField record usage also extends to "KeyLess", or automatic functions. These functions include defining default values for a field, actions to perform during a record deletion, or rules to prevent or allow record deletion. This is accomplished through the FunctionKeyField database, using specific numbers to indicate when this function is performed.

Automatic Field Actions: Enhancement #16

Key #	Name	Description
0	**Default**	Default value to assign to this field when blank.
1	**DefaultPr**	Procedure that returns default value in #Target#.
2	**Mandatory**	Mandatory Value expression. This is a logical expression that will prevent the User from leaving the field until the field's value makes the expression return a "True" (non-zero) value.
3	**Mandatory Pr**	Mandatory Value Procedure. The procedure receives the field's value in #Target#, and clears #Target# if the value



Function Keys

Automated Screen Controls: Enhancement #32

Key #	Name	Description
0	**Pre-Add**	Procedure to perform before opening a 'blank' addrec screen.
1	**Mandatory To Add**	Mandatory Value procedure. #Return# is set to 1 before calling this procedure. If the procedure changes #Return# to 0, the record will not be added, and the user will be returned to the data entry screen.
2	**Illegal To Add**	Mandatory Value Procedure. #Return is set to 0 before calling this procedure. If the procedure changes #Return# to 1, the record will not be added, and the user will be returned to the data entry screen.
3	**Post-Add**	Procedure to perform after the record has been added to the database.
4	**Pre-Edit**	Procedure to perform before opening an existing record for editing.
5	**Mandatory To Save**	Mandatory Value procedure. #Return# is set to 1 before calling this procedure. If the procedure changes #Return# to 0, the record will not be edited, and the user will be returned to the data entry screen.
6	**Illegal To Save**	Mandatory Value Procedure. #Return is set to 0 before calling this procedure. If the procedure changes #Return# to 1, the record will not be edited, and the user will be returned to the data entry screen.
7	**Post-Edit**	Procedure to perform after the record has been edited.
8	**Pre-Kill**	Procedure to perform before opening an existing record for deletion.
9	**Mandatory**	Mandatory Value procedure. #Return# is set to 1 before

	To Kill	calling this procedure. If the procedure changes #Return# to 0, the record will not be deleted, and the user will be returned to the browser.
10	**Illegal**	Mandatory Value Procedure. #Return is set to 0 before
	To Kill	calling this procedure. If the procedure changes #Return# to 1, the record will not be deleted, and the user will be returned to the browser.
11	**Thru-Kill**	Procedure to perform immediately after confirmation and approval to delete, but prior to the record's removal.
12	**Post-Kill**	Procedure to perform after record has been removed.

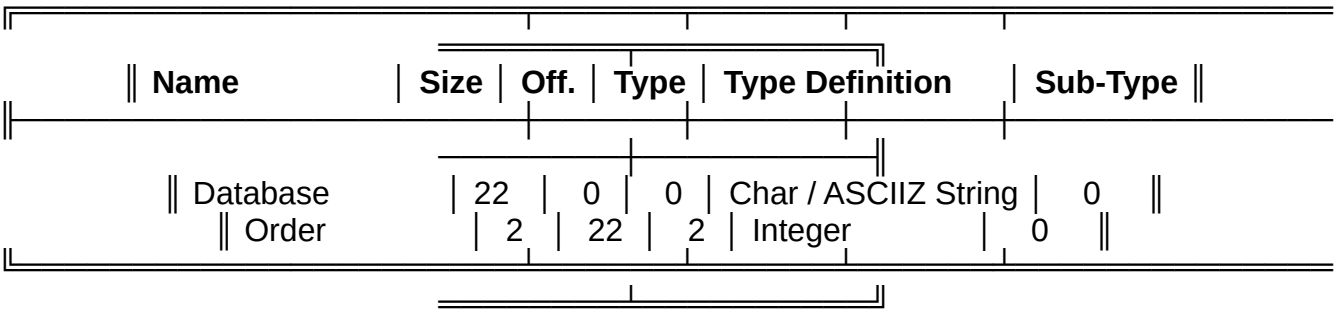
Secondary Data Structures

There are several databases that enhance the primary data structures. These specific-use databases are considered secondary, because the information has little meaning or use without being referenced by one of the primary data structures.

BatchPacking	list of databases that can be reorganized in a group
Chart of Charts	numeric abbreviations (number dictionary)
Colors	simple reference for sophisticated color schemes
Error/Action Log	historical diagnostic information
Help and HelpText	context-sensitive online assistance
Printers	special character formatting commands within reports
Report Batches	clusters of reports that can be called as a group
Table of Codes	text abbreviations (character-string dictionary)
User Settings	login security and some basic session information
Virtual Fields	automatic declaration of Variables for Procedures

Secondary Data Structures

BatchPack Database



Name	Size	Off.	Type	Type Definition	Sub-Type
Database	22	0	0	Char / ASCIIZ String	0
Order	2	22	2	Integer	0

When databases get very large, or the computer system slows down, such as a heavily loaded network, or a nearly full hard drive, 'Packing', or re-organizing databases, can take a long period of time. The procedure system allows users to set up commands, such as "CallPack(databasename)", that will allow packing of one or more databases overnight, without the need for human intervention.

The 'BatchPack' Database simplifies the process of packing several databases at once. Instead of using a "CallPack" for each database, and re-editing the overnight process script on a daily basis, users can list the databases to be packed in the BatchPack database, and the system will pack all listed databases, in "Order" sequence, with one call to "BatchPack()".

The **Database** field contains the name of a database to pack.

The **Order** field contains a number that organizes the list. The databases will be packed based on the lowest value of "Order" first.

Secondary Data Structures

Chart of Charts

VariForm allows users to cross-reference database information, to allow databases to store smaller abbreviations that still can contain meaningful values to the end-user. Many database systems call this capability a 'dictionary system', and rightly so; most dictionary systems constrain the meanings or uses of a reference to very limited uses. VariForm, on the other hand, allows any database to be used, in whole or in part, as a dictionary for other databases.

In VariForm, this cross-referencing process is called 'looking up information in an unrelated database', which refers to the fact that there is no mandatory relationship between the database containing the abbreviation, and the database containing the abbreviation's full meaning. The advantage of using databases for lookups instead of dictionaries is straightforward: it is one less interface to learn, and one less 'special process' to learn. A lookup is simply the results of a search performed on another database.

There are two distinct categories of lookup formats within VariForm: simple lookups and complex references. In general, a simple lookup analyzes to the contents of a single field across an entire database to produce results, while a complex reference analyzes the contents of more than one field to produce results.

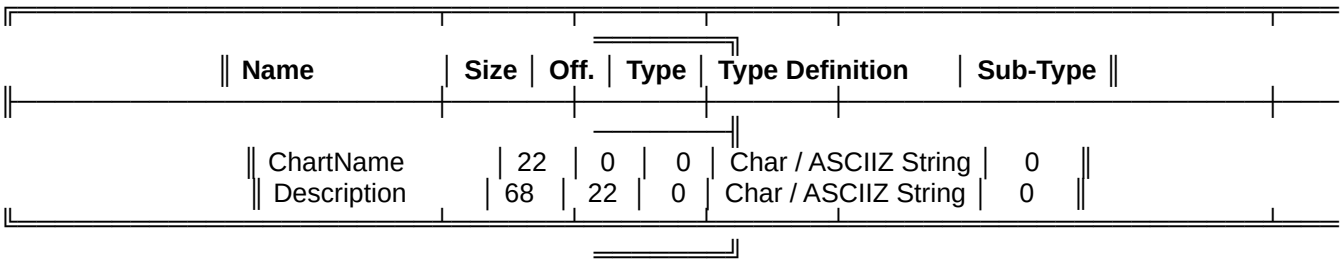
An example of a simple reference is the 'colors' database, as it is referred to in the Display Structure system: the contents of a 'color' field in the Display Structure database generates a lookup, which displays the contents of the appropriate Colors database record's Description Field. There is, in fact, a complex reference that is also performed, which provides all of the individual color fields to the Display Structure, when appropriate, but the User Interface of the Display Structure editing system remains a simple lookup.

Complex references essentially reduce a large database into several smaller databases, each of which is a simple lookup. VariForm uses a pair of complex references to show most of the system field abbreviations, for example. Those references are in the Chart of Charts, and the Table of Codes databases. The complex references allow VariForm to use 2 databases for a variety (30 or more) of lookup conditions, instead of having to use 30 or more separate databases.

Secondary Data Structures

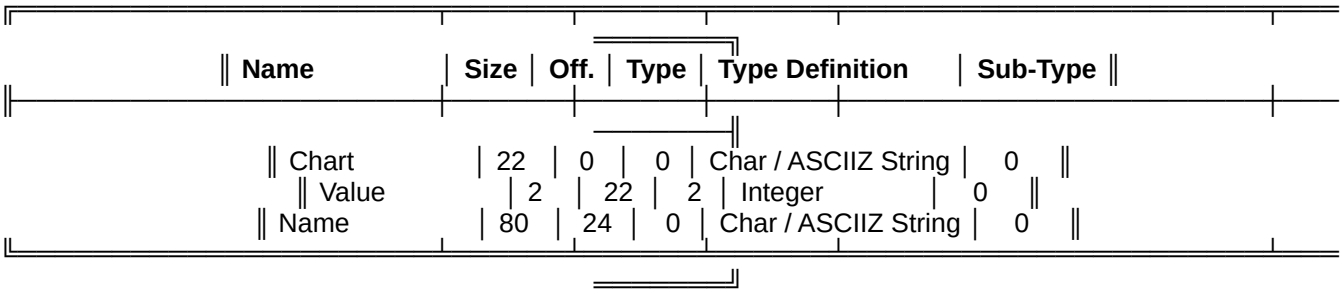
Charts

Chart Database



Name	Size	Off.	Type	Type Definition	Sub-Type
ChartName	22	0	0	Char / ASCIIZ String	0
Description	68	22	0	Char / ASCIIZ String	0

ChartField Database



Name	Size	Off.	Type	Type Definition	Sub-Type
Chart	22	0	0	Char / ASCIIZ String	0
Value	2	22	2	Integer	0
Name	80	24	0	Char / ASCIIZ String	0

To simplify data entry in the Chart of Charts, VariForm provides a Chart database, which simply contains the Chart Name, which is the primary field condition, and a text description of what the ChartName is used for. ChartField records are used to provide text descriptions of numeric values.

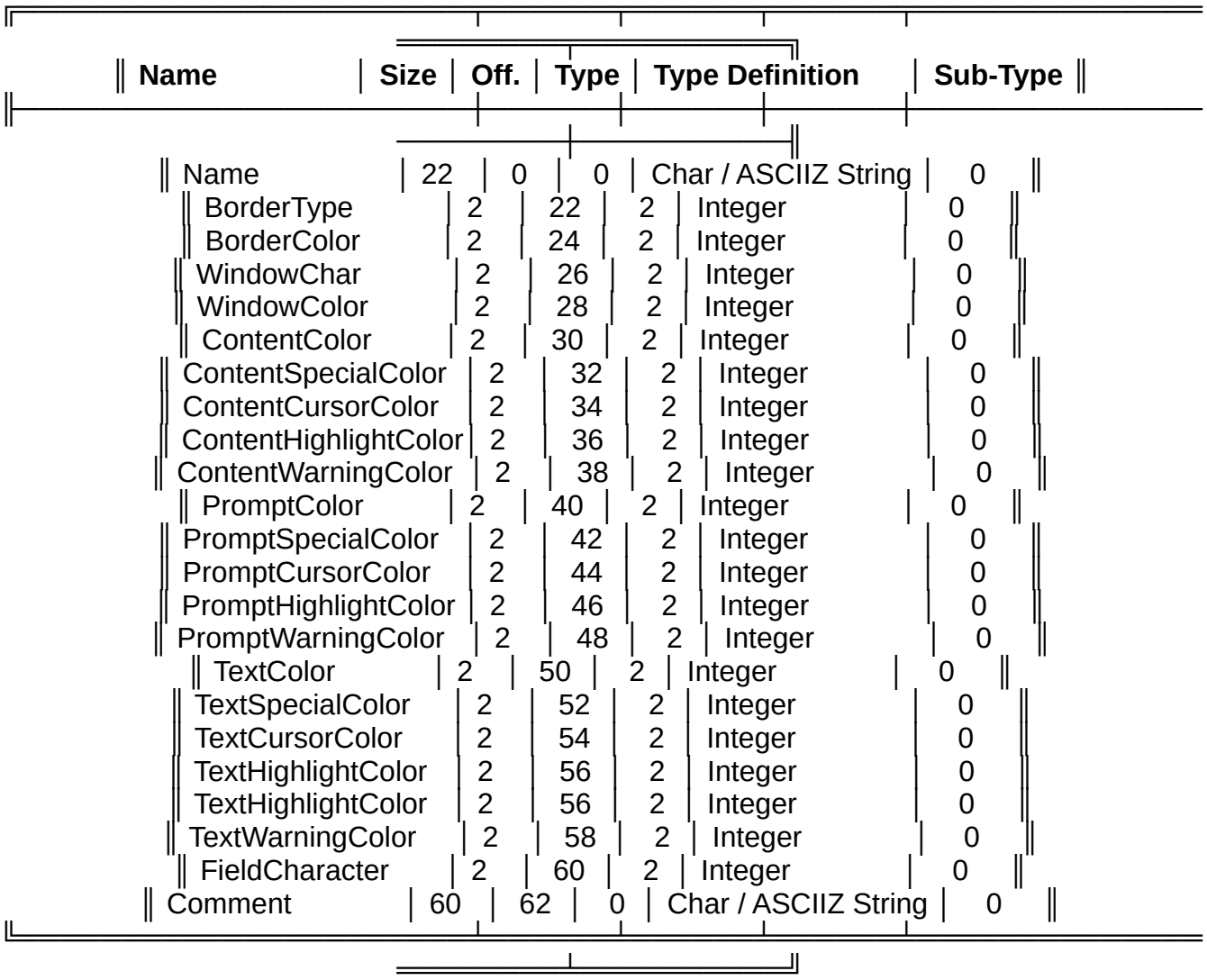
There is a one-to-many relationship between the Chart database and the Chart Field database, which connects 'ChartName' in the former to 'Chart' in the latter. This facilitates data entry in the chart system two ways: the user may browse for the appropriate description in the Chart Title database, and then, through a Search Type 4 mandatory relationship in the Nested Field, the appropriate ChartName will be inserted for the user each time they add a Chart record.

Example: There is a list of the text descriptions of the numeric color codes that use the ChartName 'colors'. That means that each appropriate entry in the ChartField database will contain 'colors' in the Chart field, and an appropriate Value and Name combination. This way, the value 0 in the colors chart refers to 'Black on Black'. Zero may have other meanings in other charts, (for instance, NULL in the ASCII chart), but not in the colors chart.

This is also a good example of sub-grouped uniques, which are unique fields in a group within a database, but which may be duplicated in other parts of the same database.

Secondary Data Structures

Color Database



Name	Size	Off.	Type	Type Definition	Sub-Type
Name	22	0	0	Char / ASCIIZ String	0
BorderType	2	22	2	Integer	0
BorderColor	2	24	2	Integer	0
WindowChar	2	26	2	Integer	0
WindowColor	2	28	2	Integer	0
ContentColor	2	30	2	Integer	0
ContentSpecialColor	2	32	2	Integer	0
ContentCursorColor	2	34	2	Integer	0
ContentHighlightColor	2	36	2	Integer	0
ContentWarningColor	2	38	2	Integer	0
PromptColor	2	40	2	Integer	0
PromptSpecialColor	2	42	2	Integer	0
PromptCursorColor	2	44	2	Integer	0
PromptHighlightColor	2	46	2	Integer	0
PromptWarningColor	2	48	2	Integer	0
TextColor	2	50	2	Integer	0
TextSpecialColor	2	52	2	Integer	0
TextCursorColor	2	54	2	Integer	0
TextHighlightColor	2	56	2	Integer	0
TextHighlightColor	2	56	2	Integer	0
TextWarningColor	2	58	2	Integer	0
FieldCharacter	2	60	2	Integer	0
Comment	60	62	0	Char / ASCIIZ String	0

The color database provides a mechanism for specifying an entire color scheme with one name: window colors, border types, colors of fields and prompts can all be controlled by the use of one "color name". This also allows systems with the same color scheme to re-use the color definition, instead of always re-loading the information from disk.

Fields for the color database are:

Name 22-character name field uniquely describes the color scheme.

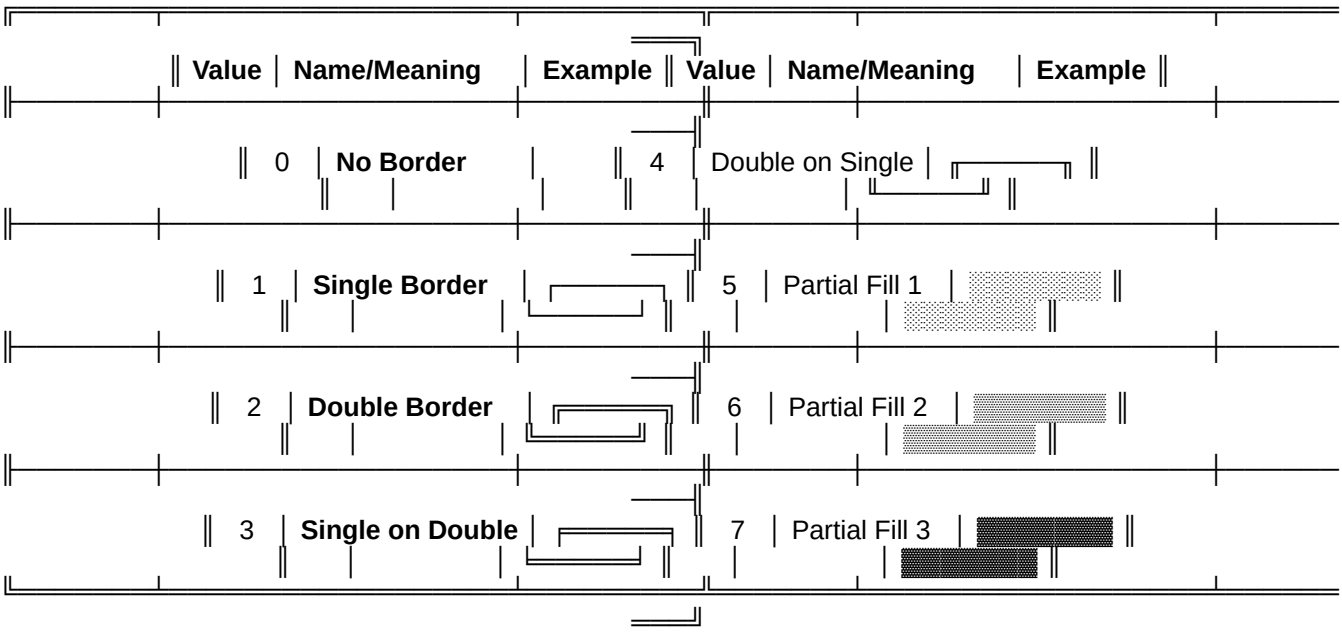
Comment For PickLists of colors, this text describes the color's intended use.

Border Type A number that signifies what kind of border to put on windows that use this color scheme.

Secondary Data Structures

Color Database

Border Type Values



Border Color Color to use when and if a border is drawn.

Window Char Default character for window background. Clearing the window fills it with this character. (Recommended: space or period).

Window Color Default color for the window background.

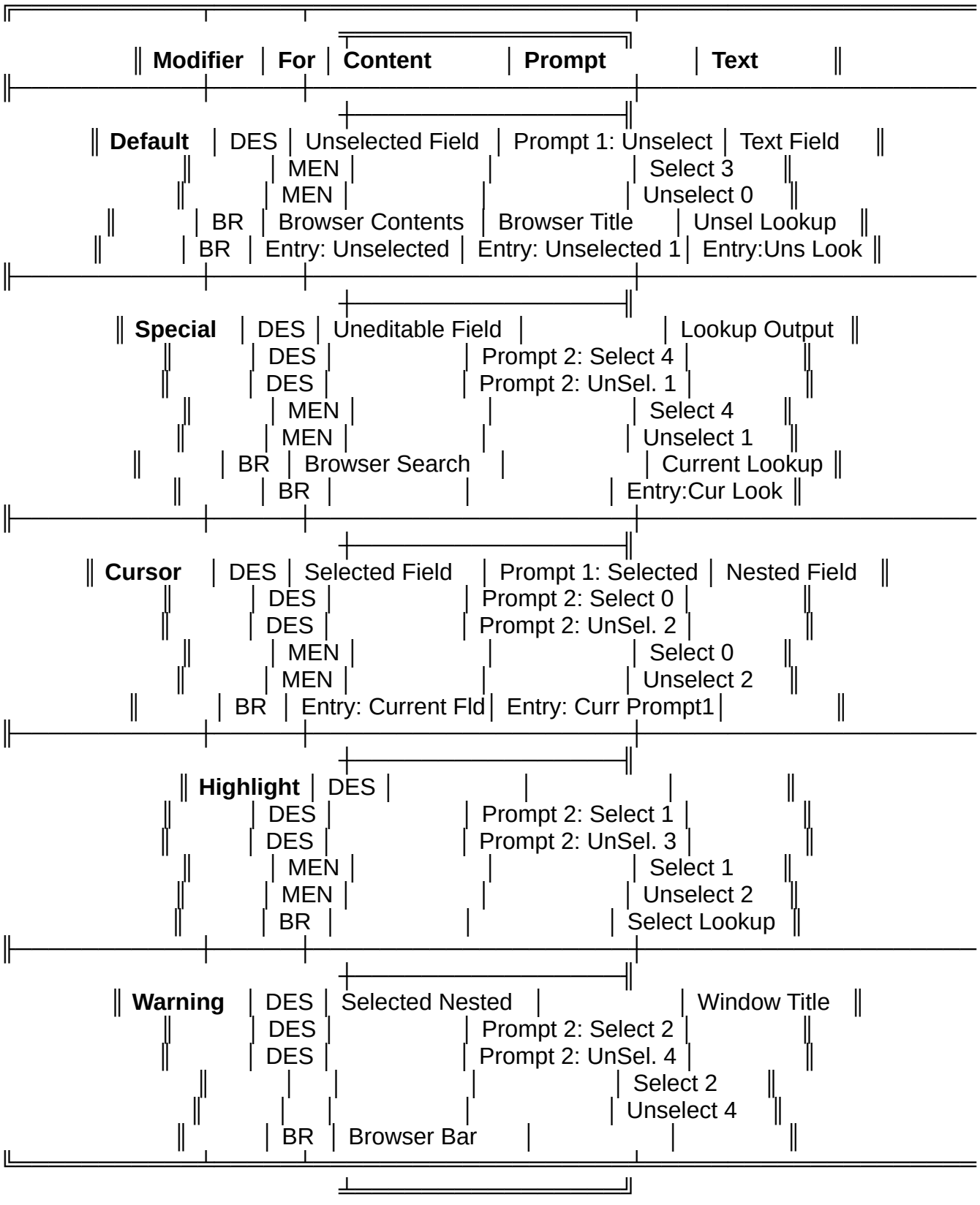
Field Character Character to use for Data Entry screens to show the user the size of the field, if the contents do not completely fill the field space.

Color Values There are 15 color values that make up the 'color scheme' for the calling window.

Secondary Data Structures

Color Database

Color Value Uses



Secondary Data Structures

Notes about the table

"For" column: Refers to what display formats use this color.

DES refers to Data Entry Screens, Reports, and Browser/Entry Mode.

Prompt 1 refers to the primary field prompt.

Prompt 2 refers to the secondary field prompt.

The number after the Select/Unselect note refers to the value of the "Prompt2ColorSet" field of the appropriate DisplayField.

BR refers to Standard Browsers, BrowseLists and PickLists.

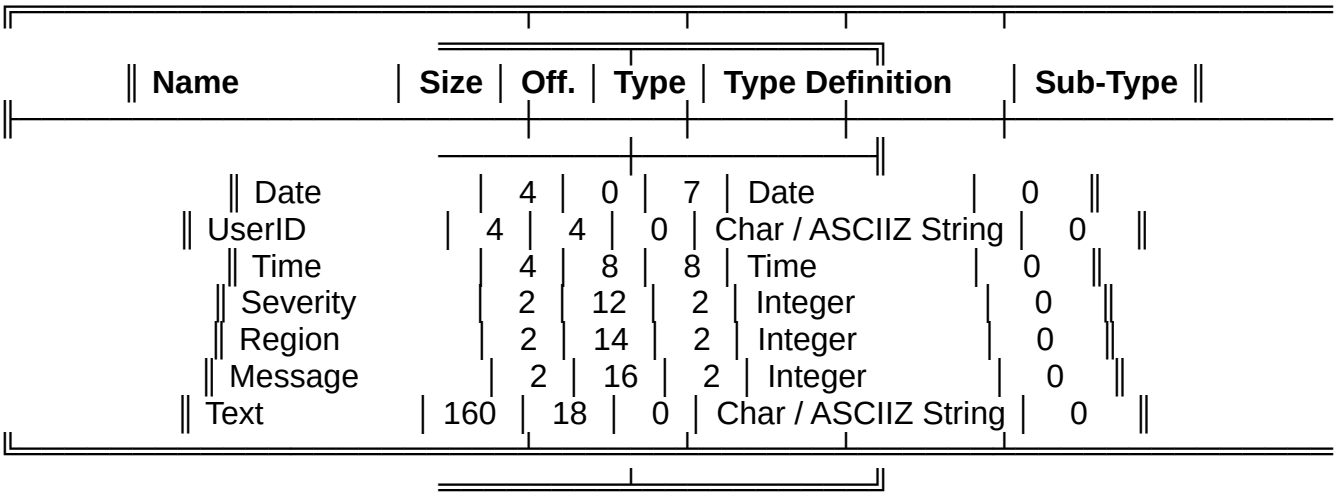
MEN refers to Menu Displays.

The number after the Select/Unselect note refers to the value of the "Prompt2ColorSet" field of the appropriate DisplayField.

It should be noted that the most common use for Menu Prompt2's is for 'HotKeys' in the line to stand out from the text being shown.

Secondary Data Structures

ErrorLog Database



Name	Size	Off.	Type	Type Definition	Sub-Type
Date	4	0	7	Date	0
UserID	4	4	0	Char / ASCIIZ String	0
Time	4	8	8	Time	0
Severity	2	12	2	Integer	0
Region	2	14	2	Integer	0
Message	2	16	2	Integer	0
Text	160	18	0	Char / ASCIIZ String	0

A consultant / programmer's worst nightmare does not begin when something goes wrong: it begins when the user doesn't remember what happened, or what complaints the system provided before things started going wrong. A user may not be at the console when an error message displays, or they may press the 'Any Key' (spacebar) reflexively before reading or writing down the error message. This type of behavior, on the part of the user, makes bugs almost impossible to track down, and drives developers crazy.

VariForm has a series of complaints and warnings that display before things start going wrong, and users have the ability to ignore those warning messages, as they do on any other system. However, if bugs do occur, a developer or system controller can set the "RecordSeverity" variable. This forces a copy of every complaint, along with a local time and date stamp and the user's id, to be saved into a database called the ErrorLog. Adjusting this variable provides control over the error levels that will be stored, from 'system recommended abort', down to the minutiae of what message was being displayed on the cue-line section of the computer screen.

Date/Time The date and time when the error occurred, according to the user's system clock.

UserID Abbreviation of the user (as logged in at VariForm Startup) having or finding the error. On standalone systems, this will almost always be "DEF" for "System Default".

Severity How important / significant is this error or message. This value tells VariForm whether to display the message, Pause the Message (press any key to continue...), or store the message in the ErrorLog Database.

Region Which module in VariForm is doing the complaining? This value tells the Developer more precisely what VariForm was trying to do when the problem occurred.

Message This is the number that controls the specific text of the message: what precisely was the error?

Text If VariForm has enough memory available, it will load a text message from the

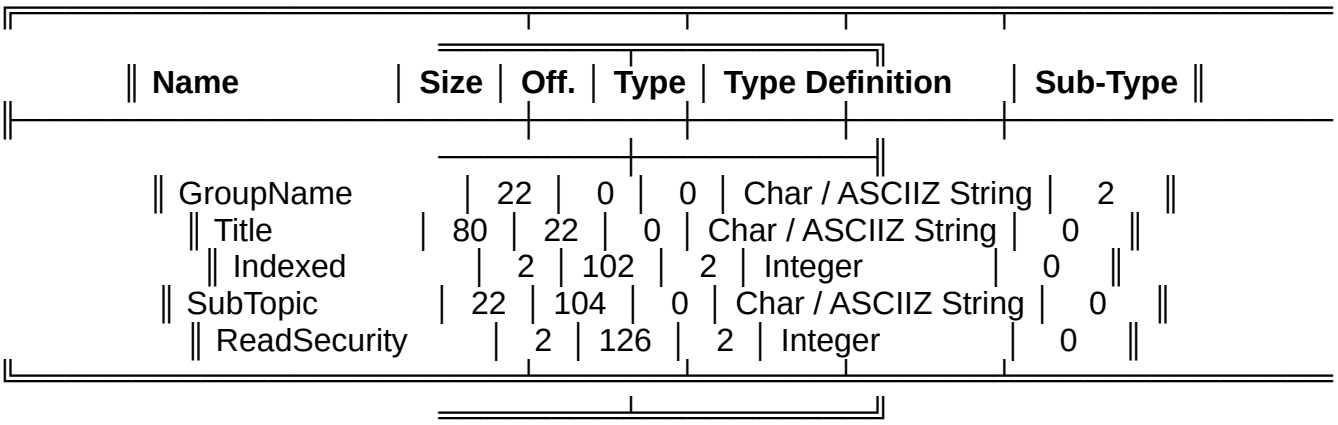
ErrorMessage section of the Charts Database, fill in the blanks, and send that message to the user and/or the file. This helps pinpoint what specific step was occurring when the error occurred.

Details on the meanings of the Message and Text field contents can be found in Appendix 2: the Error System, which starts on page . This contains a list of error message meanings, along with recommended remedial actions to take.

Secondary Data Structures

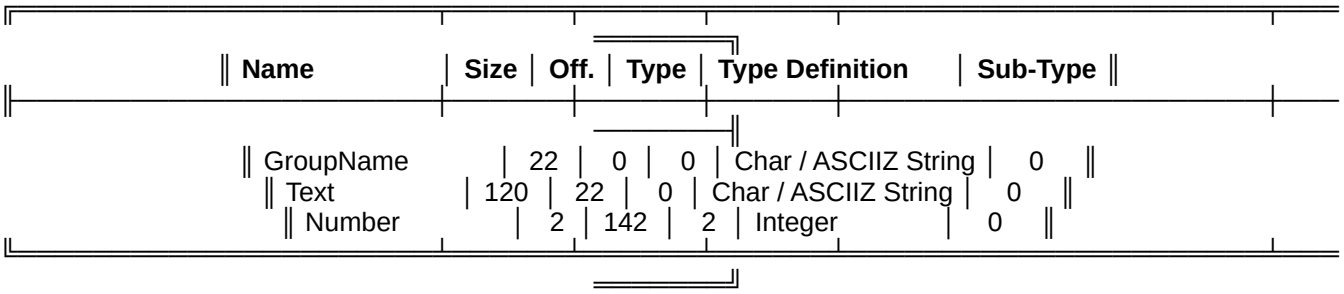
Help System

Help Database



Name	Size	Off.	Type	Type Definition	Sub-Type
GroupName	22	0	0	Char / ASCIIZ String	2
Title	80	22	0	Char / ASCIIZ String	0
Indexed	2	102	2	Integer	0
SubTopic	22	104	0	Char / ASCIIZ String	0
ReadSecurity	2	126	2	Integer	0

HelpField Database



Name	Size	Off.	Type	Type Definition	Sub-Type
GroupName	22	0	0	Char / ASCIIZ String	0
Text	120	22	0	Char / ASCIIZ String	0
Number	2	142	2	Integer	0

The Help System uses two databases to both facilitate data entry, and to provide the user with choices of help topics. The Help database provides information on the Title, SubTopic, and Index status of a particular help screen. The HelpField Database contains the 'meat' of the information that will fill the help window.

Help

GroupName The primary field: contains the name that binds all associated HelpFoot records together.

Title The title to place on the Help Window when this help is being displayed, and the text to display to represent this Help in a picklist of help topics.

Indexed If non-zero, this help topic is on the "Master Index".

SubTopic This contains another group field. Several Help records can be related by having the same text in the SubTopic Field. This is used to generate a picklist of 'Related SubTopics' for the current help topic being displayed.

ReadSec'ty The minimum security level required to read this help topic.

Secondary Data Structures

HelpField

GroupName The primary field: binds all HelpFoot records with the same Group Name to a related HelpHead record.

Text The contents of a line on the help window for this topic. This field may also include color and cross-reference designations (see below).

Order This integer sorts the help text, so that the lines always display in the same order.

Secondary Data Structures

Help System

Help System Color Designations

The Help System allows text to be displayed in up to 14 simultaneous colors. This helps to emphasize and clarify the information for the user. Coloring is controlled by a slash (/) character, followed by the number of the color to be used. Each color control acts as a toggle to the default color: a second occurrence of a color will reset the following text to the background color.

Example of help text coloring:

For the 'display' on this page, assume that:

Color 1 is normal text.
Color 2 is underlined text.
Color 3 is **bold** text.

The help text line is:

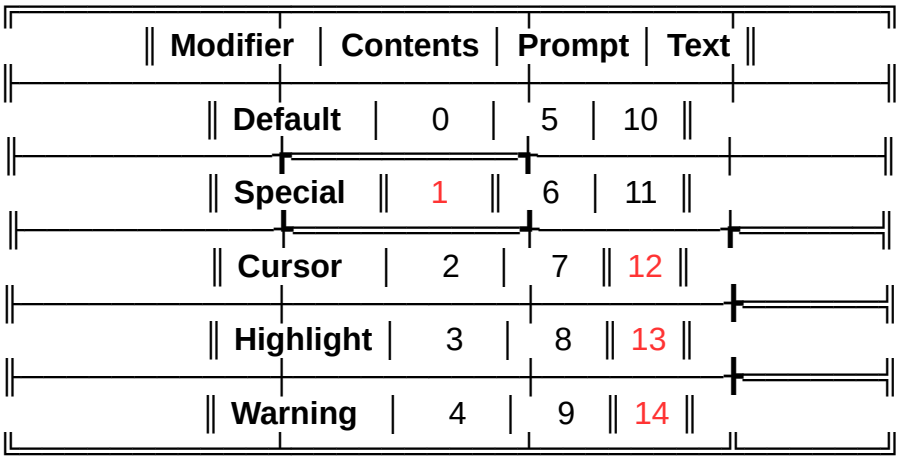
The /2information/2 will /3display /2as if/2 it were/3 different.

The help text will 'display' as:

The information will **display** as if **it were** different.

The color characters will not take up a column of display space, but may shorten the available space at the end of the line. For this reason, it is recommended to keep displays under 90 columns, to allow ample room for coloring within each line.

The color scheme used is determined by the "HelpColors" field in the UserData database. The color scheme determines the actual colors in use. Unless otherwise directed, the help system will draw all text in color 1: Special Contents. The colors' numeric designations are as follows:



Colors 1, 12, 13, and 14 are boxed because they represent special uses of the help colors.

Secondary Data Structures

Help System

Color 1, the Special Contents color, is the default color for text in the help system. Colors 12 and 13, the Text Curstor and Highlighted Text color, are the colors used for cross-references (12 is default color, 13 is selected color). Color 14, the Warning Text color, is also the color of the help window's title. Therefore, colors 1 and 14 are redundant, and colors 12 and 13 can be confusing if cross-references exist (see next page).

Secondary Data Structures

Help System

Help System Cross Referencing

Cross-referencing provides the help-system developer with a tool for generating both specific cross references and topical indexes, allowing the user to get more detailed help on a topic without resorting to the Master Index each time.

A cross-reference is defined by marking certain text as if it were color 'i', that is to say that text must be enclosed by /i. This text must comprise a group name of some other help. The result is that the user can tab to the cross-referenced text, press <Enter>, and the help named by the cross-reference will appear. The user can also press the 'Topical Index' key and get a picklist of all cross-referenced text in this help topic.

Example of cross-referencing:

There are three, related help topics:

Birds
Fowl
Chickens

The 'Birds' topic could have a text line that contains:

See Also: /iFowl/i Specific Example: /iChickens/i

Which would display as:

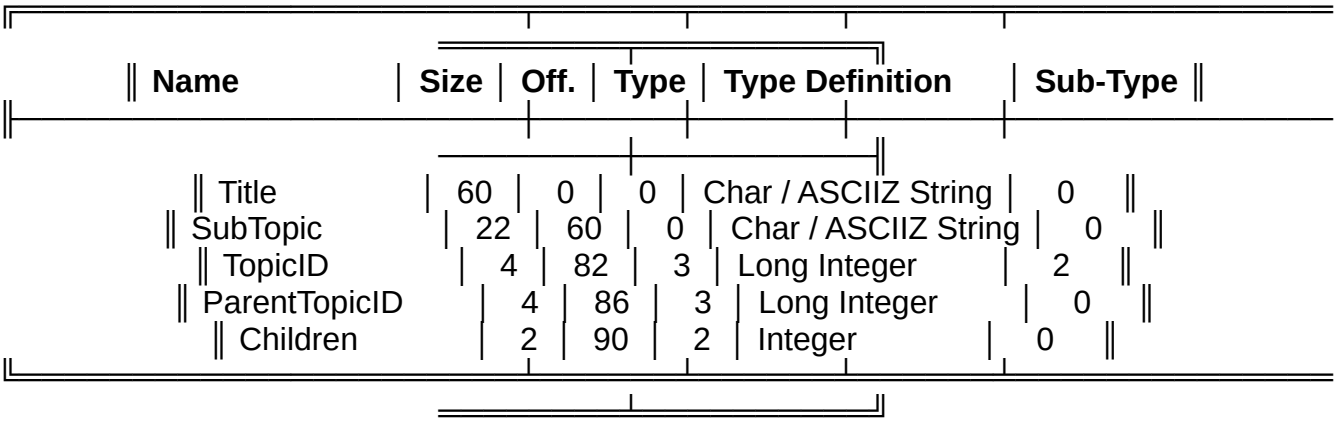
See Also: Fowl Specific Example: Chickens

Pressing <Enter> on either Fowl or Chickens would load that help topic, while the SubTopic Index Key would make a picklist containing the titles of both the "Fowl" and "Chickens" HelpHead records.

Secondary Data Structures

Help System

HelpTopic Database



Name	Size	Off.	Type	Type Definition	Sub-Type
Title	60	0	0	Char / ASCIIZ String	0
SubTopic	22	60	0	Char / ASCIIZ String	0
TopicID	4	82	3	Long Integer	2
ParentTopicID	4	86	3	Long Integer	0
Children	2	90	2	Integer	0

Another way to access additional help information is through the Categorical HelpTopic Database. It is accessed through the F8 key when the user is in the help system. It brings up a PickList of all Topic Record's titles that have no ParentTopicID (value of 0). The user's selection lists all Topic Record's titles that have the selected record's TopicID as a ParentTopicID. This system allows for extremely detailed cross-categorization within the help system.

Title Text displayed in Categorical PickLists

SubTopic SubTopic of Helps to list if this topic, when selected has no children.

TopicID Unique number identifying this particular Topic record.

ParentTopicID The TopicID number of the record that is the "parent" of this record.

Children A yes/no field, where zero means no, indicating if this topic has "children" topic records that are available to display.

Example of topic categorization:

Expanding on the example used for cross-referencing, there are several semi-related topics.

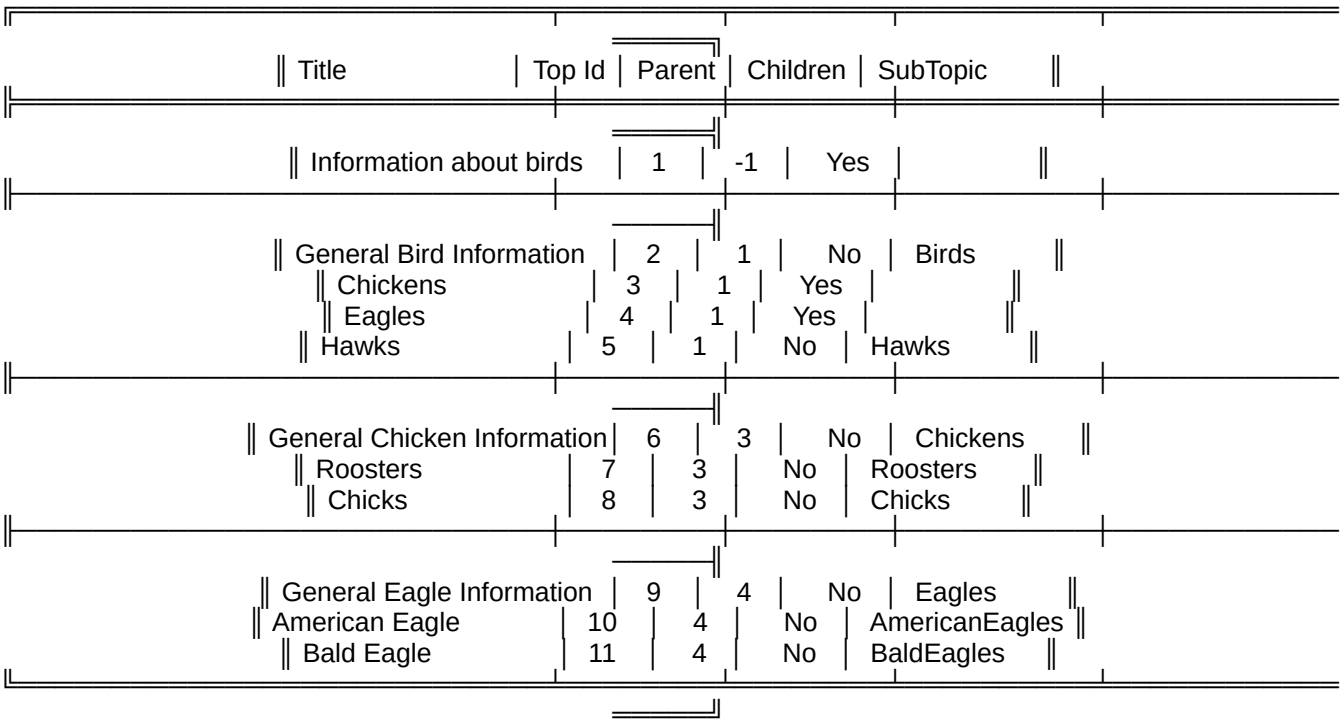
"Birds" is the Main Topic
- "Chickens" refers to a kind of bird
  - "Roosters" refers to a kind of chicken
  - "Chicks" refers to a kind of chicken
- "Eagles" refers to a kind of bird
  - "American Eagles" refers to a kind of eagle
  - "Bald Eagles" refers to a kind of eagle
- "Hawks" refers to a kind of bird

This example, if expanded and deepened sufficiently, could extend the number of 'see also' cross-references beyond the scope of a PC's available memory, and the number of references to read through would distract the user from the original goal. Categoric topics allow the one, long list to become several smaller, more useful lists in a hierarchy.

Secondary Data Structures

Help System

Suggested HelpTopic layout:



With these records in the HelpTopic database, the user gets a Main Topic List consisting of "Information About Birds". Selecting this topic brings a new Topical PickList of "General", "Chickens", "Eagles", and "Hawks". Selecting General would bring up a PickList of all help records with "Birds" as a SubTopic. If, instead, the user selected "Chickens", a Topical PickList of "General", "Roosters", and "Chicks", all of which pull up PickLists of Help records with appropriate subtopic records.

Secondary Data Structures

Help System

Function Keys in the Help System

When the help window is current, the end user will have several standard options:

<Esc> Quits help, and returns to the location where help was requested.

<Arrows / Page keys> move the help window around on the help system.

<Tab> "Jumps" the help window to the first or next available cross-reference.

<sh-Tab> "Jumps back" to the last or previous available cross-reference.

<Enter> Loads and displays the help currently highlighted in the cross-reference.

<F1> Brings up "Help on Help".

<F2> Recalls the Previous Help that was called

<F3> Calls the next help (assuming that <F2> was used previously)

<F4> Generates a PickList of all Help Topics, sorted by description. This allows the user to get general help on any listed subject.

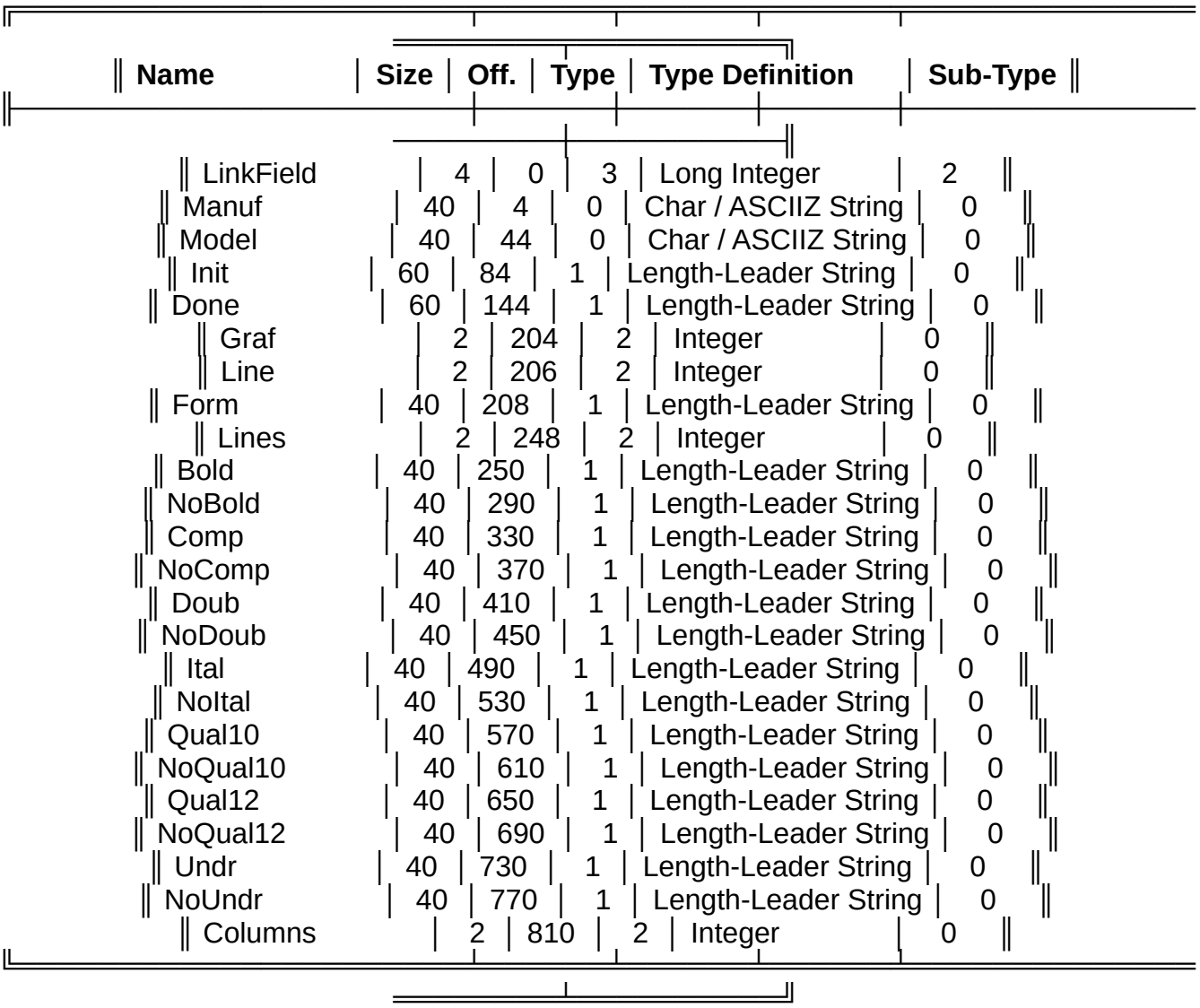
<F5> Generates a PickList of Indexed SubTopics: This is a list of all topics with a 'Yes' in the "Indexed" field of the Help database.

<F6> Topical help: Generates a PickList of all Topics with ParentID's of 0, which will continue loading SubTopic PickLists until the user selects a Topic containing a "No" in the "Children" field, and a valid Help SubTopic. This helps to systematically organize the help available to the user.

<F7> Generates a PickList of all Help records with the same SubTopic.

Secondary Data Structures

Printers Database



Name	Size	Off.	Type	Type Definition	Sub-Type
LinkField	4	0	3	Long Integer	2
Manuf	40	4	0	Char / ASCIIZ String	0
Model	40	44	0	Char / ASCIIZ String	0
Init	60	84	1	Length-Leader String	0
Done	60	144	1	Length-Leader String	0
Graf	2	204	2	Integer	0
Line	2	206	2	Integer	0
Form	40	208	1	Length-Leader String	0
Lines	2	248	2	Integer	0
Bold	40	250	1	Length-Leader String	0
NoBold	40	290	1	Length-Leader String	0
Comp	40	330	1	Length-Leader String	0
NoComp	40	370	1	Length-Leader String	0
Doub	40	410	1	Length-Leader String	0
NoDoub	40	450	1	Length-Leader String	0
Ital	40	490	1	Length-Leader String	0
NoItal	40	530	1	Length-Leader String	0
Qual10	40	570	1	Length-Leader String	0
NoQual10	40	610	1	Length-Leader String	0
Qual12	40	650	1	Length-Leader String	0
NoQual12	40	690	1	Length-Leader String	0
Undr	40	730	1	Length-Leader String	0
NoUndr	40	770	1	Length-Leader String	0
Columns	2	810	2	Integer	0

The Printers Database allows users and developers to add or customize a variety of settings for multi-font reports. The printer record to use is generated by either the Printer Number field in the UserData database record for the current user, or specified in the ReportAs...() series of ReportCall commands. This allows for the same multi-font report to be sent to different types of printers without editing the report.

LinkField A unique number that will identify the printer record to the report system.

Manuf The text name of the printer's manufacturer (optional).

Model The text name of the printer's exact model (optional). Many users on network systems will use the Manuf/Model combination to specify who or where the printer is, such as Tom's Laser, or Rick's LinePrinter.

Init This is a code or series of codes that is sent to initialize the printer before each

report is run.

Done This is a code or series of codes that is sent to the printer after each report is run, to 'de-initialize', or 'reset' the printer, if necessary.

Secondary Data Structures

Printers Database

Graf This is a numeric code that indicates if character translations should be performed when sending IBM PC Graphic (box characters) to the printer.

Val.	Meaning of Graf Setting
0	No Translation
1	Epson Graphic Character codes
2	Translate to lower ASCII
3	Convert to spaces: blank out

Line This is a numeric code that indicates how to advance the printer to the next line, and is especially useful for older printers, that may not have a compatibility switch on them.

Val.	Meaning of Line Setting
0	Normal: Send Carriage Return and Line Feed
1	Auto LF: Send Carriage Return Only
2	Auto CR: Send Line Feed Only

Form This is a code or series of codes used to advance to the top of the next page (form feed).

Lines The number of lines on a page in default printing mode (Example: 8 1/2 x 11 page has 66 lines at 6 lines per inch, which is fairly standard). This setting can override the master page setting in ReportSafe mode.

Columns The number of columns on a page in default printing mode (Example: 8 1/2 x 11 page can actually only fit 80 characters at 10-pitch, since most printers automatically provide a 1/2" page edge margin.) This setting can override the master page setting in ReportSafe mode.

The remainder of the fields are 'toggle codes': when the report generator sees a font code, it toggles the value and inserts the appropriate code into the report.

Example of toggle code:

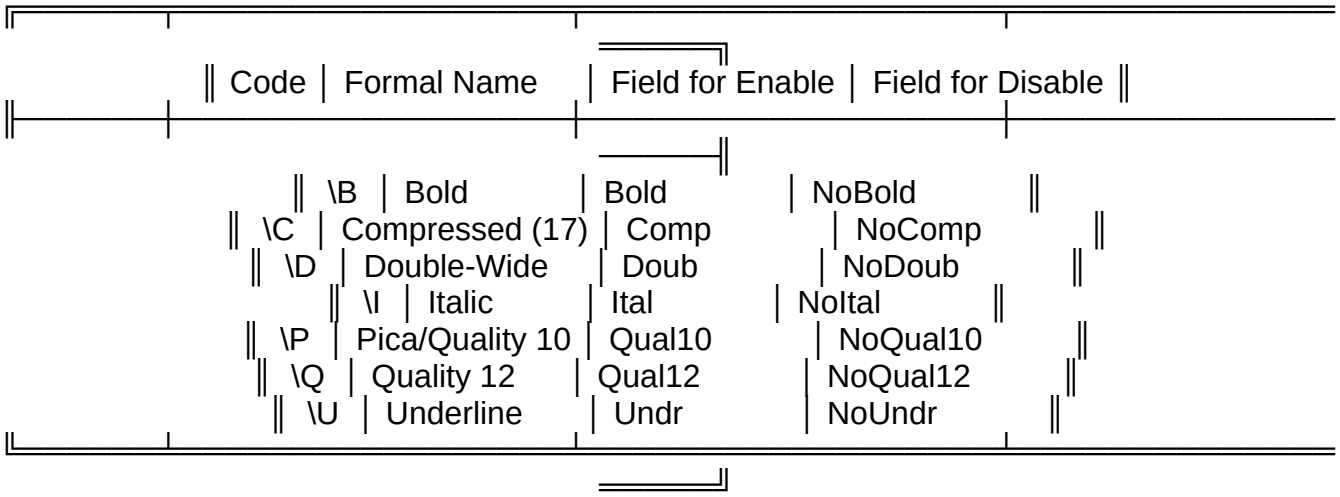
> Report has a prompt of "\BHello\B world" (\B is the Bold Toggle Code).
>
> Report will print as "<Bold>Hello<NoBold> world", or "**Hello** world".

On the following page is the complete list of font toggle codes and their corresponding fields in the Printers Database.

Secondary Data Structures

Printers Database

Font Toggle Codes



Code	Formal Name	Field for Enable	Field for Disable
\B	Bold	Bold	NoBold
\C	Compressed (17)	Comp	NoComp
\D	Double-Wide	Doub	NoDoub
\I	Italic	Ital	NoItal
\P	Pica/Quality 10	Qual10	NoQual10
\Q	Quality 12	Qual12	NoQual12
\U	Underline	Undr	NoUndr

When a report starts, VariForm assumes that all codes are disabled, regardless of the actual status of the printer. That is why the INIT code often resets the printer: to make sure that the printer is actually in the appropriate mode at the beginning of the report.

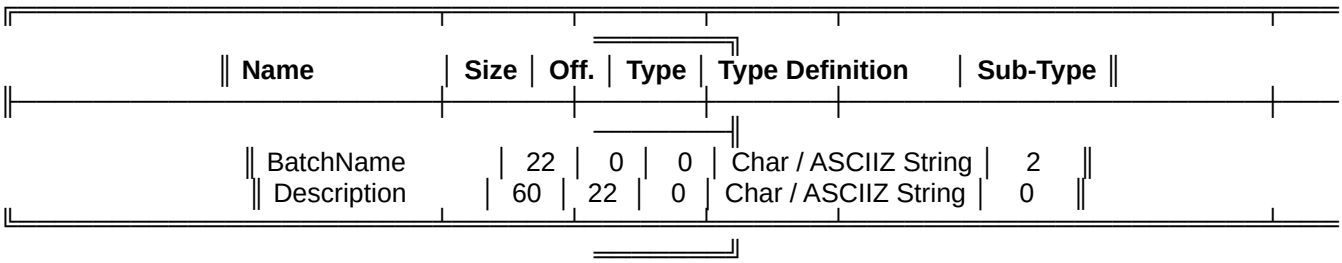
Once a report begins to generate, the font codes' statuses are kept, so it is best to make sure that all font codes in a report are paired, to prevent random fluctuations in font size or style.

Further, some users may want to set these codes to other values, such as color settings for color printers. The key point to keep in mind is that the 'Disable' field should undo only the settings established by the corresponding 'Enable' field. i.e.: The NoBold should not reset the printer to 10 pitch.

Secondary Data Structures

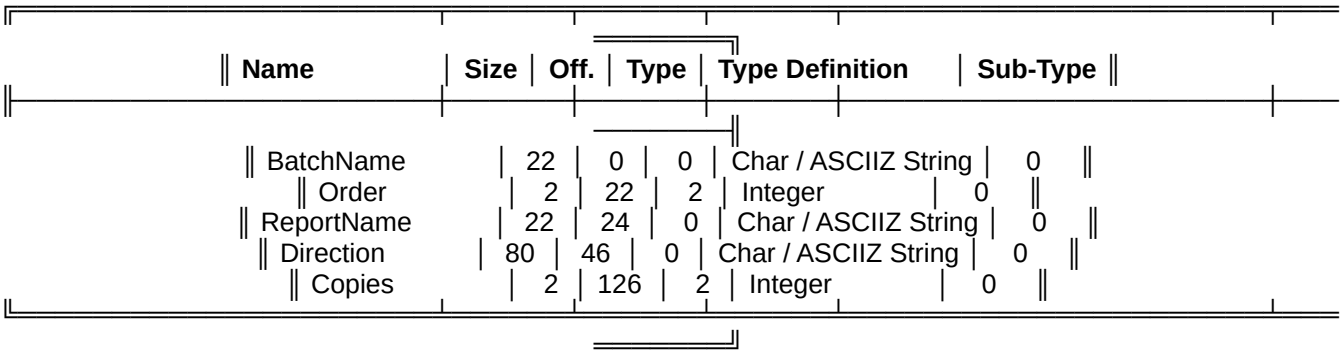
Report Batches

ReportBatch Database



Name	Size	Off.	Type	Type Definition	Sub-Type
BatchName	22	0	0	Char / ASCIIZ String	2
Description	60	22	0	Char / ASCIIZ String	0

ReportBatchField Database



Name	Size	Off.	Type	Type Definition	Sub-Type
BatchName	22	0	0	Char / ASCIIZ String	0
Order	2	22	2	Integer	0
ReportName	22	24	0	Char / ASCIIZ String	0
Direction	80	46	0	Char / ASCIIZ String	0
Copies	2	126	2	Integer	0

Once reports have been designed, it is common to need several reports, or several copies of certain reports. VariForm contains a Batch Reporting System to service these needs. Batch reporting allows a User or a System Designer to assign a 'Batch Name' to a group of reports, with various numbers of each report printing. The user can generate the entire series of reports directly, through the Command Line Interface, or from within a procedure by using a "CallBatch(BatchName)" command.

To simplify data entry in the Batch Reporting System, VariForm provides a ReportBatch database, which contains the BatchName, and a text description of what the BatchName is used for.

BatchName Connects the ReportBatch record to the appropriate ReportBatch Title record.

Order Organizes numerically the order in which the reports will be generated.

ReportName The name of the Primary Page Definition (type 2 DisplayStruct) record.

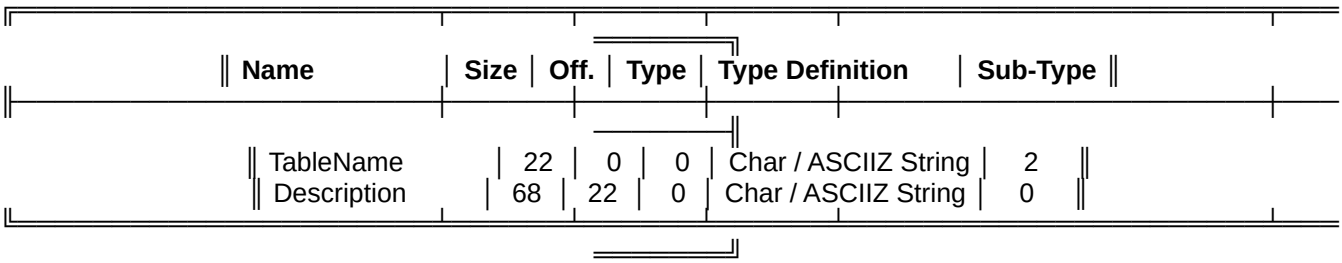
Direction The filename or path where this report will be sent. This can be overridden by the batch call's 'Override' field.

Copies The number of copies of this report to generate.

Secondary Data Structures

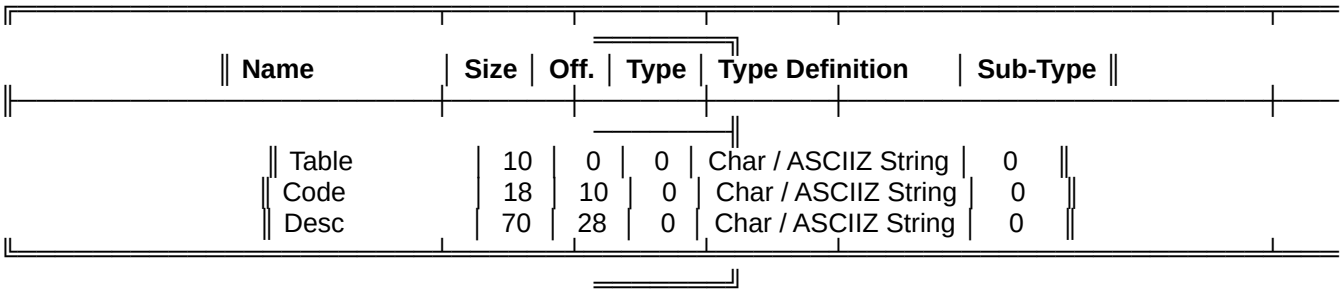
Table of Codes

Table Database



Name	Size	Off.	Type	Type Definition	Sub-Type
TableName	22	0	0	Char / ASCIIZ String	2
Description	68	22	0	Char / ASCIIZ String	0

TableField Database



Name	Size	Off.	Type	Type Definition	Sub-Type
Table	10	0	0	Char / ASCIIZ String	0
Code	18	10	0	Char / ASCIIZ String	0
Desc	70	28	0	Char / ASCIIZ String	0

To simplify data entry in the Table of Codes, VariForm provides a Table database, which contains the Table Name and a text description of what the TableName is used for. Table Field records are used to provide text descriptions of abbreviations stored in the "Code" field.

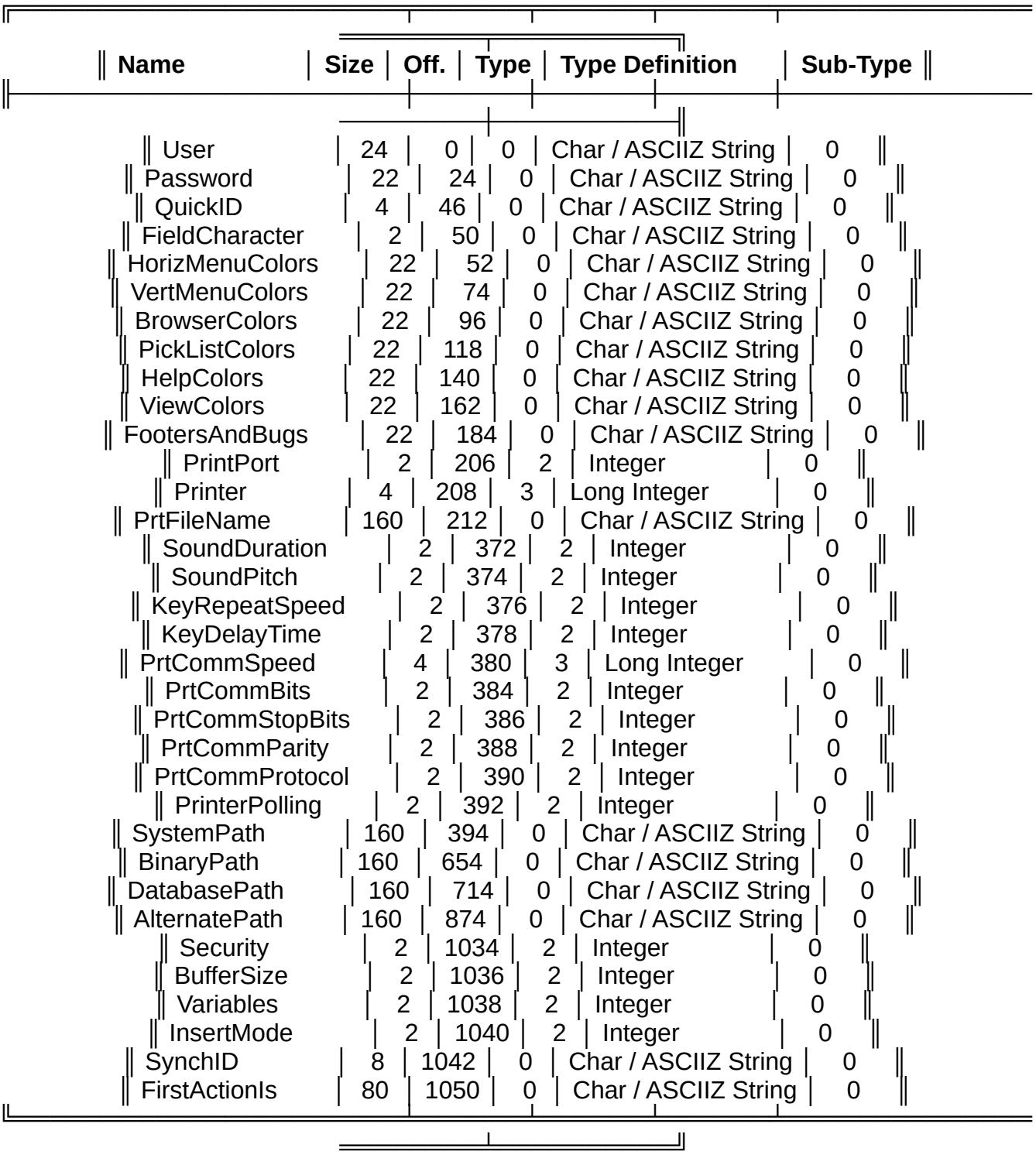
There is a one-to-many relationship between the Table database and the TableField database, which connects 'TableName' in the former to 'Table' in the latter.

Example: There is a list of the text descriptions of the U.S. State abbreviations found in the CodeName 'state'. That means that each appropriate entry in the TableField database will contain 'state' in the Table field, and an appropriate Code and Name combination. This way, the value 'PA' in the colors chart refers to 'Pennsylvania'. 'PA' may have other meanings in other tables, but not in the state table.

This is also a good example of sub-grouped uniques, which are unique fields in a group within a database, but which may be duplicated in other parts of the same database.

Secondary Data Structures

UserInfo Database



Name	Size	Off.	Type	Type Definition	Sub-Type
User	24	0	0	Char / ASCIIZ String	0
Password	22	24	0	Char / ASCIIZ String	0
QuickID	4	46	0	Char / ASCIIZ String	0
FieldCharacter	2	50	0	Char / ASCIIZ String	0
HorizMenuColors	22	52	0	Char / ASCIIZ String	0
VertMenuColors	22	74	0	Char / ASCIIZ String	0
BrowserColors	22	96	0	Char / ASCIIZ String	0
PickListColors	22	118	0	Char / ASCIIZ String	0
HelpColors	22	140	0	Char / ASCIIZ String	0
ViewColors	22	162	0	Char / ASCIIZ String	0
FootersAndBugs	22	184	0	Char / ASCIIZ String	0
PrintPort	2	206	2	Integer	0
Printer	4	208	3	Long Integer	0
PrtFileName	160	212	0	Char / ASCIIZ String	0
SoundDuration	2	372	2	Integer	0
SoundPitch	2	374	2	Integer	0
KeyRepeatSpeed	2	376	2	Integer	0
KeyDelayTime	2	378	2	Integer	0
PrtCommSpeed	4	380	3	Long Integer	0
PrtCommBits	2	384	2	Integer	0
PrtCommStopBits	2	386	2	Integer	0
PrtCommParity	2	388	2	Integer	0
PrtCommProtocol	2	390	2	Integer	0
PrinterPolling	2	392	2	Integer	0
SystemPath	160	394	0	Char / ASCIIZ String	0
BinaryPath	160	654	0	Char / ASCIIZ String	0
DatabasePath	160	714	0	Char / ASCIIZ String	0
AlternatePath	160	874	0	Char / ASCIIZ String	0
Security	2	1034	2	Integer	0
BufferSize	2	1036	2	Integer	0
Variables	2	1038	2	Integer	0
InsertMode	2	1040	2	Integer	0
SynchID	8	1042	0	Char / ASCIIZ String	0
FirstActionIs	80	1050	0	Char / ASCIIZ String	0

VariForm provides users with a large variety of options, and to run, several questions must immediately be answered: How much memory should be set aside for Variables, for internal disk buffering? What directories are the databases stored in, what printer, and what type of printer interface should be used as a default? The UserInfo System provides the answers for these questions on behalf of the user.

Secondary Data Structures

When first installed, VariForm contains 1 UserInfo record, with the name 'System Default', and no password. When there is only one record in the UserInfo database, the system will start automatically. For logon security, the record should at least be edited, and all UserInfo Records should contain passwords. Unless there is more than 1 data record, VariForm will automatically start up. There must be more than one record for the mandatory 'Login' screen to appear.

The UserInfo fields are arranged by topic, instead of structure order, for clarity.

Secondary Data Structures

UserInfo Database

General Information

User "User Name" for login screen and ErrorAction Database Entries

QuickID A 3-character abbreviation for the user. This is used in the Error Log.

Password Applies only to the login screen.

FieldCharacter Default character for Data Entry screens that indicates the portion of the current field that does not contain data. This is overridden by Color Database information.

SoundDuration Length, in milliseconds, of the system beep, for error messages.

SoundPitch Frequency, in Hertz, of the system beep (Lower number=lower pitch).

KeyDelayTime When a key is held down for a period of time, it begins to repeat itself, as if the user was pressing the key over and over again. The length of the delay before repetition begins is called the "key latency period", and is controlled by this field. The larger this number, the longer the delay before the key begins repeating.

KeyRepeatSpeed Once a key is held down past the latency period, how quickly should it repeat chacters. Lower numbers represent faster repeat speeds.

Security This number, or assortment of bitfields, is the assigned security value, or the group of security permissions available to the user. This field is ignored unless the Security Module is used.

BufferSize Size, in KiloBytes, of the built-in disk cache. Acceptable values are between 4 and 200; values outside this range will either disable the cache (lower values), or disable VariForm (insufficient memory problems with higher values).

Variables Size, in KiloBytes, of the Variable content area. Acceptable values are between 4 and 62. Lower values will prevent VariForm from running properly (out of variable space).

InsertMode Automatic cursor mode: should the user always start editing a data entry screen in overwrite mode (0), insert mode (1), or should the system 'remember' the mode from the last data entry screen used (2).

FirstActionIs This field allows each user to see something different at login. If left blank, the user will see the "Main" menu, as if this field contained "Menu("Main")". Otherwise, the user will see whatever their FirstActionIs command shows them.

HorizMenuColors If used, this color scheme will override the chosen color scheme for all horizontal menus.

VertMenuColors If used, this color scheme will override the chosen color scheme for all vertical menus.

BrowserColors If used, this color scheme will override the chosen color scheme for all Browsers, including BrowserLists.

PickListColors If used, this color scheme will override the chosen color scheme for all PickLists.

HelpColors If used, this color scheme will override the chosen color scheme for the Help System's display screens.

ViewColors If used, this color scheme will override the chosen color scheme for the Text Editor's display screens.

FootersAndBugs Mandatory: this color scheme determines what colors are used for the login cue lines, as well as for the Error Message Displays and System Inquiries.

Secondary Data Structures

UserInfo Database

Printer/Report Generator Defaults

PrintPort If the communications module is installed, this determines the default output mode for reports. Without the module, or if the CommSession variable is off, the default output mode for reports is "To File of name (UserFileName)".

PrtFileName Default filename to use in sending reports to a file.

Printer The 'LinkField' number signifying which printer is in use by default.

PrinterPolling When sending reports to a 'hardware port', such as a PrintPort 0, 1, 2, or a Comm Port, this field acts as a 'Yes/No' field determining whether or not the port provides a 'PortReady' interrupt. If printing to a file, this field is ignored. Some computer hardware will 'hang' or freeze up when this field is 'Yes'.

Printer Defaults for Printing to a Serial Port
(all of these settings require the Communications Module)

PrtCommSpeed Communications speed in bits-per-second (baud) when sending reports to a hardware serial port.

PrtCommBits Data Bits (7 or 8) to use when sending reports to a hardware serial port.

PrtCommStopBits Stop Bits (0, 1, or 2) to use when sending reports to a hardware serial port.

PrtCommParity This number determines what communications Parity to use when sending reports to a hardware serial port. Parity is a crude, character-by-character error recognition protocol.

PrtCommProtocol This number determines what 'hardware' communications protocol to use to determine when the serial port can receive characters.

Database Paths (Directories)
(can be overridden by full path (including drive) in Database of Databases Record)

DatabasePath This is the path for all databases that have neither the "System" nor the "Binary" flag set (both flags are "No").

SystemPath This is the path for all "System" databases, once VariForm has started the AutoLoad process.

BinaryPath This is the path for all "Binary" databases, once VariForm has started the AutoLoad process.

AlternatePath This is the patch for all databases that have both the "System" and the "Binary" flags set to "Yes".

Secondary Data Structures

Virtual Field System

VirtualGroup Database

Name	Size	Off.	Type	Type Definition	Sub-Type
SearchName	22	0	0	Char / ASCIIZ String	2
Title	60	22	0	Char / ASCIIZ String	0
LoadSecurity	2	82	2	Integer	0
EditSecurity	2	84	2	Integer	0

VirtualField Database

Name	Size	Off.	Type	Type Definition	Sub-Type
SearchName	22	0	0	Char / ASCIIZ String	0
Order	2	22	2	Integer	0
Name	22	24	0	Char / ASCIIZ String	0
Length	2	46	2	Integer	0
Type	2	48	2	Integer	0
SubType	2	50	2	Integer	0
InitValue	180	52	0	Char / ASCIIZ String	0
ReadSecurity	2	232	2	Integer	0
EditSecurity	2	234	2	Integer	0

VariForm has a method of loading system information that can be modified more quickly than a database record. This is the primary purpose of the Virtual Field System: providing groups of variables for VariForm its Procedures to use and manipulate quickly and effectively.

To simplify data entry in the Virtual Fields System, VariForm provides a VirtualGroup database, which contains the SearchName, a text description of what the Fields are used for, and security fields to prevent unauthorized loading or modification of Virtual Fields.

There is a one-to-many relationship between the VFTitle database and the VirtualFields database, which connects records with the same contents in the 'SearchName' fields.

When VariForm is started, it automatically loads the Virtual Field Group with the SearchName of "Main". Through the "LoadVirtuals(SearchName)" and "KillVirtuals(SearchName)" commands, Procedures and users may load, manipulate, and discard large groups of variables without the necessity of performing an individual "Assign()" and "Remove()" command on each

variable.

Secondary Data Structures

Virtual Field System

Meanings of VirtualField Record's Fields

SearchName The primary search field, associating all virtual fields with the same name together, and relating them to the appropriate VFTitle record.

Order The numeric order in which the VirtualField records will load within a SearchName group.

Name The actual name of this field. This should be a unique name within the list of all virtual fields that will be in the system concurrently, as well as the field names within the 'current' database, being edited, searched, or browsed at the time.

Length As a database field: the size, in bytes, of the data in this field.

Type As a database field: the type of data contained in this field.

SubType As a database field: the specific details about data contents. Note: SubTypes in Virtual fields are ignored, except during character translations. All references to 'unique' and 'union' values are ignored.

InitValue What value should be loaded into this field when the 'load virtuals' command is issued. Note: see the 'System Variables' section, starting on page about the DateFormat Variable to determine how to enter initvalues for date fields: date fields' initvalues must be entered in the appropriate order for the value to be loaded correctly.

ReadSec'ty Minimum Security required to display the contents of this Virtual Field in the CommandLine Interface.

EditSec'ty Minimum Security required to change the contents of this Virtual Field through the CommandLine Interface.

The CommandLine System

VariForm has a built-in, function-oriented programming language called "the CommandLine System". This system allows VariForm to perform very sophisticated tasks without using a compiler.

The CommandLine System takes input from a variety of sources, processes it, and returns the results. These inputs include:

Search System search fields of any type
Display System's MetaFields, for text fields in Menu mode
Function Key Action lines
Calls from the Procedure Processor
The Command Line Interface

Most functions and features of the CommandLine system are universally available, but some functions require input that can not be readily typed in. These are called "Internal System Functions", and can only be accessed under certain conditions (such as being called from a data entry screen or browser), and can not be called from the Command Line Interface. Wherever possible, similar functions are available to the Command Line Interface with a similar name.

The CommandLine System also functions as a procedural language via the 'procedures' database, providing if/elseif/else branching, and do/while, while/do loops. Procedures are loaded via the 'Call()' and 'IfCall()' functions, when they are processed by the CommandLine system.

Scientific Trivia: VariForm uses a unique method of translating complex commands into actions, called 'dual loop linear evaluation'. This offers two distinct advantages over recursive or "Reverse Polish Notation" processing. Linear evaluation does not use the system stack for single-line evaluations, which makes complex statements 'safer' to operate at deep levels of operation. Linear evaluation also cleans up its memory after each line, meaning that definition errors do not generally cause 'daisy chain' errors beyond the following line.

This same evaluation process, applied to loops and branches, allows VariForm to provide hundreds of levels of loop and parenthetic access without the need for hundreds of Kilobytes of precious Ram tied up in a system stack. In general, it means safer, smaller functions and macros without the high system requirements of recursive systems.

The CommandLine System

Command Line Interface

The Command Line Interface is actually a line-by line interpretation of non-internal system functions. The same commands that work here are available as function key commands, menu selection actions, search system actions, and may be lines of 'code' in a procedure.

This system forms the basis of the procedural language for all non-compiled (i.e.: user customizable) VariForm applications.

Variables: The system provides a virtual database: "Virtual", which contains a variable number of "Database Fields". These fields are initialized at runtime for the VirtualField database group "Main". Other fields can be declared and removed as needed during system operations through the 'LoadVirtual' and 'KillVirtual' functions. Browsers and picklists cannot view the virtual database, since there are no file handles or records associated with it, but the Data Entry, Menu, and Reporting system can use this "database" for direct entry to system variables.

A psuedorecord containing the values of the variables is allocated at runtime. Its size is controlled by the user setting: Var Size. 16K should be more than enough room for most applications, but the system can reserve up to 64K, if necessary. Note that this is the size reserved for the Variables' contents; their names and other references use system memory. Since the memory space is reusable, this is indeed a large area to work within.

Variables can be any legitimate field type: integer, character, long int, floating point, time, or date. Variables are created in one of two ways: they are loaded by the search & load system from the "Virtual Fields" database when appropriate, or a user can "Declare" or "Remove" a variable explicitly from any level of the Command Line Interface.

Variables are associated together by either a "group name", or by the explicit level of command line interface in which they were declared. They should always have system-wide unique names, to prevent the system from misinterpreting where to get information for the CommandLine system. They should also never have the name of a function, or the CommandLine System Interpreter will not consider them as variables.

When VariForm is run, all variables with the group name 'Main', are loaded into the virtual system. These variables will be automatically removed when the program is terminated. Variables loaded from a data entry screen, browser or report precondition will be unloaded when the screen, browser, or report is closed.

Variables that are declared within the Command Line Interface will be automatically removed when the command line interface is terminated (blank line or escape key).

Variables can be used without discretion within command lines . They can be tested, assign values, pass parameters, et cetera. If an undeclared variable is used, the command line interface will interpret the characters as a discrete constant.

Discrete constants may only be character, floating point, date, or time, based on the existance of characters, or slash marks in the initializer statement. It is best to avoid these constants whenever possible. The rule for analyzing a discrete constant:

If it is not a function name, nor a virtual field name, and
If it is a non-internal, or not a current database field name:

If it starts with a number:
If it contains a backslash (\), it is a date var
Else If it contains a colon (:), it is a time var
Else If it contains all numbers, (incl +-.), it is numeric
Else it is a character string
Else it is a character string

Discrete constants cannot receive values, since they are values themselves, so assigning values to them will cause a non-critical system error. For example, if the system had some character string variables called #Text1# and #Text2#, with #Text2# containing the value "book":

#Text1# = #Text2# Would put the string "book" into #Text1#

#Text1# = #Text2 Would put the string "#Text2" into #Text1#

#Text1 = #Text2# Would cause a system error, since "#Text1" is a discrete constant, not the name of a variable

Basic Math Operations

All basic math functions (binary operators between the two target values), use the first, or 'left-hand' value to determine the target values' types. Return values are numeric, and can be assigned or ignored.

= Assign a value to a variable. Returns -1.

+ Add values together. Characters are appended as strings, Dates are added to a Number of days, or if the second value is a date, Number or years, Number of months and days.

- Subtract values: as addition, but no string values are permitted.

***** Multiply values: only for numbers. Non-numerics will interpret as zero. Dates will interpret based on the DateFormat Value (whichever value is first).

/ Divide values: only for numbers 0 denominator makes the return value zero.

% Modulus: return the remainder of a / b. 0 denominator makes the return value zero.

All functions and operators return floating point numbers with the 32-bit accurate value of the operation performed.

| "inclusive or" of 2 numeric values: (1 | 4 == 5), (2 | 2 == 2)

^ "exclusive or" of 2 numeric values (1 ^ 4 == 5), (2 | 2 == 0)

& "and" of 2 numeric values (1 & 4 == 0), (2 & 2 == 2), (1 & 5 == 1)

lsl Logical shift left: Shift a left b bits in a c-bitsize region. In a logical shift, bits that exceed the region size are dropped. VariForm allows 1 - 32 bit logical bit regions.

lsr Logical shift right: Shift a right b bits in a c-bitsize region. In a logical shift, bits that exceed the region size are dropped. VariForm allows 1 - 32 bit logical bit regions.

rol Rotate Left: In a rotation, bits that exceed the region are reentered at the other end : (rol(2, 2, 3) == 1). VariForm allows 1 - 32 bit logical bit regions.

ror Rotate Right: In a rotation, bits that exceed the region are reentered at the other end : (ror(2, 2, 3) == 4). VariForm allows 1 - 32 bit logical bit regions.

The CommandLine System

CommandLine Functions

Trigonometry Functions

The trigonometry functions work in radians, as per the IEEE standard for 64-bit floating point precision operations involving the non-rational, real value of pi. There are also functions defined to perform conversions between degrees and radians with very little effort. VariForm uses fractional degrees, which retranslate into degree, minute, second format, with only a .001 second margin of rounding error.

cos	Cosine of (a), where a is in radians
sin	Sine of (a), where a is in radians
tan	Tangent of (a), where a is expressed in radians
acos	Arc Cosine of (a), in radians
asin	Arc Sine of (a) in radians
atan	Arc Tangent of (a) in radians
cosh	Hyperbolic cosine of (a), where a is in radians
sinh	Hyperbolic sine of (a), where a is in radians
tanh	Hyperbolic tangent of (a), where a is expressed in radians
atan2	4-Quadrant arc tangent of (a), in radians

Degree - Radian Conversions

degrad	Return radians for (a) degrees
raddeg	Return complex degrees for (a) radians
degmin	Return Minutes of a complex degree number
degsec	Return Seconds of a complex degree number
dms	Return complex degree of (a degrees, b minutes, c seconds)

For more information on the specific meanings of these trigonometric functions, see the Trigonometry section of Appendix 5 (page) at the end of the manual.

The CommandLine System

CommandLine Functions

Miscellaneous Floating Point Functions

All of these operations work on, and return 64-bit floating point numbers.

abs Absolute Value value of (a).
abs(-4) == 4

ceil Greater Whole Number.
ceil(1.45) == 2

exp Natural exponent: n to the power of a.
exp(1) == n

fact Factorial of (a), which may be negative.
fact(4) == 24

floor Lesser whole number.
floor(1.9) == 1

log Log in base 10 of (a).
log(10) == 1

logn Natural log of (a).
logn(exp(1)) == n

max Return greater of a and b.
max(1, 5) == 5

min Return lesser of a and b.
min(1, 5) == 1

pow Exponentiate a to the power of b.
pow(3, 4) == 81

rand Generate a random number from 1 to (a), inclusive.
rand(5) == some number in {1, 2, 3, 4, 5}

round Round a to decimal place b (rule of 1/2).
round(1.45, 1) == 1.5
round(1.45, 0) == 1
round(1.50, 0) == 2
round(15.5, -1) == 20

sqrt Square root of(a). Returns 0 if a <= 0.
sqrt(9) == 3

srand Seed the Random Number generator with a starting value. This helps to make the random Number generator less predictable. Starting Values can be between 0 and 65,535.

trunc Truncate a to decimal place b.

```
trunc( 1.45, 1 ) == 1.4
trunc( 1.45, 0 ) == 1
trunc( 1.9,  0 ) == 1
trunc( 19, -1 ) = 10
```


The CommandLine System

CommandLine Functions

Relational & Logical (True/False) Operations

Relational & logical operations return either true (1), or false (0), as than (<) signs have alternate uses in specifying variables, they should always be followed by a space. Comparisons will compare any value types, but both values should be of the same type for meaningful results. Logical operators will interpret any non-zero value as a 'true'.

== 'is equal to'

!= 'is not equal to'

> 'is greater than'

<= 'is less than or equal to'

< 'is less than'

>= 'is greater than or equal to'

And 'both must be true'

Contains string b is contained within string a

IfIs if a is true (non-zero), return b, else return c
ifis(1, 2, 3) returns 2
ifis(0, 2, 3) returns 3

Nand 'at least on must be false'

Nor 'neither must be true'

Not inverse

Nxor 'both must be true or false'

Or 'either must be true'

Range a <= b <= c: range(1, 2, 3) is true
range(1, 4, 3) is false
range(1, 1, 3) is true

StartsWith string a begins with string b

Xor 'exactly one must be true'

Xrange a < b < c: xrange(1, 2, 3) is true
xrange(1, 4, 3) is false
xrange(1, 1, 3) is false

DayOfMonth(a)	Extracts numeric day from date a
DayOfWeek(a)	Determines day of week (0-6, 0=Sun)
DayOfYear(a)	Determines day of year (1-366)
DaysBet(a, b)	Returns #/days between date a and date b INCLUSIVE DaysBet(today, (today+1)) = 2
Month(a)	Extracts numeric month from date a
Year(a)	Extracts numeric year from date a

The CommandLine System

CommandLine Functions

Variable Functions

Declare	Declare a local variable of name a, type b, size c, with initial value d
KillVirtuals(a)	Kill Virtual Fields with Group Name a
LoadVirtuals(a)	Load Virtual Fields with Group Name a
Remove("a")	Remove the named local variable a (put name in quotes)
Requestor(a)	Call Requestor a

The CommandLine System

CommandLine Functions

Basic Character String Functions

These functions return character strings, and do not modify the original values passed in as parameters.

Capital Capitalize string: first character, and after spaces are upper case, all else is lower case.

Insert Insert string b in string a starting at position c for d characters.
Insert("12345", "abc", 2, 2) == "12ab345"

Left Leftmost b characters of string a.
Left("abcde", 2) == "ab"

Lower Make 1st character lowercase.

Ltrim Trim leading (left hand) spaces.
LTrim(" abc   ") == "abc   "

MakeLower Make entire string lowercase.

MakeUpper Make entire string uppercase.

Mid Get string from position b, for c length from string a.
Mid("Abcde", 2, 2) == "cd"

Right Rightmost b characters of string a.
Right("Abcde", 2) == "de"

Rtrim Trim trailing (right hand) spaces.
RTrim(" abc   ") == " abc"

StrCat Add string b to end of string a.
Strcat("abc", "def") == "abcdef"

String Create a string of b copies of string a.
String("abs", 3) == "absabsabs"

Stuff Place string b in string a starting at position c for d characters.
Stuff("12345", "abc", 2, 2) == "12ab5"

Trim Trim leading and trailing spaces.
Trim(" abc   ") == "abc"

Upper Make 1st character uppercase.

The CommandLine System

CommandLine Functions

Advanced Character String Functions

These functions return numerics, and do not modify the original values passed in as parameters.

Asc Numeric (ascii) code of first character.
Asc("Abcde") == 65 (Capital A)

Len Length of a string.
Len("Abcde") == 5

SubStr Return start of string b in string a, or -1 if not contained.
Substr("Abcde", "de") == 3
Substr("Abcde", "ce") == -1

These functions return 1 as a floating point for true, and 0 for false.

IsAlnum True if character is alpha or numeric.

IsAlpha True if first character is alpha (letter).

IsLower True if first character is lower case.

IsDigit True if first character is a digit.

IsNumeric True if first character is a digit, + or -.

IsPunct True if character is in the set of {."'!;:`}.

IsSpace True if character is a space (asc(a)==32).

IsUpper True if first character is upper case.

The CommandLine System

CommandLine Actions

Action Commands (verbs)

These functions or commands perform actions within the user interface, and generally have no significant return value.

AddEntry(a) Display a blank record using Display "a" as a data entry screen, allowing the "Save" command to add the record to the Display's database. If the record is added, AddEntry will return the record number of the new information. Otherwise, -1 is returned.

BatchReport(a, b) Call batch report a, mandatory reroute all output to filename b.

Beep() Cause a system Beep (sound of pitch at duration: see userinfo).

Browser(a) Call browser of name a. Initial Index will be determined by either the Browser's Search, or the first index of the database.

BrowseRequestor(a, b) Call requestor a, then browser b.

CallPack(a) Pack database name a.

DoSearch(a) Perform the search named in a, performing whatever positive or negative actions are indicated in type 2 and 3 Search Field Records, one time.

EditEntry(a, b) Display record (b), using Display "a" as a data entry screen, allowing the "Save" command to edit the record to the Display's database. If the record is edited, EditEntry will return the record number of the new information. Otherwise, -1 is returned.

EBrowser(a) Call Browser "a", but begin display in Browser Entry Mode.

EditDisplay(a) Edit Any Display: Browser, Report, Menu, DEScreen, PickList, et cetera, in What-You-See-Is-What-You-Get (WYSIWYG) mode. Minimum: DisplayStruct record "a" must exist before calling this command.

EIBrowser(a, b) Call Browser "a", but begin display in Index "b" order, and begin display in Browser Entry Mode.

ExpXbase(a, b) Send all records from database "a" to Xbase format file "b". All fields, in structure "a" order, will be sent.

FileAReport() Call standard Report-to-(system default) File Procedure.

Help(a) Call help for topic a.

IBrowser(a, b) Call Browser "a", with Initial Index named "b".

ImpXBase(a, b) Import XBase information from file "b" to database "a".

KillSafeToggle() Toggle Global Killsafe variable (not internal). While not internal, this only affects browsers.

LoopSearch(a) Perform the search named in a, restarting the search from the beginning after each successful record find. This is useful for purge/edit functions that modify the list being searched.

Map(a, b) Call the search named in a, and copy all records to database named in b that pass the search condition. Type 2 Search Fields will determine how the fields in the receiving database will be filled in.

MemWindow(a) Change memory window 0=off, 1=on, 2=toggle mode.

Menu(a) Start Menu of name a, escaping the menu returns to calling point.

Mesg(a) Display message on screen line 2.

MultiSearch(a, b) Perform the search named in a, for b levels of times, and return the number of records found during level a.

Rebuild(a, b) Rebuild Index "b" for Database a.

RebuildAll(a) Rebuild all Indexes for Database "a".

Report() Call report-to-(inquired) file system.

ReportCall(a, b) Call report a, send to file b.

RExpXbase(a,b,c,d) Restricted Export: Export records from database file "a" to a temporary record in database "c". If the records satisfy criterion "d", map them into XBase file name "b". All fields, in structure "c" order will be mapped.

RImpXbase(a,b,c,d) Restricted Import: Import records from XBase file "b" to a temporary record in database "c". If the records satisfy criterion "d", map them into database "a".

RunAReport() Call Standard Report-to-Screen system.

The CommandLine System

CommandLine Actions

Procedure-Oriented Commands

Call(a) Load and run procedure of name a.

CommandMode() Enter the Command Line Interface.

Do { Perform action within braces, loop until terminating while becomes false.

Else { If previous condition was false (endif, enddo, endwhile), perform actions in braces.

ElseIf(a) { If previous condion was false and a is true, perform actions in braces.

If(a) { If a is true (nonzero), perform actions within braces.

IfCall(a, b) If a is true (nonzero), call procedure named b.

Quit() Unconditional shutdown/termination to DOS.

Stop() Stop procedure: Program equivalent of <Esc> in sensitive mode.

While(a) { Loop call inside braces as long as a is true.

} Ends if, else, elseif, and do statements.

} While(a) End "Do" loop: loop if a is true.

Pause(a) Display message a, wait for keypress to continue.

YesNo(a, b) Ask for Yes/No answer with prompt a, default answer b, return true for Yes answer.

DiskBufferStats() Display statistics about internal system's disk buffers.

DosShell() Drop to a Dos Command Shell.

DosCall(a) Call dos function a (internal, batch, executable).

DosCallNoKey(a) Call dos function a, but do not ask user to "Press a Key to return to VariForm". This is for batch-mode doscall()'s.

FileCopy(a, b) Copy dos filename a to filename b.
This implementation of the copy command can be used on an already open file.

FileDate(a) Return date of last change to dos filename a.

FileExist(a) Returns true if dos filename a exists.

FileKill(a) Kill dos file named a.

FileMove(a, b) Copy dos filename a to filename b, delete a.

FileRename(a, b) Rename dos filename a to b.

FileSize(a) Return size of filename a in bytes.

FileTime(a) Return time of last change to dos filename a.

FlushFiles() Flush all changed, unsaved indexes to disk. This is only useful for single-user (OnNetwork=0) settings.

FlushSpares() Remove 'kept' processes and search conditions from memory.

The CommandLine System

CommandLine Actions

Internal Commands (Browsers and BrowseLists)

These function/verbs can only be used as functions within Browsers or BrowseLists.

Add() Add record, to Browser, by D-E Screen.

AddRecord() Add blank record to Browser database.

ChooseIndex() Pop a picklist for the user to select the Browser's active index.
* This is invalid for BrowseLists whose search contains an index.

Copy() Copy Browser's current record, enforcing uniqueness.
If there are automatic relations to child databases, those records will also be copied.

Delete() Delete current record on Browser.

EditRecord() Edit Current record on Browser, by D-E Screen.

EntryMode() Switch to Browser Entry Mode. Browser will now act like a data entry screen, using DEScreen Colors, and DEScreen Function Key Set.

Info() Generate and display record count, offer to get detail on erased and damaged records.

KillRecord() Kill Current record.

NextIndex() Switch Browser to display by next index.

ReIndex() Rebuild the Browser's Current Index.

ReIndexAll() Rebuild All indexes in the Browser's Database.

Select() BrowseList Only: exit the BrowseList, performing whatever functions are specified in the BrowseList's SearchStruct's type 10 records. This is how a BrowseList can work like a PickList.

Pack() Pack the Browser's Database.

The CommandLine System

CommandLine Actions

Internal Commands (Data Entry /Browser Entry Mode)

BrowseList(a) Call named browselist "a" from current data entry screen. This operates like a standard browser, but the "Select()" function acts like a picklist selection.

GenPickList(a,b,c) Call Complex Picklist "a", display with Title "b", and set #target2#="c". This allows one, generic picklist to do the display work of hundreds of subgroup-specific picklists. This operates as a normal picklist, but the title is replaced by the string in 'B', and the Search for the picklist will often also use #Target2# in its search criteria.

Help(a) Call help system for topic "a".

HelpInternal() Call help system, with a complex search for the help topic value. If the Current DisplayField contains a HelpTopic, display it. If the DisplayField's HelpTopic is blank, check the Current DisplayStruct's HelpTopic. If the DisplayStruct's HelpTopic is also blank, then display help based on the current interface mode, which is either "MenuSystem", "Browsers", "Data Entry", "BrowseLists", "PickLists", or "HelpOnHelp".

MapSearch(a) Perform map/search using Search "a" for the ChildRecord and criteria, and the current Browser, SearchChild record, or Data Entry record as the Parent record.

PickList(a) Call PickList "a" from current Data Entry Screen and DisplayField, returning #Target# to chosen field if a selection is made. #Target# initially contains the current DisplayField's value upon entering the PickList, and is modified by Type 10 records in the PickList's search, if the user makes a selection.

Redraw() Redisplay current Display, performing all lookups, and redrawing all fields.

The CommandLine System

CommandLine Actions

Internal Commands (Searches and Maps)

These function/verbs can only be used as functions within Browsers or BrowseLists.

AddRec() Add record to SearchChild Database.

AddRecord() Add record to SearchParent Database.

EditRec() Search: Edit / resave SearchChild current record.

FieldIncrement(a) Increment field "a" in SearchChild record.

KillRec() Delete current SearchChild Record.
* This should only be used in DoSearch or LoopSearch, since a MultiSearch or Map will try to find the next record from current after it has been deleted.

MapAllFields() Copy all fields from the child record to fields of the same name in the parent record. Generally only used in Map() calls.

The CommandLine System

CommandLine Actions

Financial (Business Math) Functions

Compound(a,b,c) Compound interest formula of a at rate b for c periods.
Example: $10,000 in the bank, earns 7.0% for 15 years.

Compound(10000, .07, 15) Returns $27,590.31

FutVal(a,b,c) Future value of a payments, earning b rate for c periods.
Example: $100 deposited per month, earning 7% for 36 months.

FutVal(100, (.07 / 12), 36) Returns $4,016.30

PV(a,b,c) Present value needed to reach future value a at rate b for c periods.
Example: Goal is $100,000, interest rate is 7%, in 12 years.

PV(100000, .07, 12) Returns $44,401.19

Annuity(a,b,c) Present value of a compounding annuity

The CommandLine System

CommandLine Internal Variables

BrowserChase If non-zero, all Browsers will "chase" the user's editing and adding work. The record edited or added will be brought to the top of the browser, and made current. This allows the user to see records after they are entered or changed, without having to search for them.

DateFormat Number indicating default format of date fields when stored or expressed as character strings.

DisplaySeverity System Error Severity necessary before an error message is displayed. Should be <= KeyPressSeverity.

InsertMode 0 = OverWrite, 1 = Insert, 2 = Store state from previous Screen.

KeyPressSeverity System Error Severity necessary before an error message pauses, waiting for a keypress. Should be >= DisplaySeverity.

#KillSafe# If non-zero, browser deletes will ask the user for confirmation (are you sure?) before deleting records.

MemWindow Status of Memory Window (0= closed, 1=open).

Now System Time, updated by data entry system.

OnNetwork (MultiUser) If nonzero, all index writes are automatically flushed upon changing (defeating the internal buffer), and all index reads are preceded by a check of file date and time (defeating the internal cache). This makes multiple-user concurrent editing on a database safe.

The specific value of OnNetwork determines which Database Types, according to the Characteristics field, are affected. This allows the user the freedom to localize some files, for faster, single-user access, while sharing the rest. This is accomplished by treating the OnNetwork variable as a Bit Field.

Bit	Value	Database Type Affected
0	1	Basic Databases: Non-System, Non-Binary
1	2	System Databases
2	4	Binary Databases
3	8	Alternate Databases

Example: To network all Binary and Alternate databases, while leaving Basic and System Databases in single-user mode, set OnNetwork to 12.

Page Page Number field used by reports.

PagedVars When nonzero, Command Line Interface will pause after displaying a variable-screen-full of variables.

The CommandLine System

Pi The Value of pi, as is calculated by the system for use within the trigonometry system. This is reference only: changing its value will not affect system operations.

PrinterType The number of the printerID most recently used to generate a report. If zero, VariForm will use the UserInfo default printer. This allows report procedures to specify what format code set to use for a given report, by modifying PrinterType.

RecNo This variable contains a long int value tossed between browsers and data entry screens that signifies the current record number.

RecordSeverity System Error Severity necessary before an error message is recorded in the error/action log.

ScreenSaver Number of Seconds of inactivity before the VariForm Screen Saver will begin displaying. A value of zero in this field will disable the Screen Saver altogether.

SearchExact If non-zero, Browser Searches will 'Beep' if the user types in a nonexistant field string. If zero (default), Browser Searches will search to the first record with a field greater than or equal to the typed text.

SecurityActive If non-zero, the Security Fields in all display systems will be 'armed', preventing access to menus, displays, fields, and functions by users whose UserInfo->Security Field is too low.

SecurityMode With the enhanced security module, Security Fields are treated as bitfields instead of numbers. The advantage to this method is the capability of having more arbitrary, less sequential security assignments for access permissions. This field is meaningless unless the Enhanced Security Module is in use, and SecurityActive is non-zero.

Sensitivity If non-zero, Searches and procedures will abort to calling menu/browser/descreen if the user presses <Escape>.

Today System Date, updated by data entry system.

The CommandLine System

CommandLine Internal Variables

CommandLine Scratch Variables

CommandLine Temporary String Variable for VariForm when running the CommandLine Processor.

#DateAnswer# Scratch pad for date calculations.

#FloatAnswer# Scratch pad for floating point calculations.

#FloatAnswer2# Scratch pad for floating point calculations.

#FloatAnswer3# Scratch pad for floating point calculations.

#IntAnswer1# Scratch pad for integer calculations, yes/no requestors.

#IntAnswer2# Scratch pad for integer calculations, yes/no requestors.

#IntAnswer3# Scratch pad for integer calculations, yes/no requestors.

MessageSpace Free, character string scratch-pad area.

#PrintFile# Text Field set by report system to name of last file a report was extracted to.

#Target# Function Keys on DEScreens get the current field's contents here. PickLists, GenPickLists, BrowseLists will stuff the return value from #Target# into the calling variable.

#Target2# GenericFunctions (GenPickList) get their second argument placed here. This is useful for complex searches, such as the numeric charts database. This is also the second field to be filled on multi-field lookup returns.

#Target3# This is the third field to be filled on multi-field lookup returns.

#Target4# This is the fourth field to be filled on multi-field lookup returns.

#Target5# This is the fifth field to be filled on multi-field lookup returns.

Example Data System

Overview

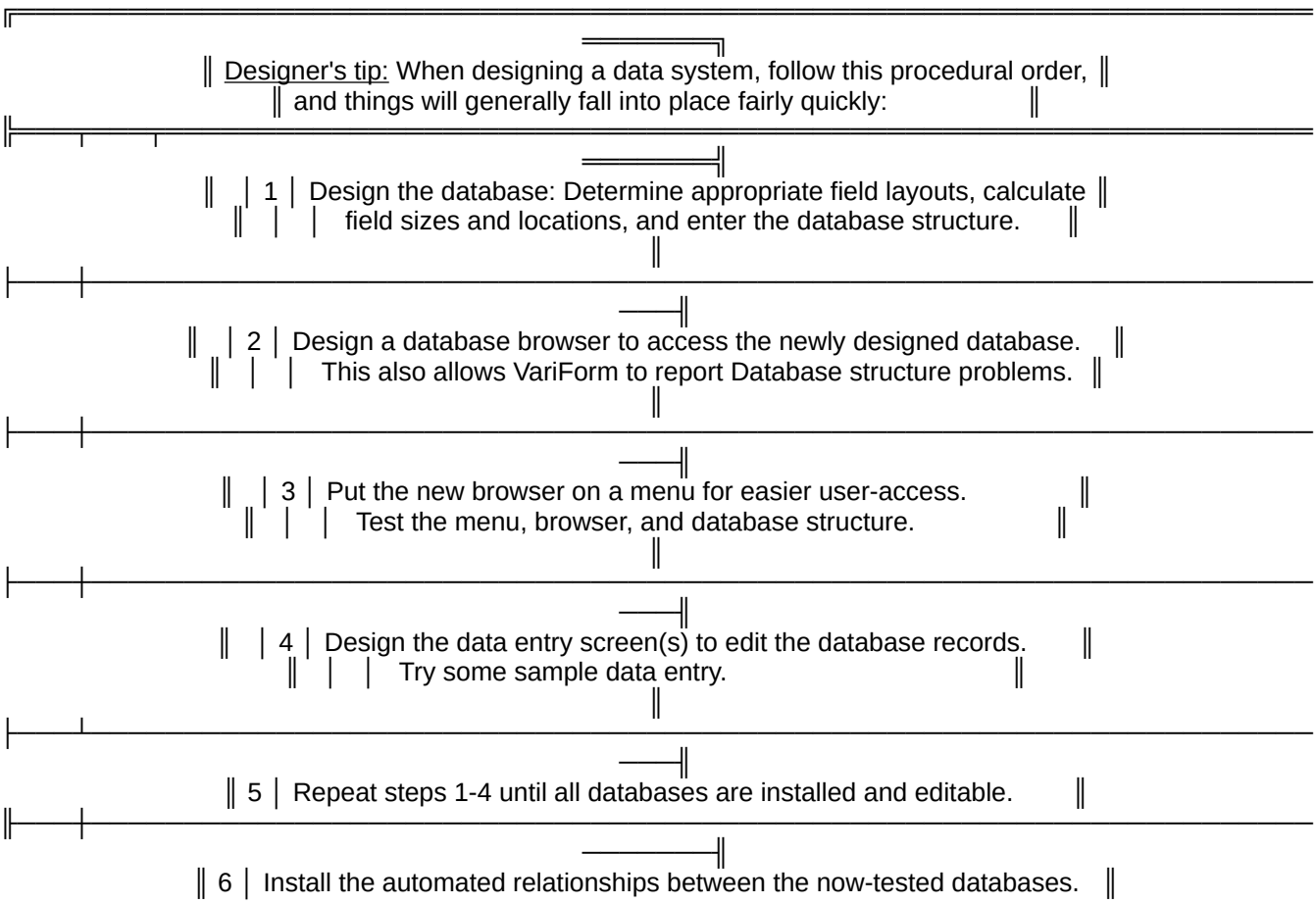
This chapter provides a step-by-step example of how to develop database systems with VariForm. The example is fairly detailed, and should provide a good 'walk through' of VariForm's standard system functions.

The need for a 'simple' client database

The goal is to develop a client database system that can store client addresses and summary information. This database will have access to a sales history database, which in turn has access to an items database. Both clients and sales may need to be reported on individually, as well as by a sales 'region'. The system must be able to accept data entry, generate total purchases per client. Reports will include mailing labels, and some regional sales information.

While this may sound sophisticated, it is actually a fairly simplistic view of a customer, a sale, and an item. None of the procedures involved in this project require any 'in-depth' analysis. This problem can be addressed simply, and in the order presented above.

At all stages, a little preliminary work can prevent a lot of mistakes later on: design, draw, layout. Whatever it is called: put it on paper initially, to check the designs for consistancy and coherency. This can save many editing and reworking sessions later.



	This includes installing nested fields and sub-browsers.
7	Develop and test procedures for the data entry screen, and the browser.
8	After all calculations have been tested, design the report.
9	Insert applicable help topics and cross references.

Example System

Step 1: Design the Database

In this system, there will actually be 4 databases for design:

- Client Info, defining a name and address with a unique 'client id'.
- Items, defining an item by number and description.
- Sales Region, defining a region by an identifier and description.
- Sales History, containing individual items sold to a customer on a given date.

Given this general description, the next step will be defining fields for each of the databases.

Client Info

The Client Info database must contain information about the client, a mailing address, telephone number, and so forth. It must also contain a unique identifier that other databases can use to refer to a particular client record. Below is an outline of the database:

Field Name	Type	Size	Reasoning
Client ID	Alpha	6	Five characters > 60,466,176 clients
Title	Alpha	6	Mrs. is longest: keep fields even length
First Name	Alpha	20	Long enough for compound first names
Last Name	Alpha	20	Long enough for European last names
Address1	Alpha	40	Most labels are < 4" wide, 40 12-cpi's
Address2	Alpha	40	Two address lines: Box, Street & Suite
City	Alpha	26	Longest City name in US is 25 char.'s
State	Alpha	4	For Canada, make this 6 for provinces
Zip	Alpha	12	Zip (5) plus 4 plus hyphen Alpha for hyphen, and for Canadian Zips
SalesRegion	Alpha	4	3-character sales region id's
FirstPurch	Date	4	Date of First Purchase

TotalPurch	Float	8	Total of all purchases ever made

Designer's Note: Field Sizing: Most field types, such as dates, numbers, times, have a constant size, which is defined by the machine in use. The primary exception to this is the character field. In general, a character field should be at least one character more than the maximum number of typed characters needed. For best results, try to round the size up to the next larger even number; this way, information can be moved to systems that are 'word-locked', or require all memory accesses to occur at even-numbered locations.

See the chart on page for a more detailed description of field sizing values.

With this database design in hand, it is time to start using VariForm.

Example System

Step 1: Design the Database

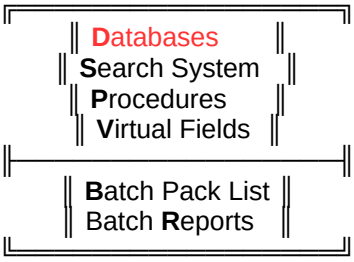
Start VariForm: Select the VariForm entry or Icon, or from the standard DOS prompt, change to the VariForm directory, type either **CORE** or **WCORE** and press <Enter>. The system will initialize, and a small menu will appear in the bottom center of the computer screen.

Note: There may be one or several applications, which is why this is shown with square brackets ([]).

Select VariForm: The topmost option of the package, when it is first installed. Select the menu option by either typing **<V>**, or moving the menu cursor bar to the line that says "VariForm", and pressing **<Enter>**. This will cause a horizontal menu to appear at the top of the screen.

Note: Most horizontal menus do not have borders. They are shown here to separate them from the text.

Select the System Menu: The leftmost option of the VariForm Menu is the Display Menu. Select the Structures menu option by either typing **<S>**, or moving the menu cursor bar to the column that says "Structures", and pressing **<Enter>**. This will cause a vertical menu to appear below the selection. For a more detailed description about the databases, see the 'Primary Databases' (p.) and 'Secondary Databases' (p.) sections of this manual.

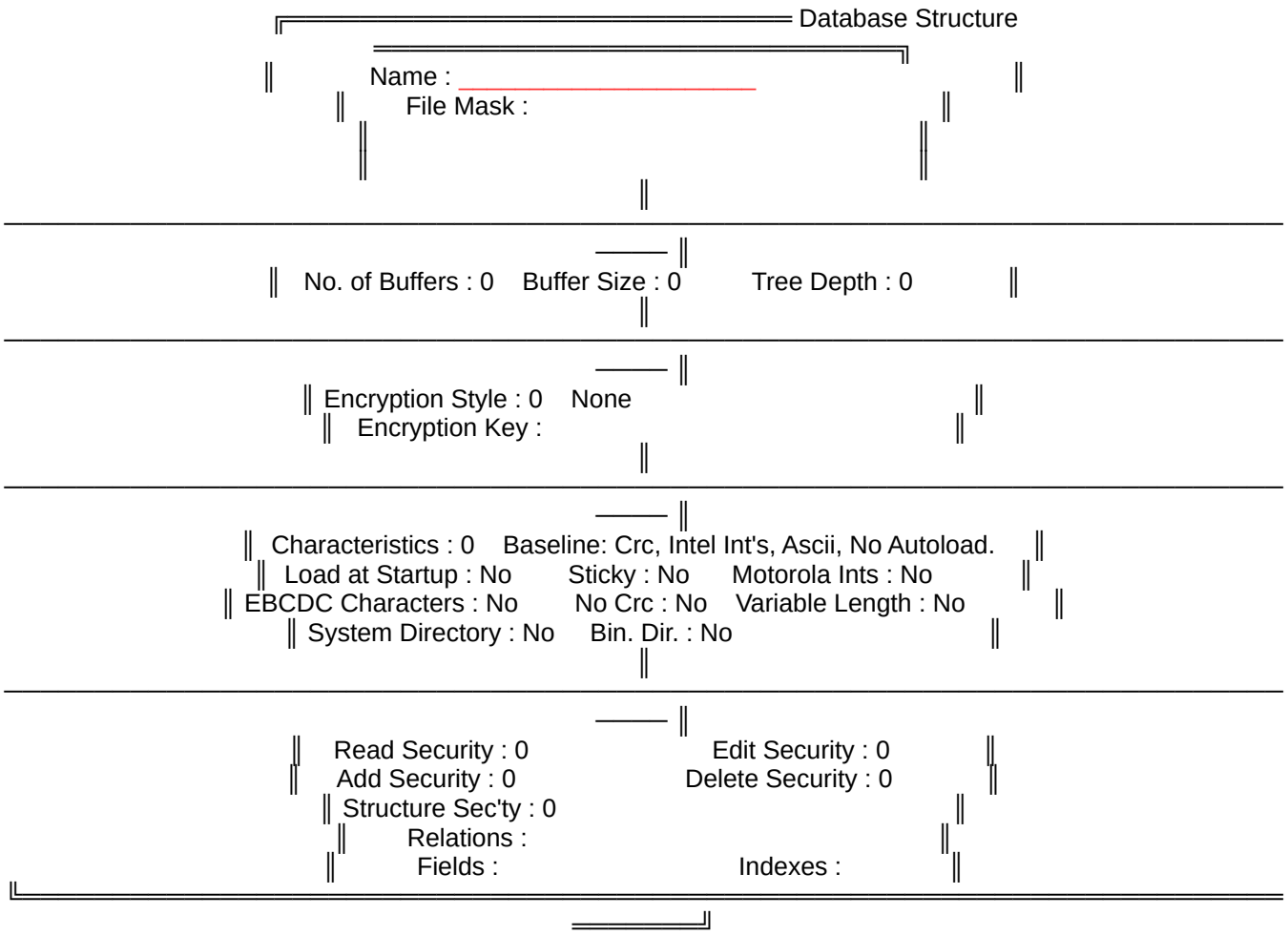


Select the Database Browser: The topmost option of the VariForm Structures SubMenu is "Databases". Select it either by pressing **<D>** or moving the menu cursor bar to the appropriate line and pressing **<Enter>**. This will bring up the Database of Databases Browser, which shows each database as an individual record.

Example System

Step 1: Design the Database

To Create the New Database: Press the **<F3>** (Add a record key), to bring up a blank database record to fill in.



Enter a unique **name** to identify the database. For this example, the name "ClientInfo" will suffice. Type ClientInfo at the Name prompt, and press **<Enter>** to move to the next field.

Enter a unique **file mask** to identify the database, such as CliInf.Dat. The file mask tells VariForm where to store the database information. It is called a 'mask', because any automatic indexes attached to this database will use the same filename, along with a 3-digit numeric extension. The mask may contain a drive and directory designation, to specify an absolute file location, but it is more convenient to use the system and binary flags to determine the database's directory location. Type CliInf.Dat at the File Mask prompt, and press **<Enter>** to move to the next field.

The next three fields control a technical storage aspect of the **automatic indexes**. For more technical information about Buffers and Trees, see the 'index calculations' section of Appendix 1, on page . As a general rule of thumb, use the following default values to begin with, and finesse the index sizes later.

No. of Buffers: 2
Buffer Size: 1024
Tree Depth: 11

Example System

Step 1: Design the Database

The next two fields control data **encryption** within the database. This prevents 'hackers', who may steal your database files, from being able to read the contents. Leave these two fields blank (**<Enter>** past them); Encryption does not become a significant until the system is ready for actual data input.

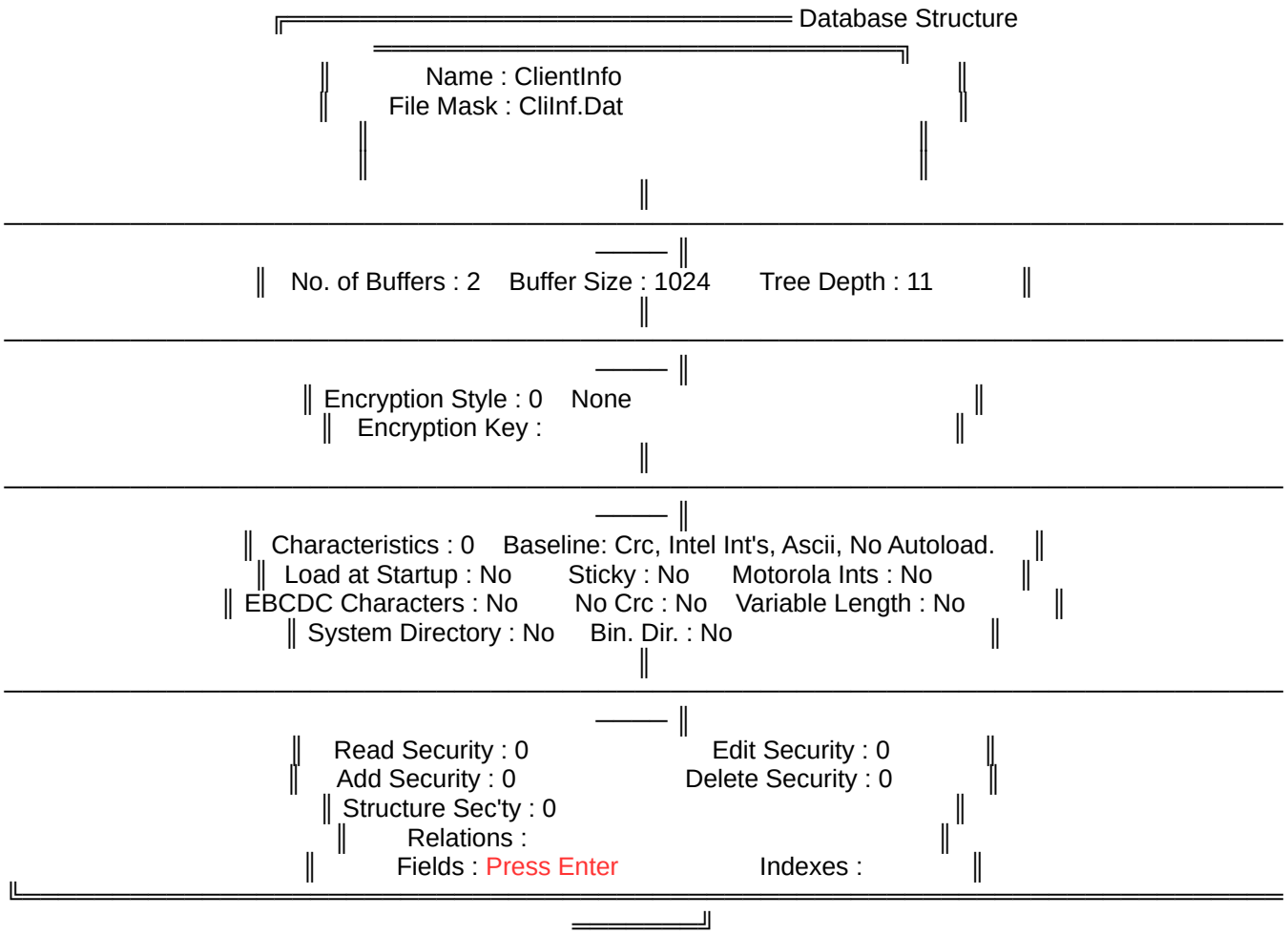
Example System

Step 1: Design the Database

The next group of fields all refer to the first field: all of the Yes/No options refer to bits within the **Characteristics** field. This allows experienced users to enter one number and move on, while giving new users a series of yes-no questions, instead of a large PickList. When designing a database, leave this section blank.

The next five fields refer to **security** limitations on the database, and should only be used after the database has been designed and tested. As with encryption, leave it blank until it is time for live data.

When the cursor reaches the **Relations** prompt, the words "Press Enter" appear. This indicates that pressing **<Enter>** will access some other information, which is the Search System in this case. The proper term for this type of field is 'Nested'. To leave this field, use **<Tab>**, or **<Down Arrow>**. Since this belongs to a later step (see the list of steps at the beginning of this section on p), skip down to the next field. The Database Record Window should now look like this:

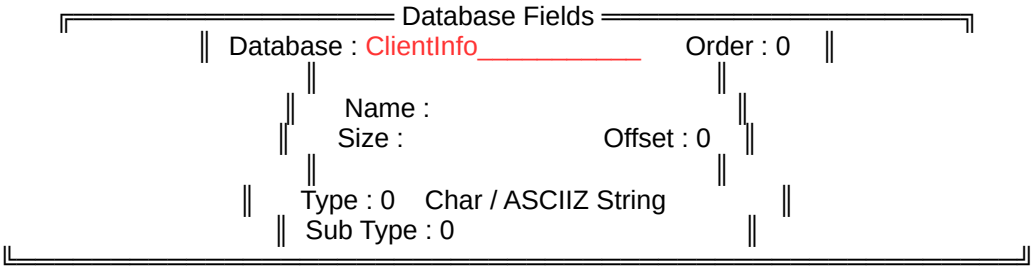


Press <Enter> to access the fields sub-database.

Example System

Step 1: Design the Database

The **Fields** prompt accesses a limited browser on the Database of Fields, to allow the user to enter the Specific Fields that are used within the database. Press <Enter> to access the Database of Fields Browser. Press <F3> (Add record) to bring up a blank record for editing.



The **Database** Field, which connects the Field Record to the appropriate database, will already be filled in. This is part of the natural function of a nested field-based relationship. **<Enter>** past this field.

Order refers to a number that will keep the fields in a specific sequence: the first field will be order number 0, the second field, order 1, and so on. **<Enter>** past this field on the first field record.

The next field requires a field **Name** that will be unique within the database. For this example, the first field's name is "ClientID".

The **Size** field needs the field's size in bytes. For character fields, this is the number of typable characters plus at least one. For non-character fields, see the chart of field sizes on page .

The **Offset** field describes the field's location within the database record. For 'simple' records, this is the sum total of all field sizes that precede it; for 'complex' records, which contain union fields, this is the total of all fixed, primary fields that precede the current entry. For the first field entry, the offset is always zero.

Type refers to a numeric code that tells VariForm what kind of information will be stored in this field: numeric, date, et cetera. If uncertain of the correct value, press the <F2> (picklist) key, and a list of field types will appear: move the picklist's selector bar with arrow keys, press **<Enter>** on the correct field type, and VariForm will remove the picklist, and insert the correct field value. For the first field record, the type is zero.

SubType is a multipurpose field that is used for "special effects" on fields, such as fixed-length decimal sizes, and unique field indicators. For the first field record only, the sub type is **2** (unique). Normally, this field is left blank.

Note: For more information on field Types and SubTypes, see the chart on page .

Note: A unique field must have an 'exclusive index'. This index will be of the unique field only, and is used by VariForm to assure that the newest record entry is different from all other entries in the database.

After pressing **<Enter>** on SubType, the cursor will move back to the Database prompt. Press **<Ctrl-Enter>** to save the record. The field record will be saved, and disappear, returning control to the Database Fields browser.

The rest of the fields can be entered in the same way, with a little care taken to maintain an increasing Order number, and a correct Offset value for each field. When completed, the Database Fields Browser should look like the table on the following page.

Example System

Step 1: Design the Database

Fields within a database

Database	Order	Name	Size	Off.	Type	Sub Type
ClientInfo	0	Client ID	6	0	0 Char / ASCIIZ	2
ClientInfo	1	Title	6	6	0 Char / ASCIIZ	0
ClientInfo	2	First Name	20	12	0 Char / ASCIIZ	0
ClientInfo	3	Last Name	20	32	0 Char / ASCIIZ	0
ClientInfo	4	Address1	40	52	0 Char / ASCIIZ	0
ClientInfo	5	Address2	40	92	0 Char / ASCIIZ	0
ClientInfo	6	City	26	132	0 Char / ASCIIZ	0
ClientInfo	7	State	4	158	0 Char / ASCIIZ	0
ClientInfo	8	Zip	12	162	0 Char / ASCIIZ	0
ClientInfo	9	SalesRegion	4	174	0 Char / ASCIIZ	0
ClientInfo	10	FirstPurch	4	178	7 Date	0
ClientInfo	11	TotalPurch	8	182	4 Floating Point	0

Things to note:

The order numbers may skip values, such as 2, 4, 6, 8, instead of 0, 1, 2, 3. There are no potential problems unless there are duplicate order numbers.

Offsets always start at zero, and add the previous field's size to the previous field's offset.

Field Names must always be unique within a database. For example, there are not two address fields, but one "Address1", and one "Address2", which are used as address fields.

Once the entries for the field structure are complete and correct, press **<Esc>** on the Field Browser to return to the parent database record. Press **<Tab>** to get to the last field on the database record. This nested field controls how the database records are organized: this is the definition of **indexes**. Press **<Enter>** on this field.

Indexes will be loaded and maintained automatically whenever the database is accessed, and provide superior searching capabilities to the VariForm searching and mapping systems. The browser for the index system will appear. Press **<F3>** (add record) to bring up a blank data entry screen for an index.

Indexes

Database : ClientInfo_____ Order : 0 Index :

Characteristics : 0

Field 0 Name :	Field 0 Size : 0
Field 1 Name :	Field 1 Size : 0
Field 2 Name :	Field 2 Size : 0
Field 3 Name :	Field 3 Size : 0
Field 4 Name :	Field 4 Size : 0
Field 5 Name :	Field 5 Size : 0
Field 6 Name :	Field 6 Size : 0
Field 7 Name :	Field 7 Size : 0
Field 8 Name :	Field 8 Size : 0

Field 9 Name :	Field 9 Size : 0

The **Database** Field, which connects the Index Record to the appropriate database, will already be filled in. This is part of the natural function of a nested field-based relationship. **<Enter>** past this field.

Example System

Step 1: Design the Database

Order refers to a number that will keep the fields in a specific sequence: the first field will be order number 0, the second field, order 1, and so on. **<Enter>** past this field on the first field record.

Index refers to a unique name for this index. An index may have the same name as a database field, but must not have the same name as another index. For this example, the first index should be named **ClientID**.

The **Characteristics** field is a number that tells VariForm of any 'special treatment' that needs to be performed on this index, such as reverse-ordering, case-sensitivity, and so forth. **<Enter>** past this field.

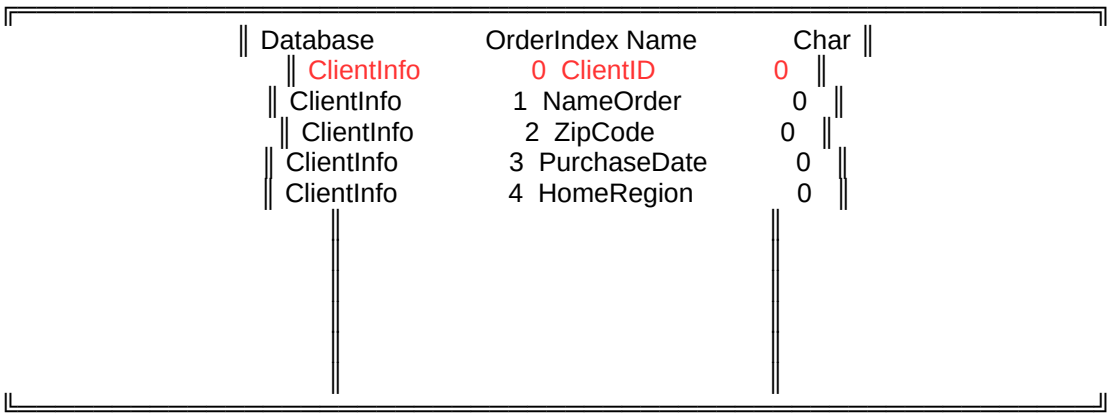
The balance of the fields are paired of in rows, and list the **Field Name** and **Number of Bytes** to reserve for each field in this index. For character fields, the size values allow the user to control how much of the field is used in the index, and can help reduce an indexe's overall size. Smaller indexes sort and search faster in a fixed amount of memory.

For the first index, the only field needed is the **ClientID** field, and all **six** bytes will be stored in the index. This is the only way to accurately maintain a unique field: an index containing only the unique field must be present, so that VariForm can search for and prevent matching field values.

Add the following indexes to this database, each as a separate record:

- Index "NameOrder", by LastName (26) and FirstName (14)
- Index "ZipCode", by Zip (12)
- Index "PurchaseDate", by FirstPurch (4)
- Index "HomeRegion", by SalesRegion (4)

Once entered correctly, the Index Browser should appear as follows:



With the index layouts in place, press **<Esc>** to leave the index browser, and the Database of Databases Record Entry becomes current. Press **<Ctrl-Enter>**, to make sure the Database Record is saved, and the Database of Databases browser becomes current. Press **<Esc>** to leave the browser, and the VariForm System submenu becomes current.

Example System

Step 1: Design the Database

The database design is complete. The last portion of step 2 will be designing a browser to access this database.

Example System

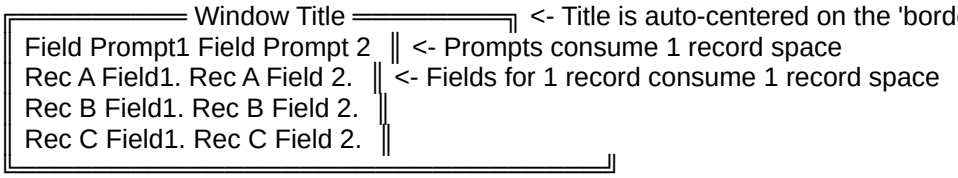
Step 2: Design the Browser

Browser Strategy

Accessing the ClientInfo database will require a browser. Browsers are comprised of 2 groups of display elements. The primary elements of a browser are Title, Item Height, and Window Size. The secondary elements of a browser are fields and field prompts. The primary elements are stored in a single Display Structure, and the secondary elements are stored in one Display Field record per field displayed on the browser.

The ClientInfo browser should show at least all of the primary indexed fields; the first field of each index should be available on the browser display, so the end-user can see the current search field for any index.

There is a strategy that makes browser design simple: make a small layout of the browser before beginning data entry.



The ClientInfo browser should have the ClientID, LastName, Zip, FirstPurch, and SalesRegion fields, which are the first fields of each index, in index order.

Suggested Browser Field Widths

- Character fields' display widths can and generally should be the full field width.
- Date fields should generally be 10 characters wide (2 digit month + 2 digit day + 4 digit year + 2 separators = 10 characters). They should always be padded to full length, for neatness.

- Time fields should generally be 8 characters wide (2 digit hour + 2 digit minute + 2 digit second + 2 separators = 8 characters), unless milliseconds are displayed. For milliseconds, time fields should be 12 characters long (8 + 1 decimal point + 3 decimals).

- Integers can contain values exceeding 32,000, but are usually sequence values or small-scale counting, such as line numbering fields, bit fields, and order/suborder elements. Often, a width of 2 or 3 characters is sufficient.

- Long Integers can contain values exceeding 4,000,000,000, but are generally used for lower-scale counting. A width of 5 to 8 characters is usually more than adequate.

- Floating Point Numbers should be sized according to the largest possible number expected (times ten), and the lowest possible fraction, plus 1 for the decimal point. Add one more character for the sign if negative numbers are allowed. Example, numbers that won't typically exceed 10,000 and don't generally have significance beyond the

thousandths place (0.001), would be treated as 6 (100,000) + 1 (decimal) + 3 (fraction). Floating point numbers appear best when they are right-justified.

Example System

Step 2: Design the Browser

To calculate the browser's width, add the display lengths together, plus spaces between the fields, which is number of fields plus 1:

6 + 20 + 12 + 10 + 4 + 6 = 58 columns in width

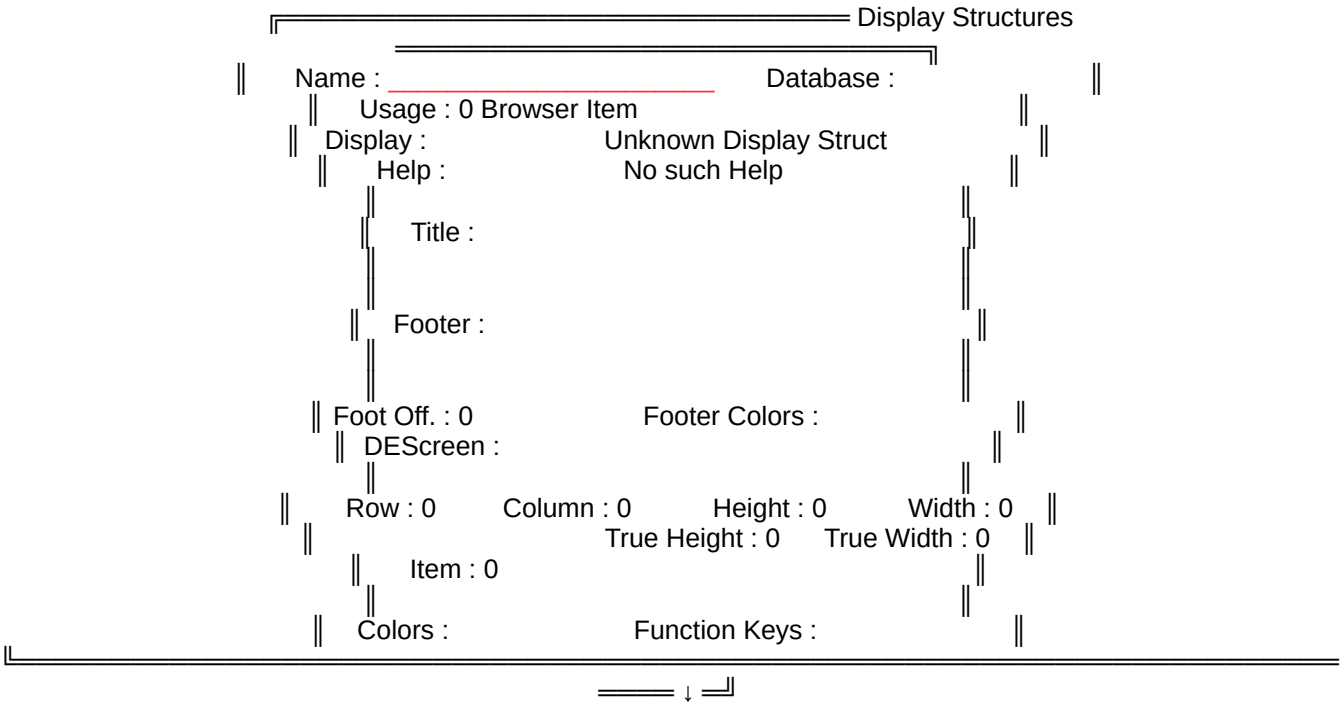
The browser should show about 14 'records', so it will be 15 record spaces high. Since each record needs 58 columns, and PC-screens are 80 columns wide, each record can fit on 1 row.

15 record spaces * 1 row / record space = 15 rows in height.

To Create a Browser: From the VariForm Displays SubMenu, select "Display Systems", either by pressing **<D>**, or moving the menu cursor bar to the appropriate line and pressing **<Enter>**. This will bring up the main Display Structure browser.

A Note about Display System records: Display System records perform most of the interface design and control on all VariForm Systems. The same database services browsers, data entry screens, menus, picklists, and report formats. Only the pertinent fields for each section will be discussed. For a comprehensive explanation of the Display Structures, see the "Display Database" portion of the Primary Database description section of this manual (page).

Press **<F3>** on the Display Structures browser to bring up a blank record for editing. The window will appear as follows:



Note: The downward-pointing arrow on the lower-right hand corner of the window border indicates that there is more on this data entry screen below the window's current viewing area. The other portion of the screen will be shown later in this section.

Example System

Step 2: Design the Browser

The **Name** of the structure must be unique, to keep it separate from other data structures. It is also a good idea to use naming conventions that help the designer remember at a glance what each structure does. In this example, the name "ClientInfoBrow" makes sense, since it is the primary Browser for the ClientInfo database. Type **ClientInfoBrow** and press **<Enter>**.

Example System

Step 2: Design the Browser

The **Database** field refers to the database the browser will be editing. Type **ClientInfo** and (PickList) key to select from an list of all database of databases entries.

Usage refers to how the Display Structure will be used: as a database, browser, data entry screen, et cetera. Since this is already at type 0, **<Enter>** past this field. When entering non-browsers, use the **<F2>** (PickList) key to select from a list of all Display Structure Types.

The **Display** field is used to allow different display structures, such as a limited and unlimited browser, to share the same display fields. The **<F2>** key will generate a PickList of all available display structure names, except the one being currently entered. The **<F3>** key will copy the Display Structure name into this field. Press **<F3><Enter>**.

If a particular **Help** topic exists that would apply to this Browser when in Entry Mode, Enter the topic name here. **<F2>** can provide a PickList of help topics. For this example, **<Enter>** past this field.

The **Title** is the text that will be displayed, automatically centered, on the top of the display window's border, if one exists. Type "**Client Information**" here, and press **<Enter>**.

The **Footer** contains the cue lines to display when the browser is in "Entry Mode". **<Enter>** past this field.

Footer Offset is how many lines (0-4) above the base line to display the screen's cue lines. **<Enter>** past this field.

The **Footer Colors** field contains the name of the color structure record to use when drawing the cue lines, even if there are no cue-lines to draw. **<F2>** provides a PickList of color structures to select from. The default is "HelpPlain", which refers to the fact that there is no separation line between the Screen- and Field-specific cue lines.

DEScreen refers to the name of the Data Entry Structure Screen to load when <Enter> is pressed on the browser. Type "**ClientInfoEntry**" and press **<Enter>**.

The next four fields require a little mathematics for aesthetics: they refer to the location and size of the browser on the computer screen, excluding borders. Although computer screens are typically 80 columns by 25 rows, centering on 80 columns by 21 rows will usually look better. This is due to the four available cue lines at the bottom of the screen, that should not be covered up by any window. The width, as calculated before, is 64 characters. The height is the number of records to display at one time, plus one record for the Title Area. Multiply the record count by the number of lines per record, which is one in this case. For this example, the browser will display 14 records at a time, and will therefore be 15 rows high.

To Calculate the **Row**, take (21 - Height) / 2: Enter **3**
To Calculate the **Column**, take (80 - Width) / 2: Enter **11**
Height will be **15** rows
Width will be **58** columns

Example System

Step 2: Design the Browser

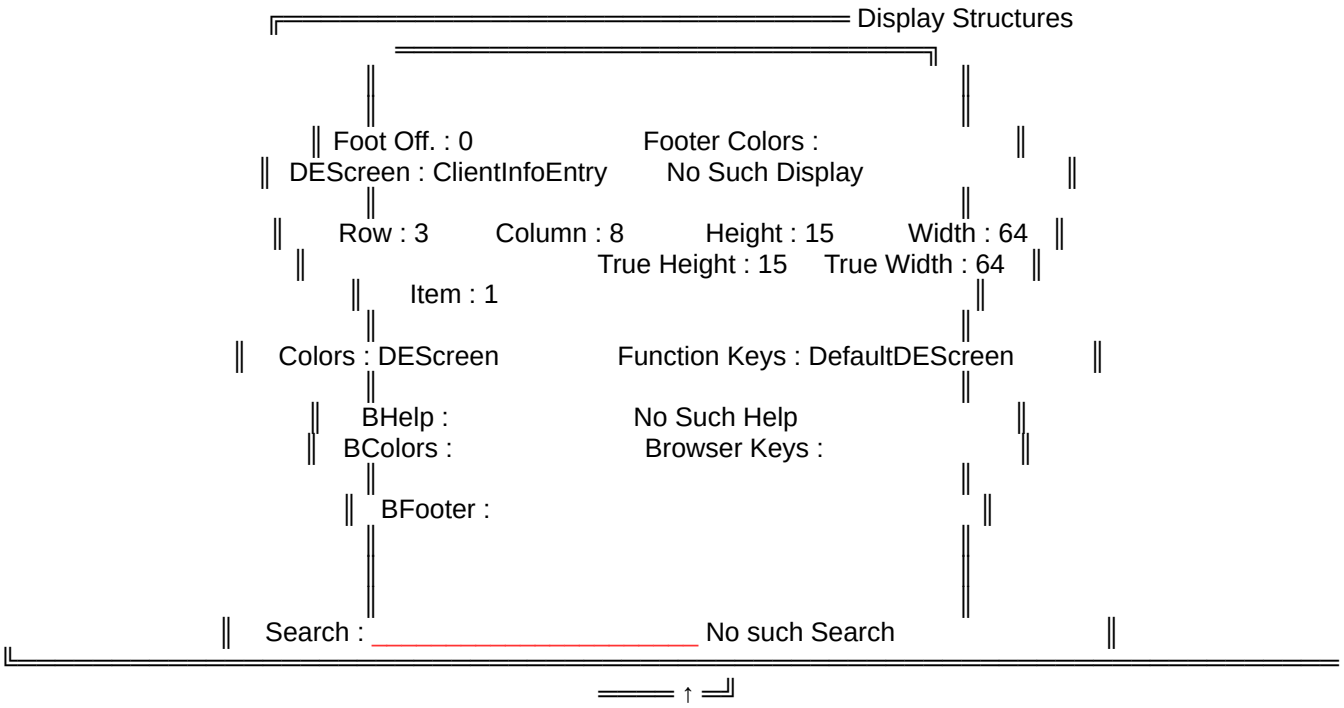
The **True Height** and **True Width** fields should always match the Height and Width fields for browsers. Only on oversized data entry screens and reports should the True values exceed the displayed values. True Height and True Width represent the absolute size of a display structure, and will nullify all values outside of their range. Set these values to **15** and **58**, respectively.

Item Height refers to how many rows will be used by each record in the browser. Only on browsers and sub-reports should this number be non-zero. Enter **1**.

Colors refers to the name of the color structure to use when the system is in Browse Entry Mode. **<F2>** will generate a PickList of available colors. In this case, select **DEScreen**.

Function Keys contains the name of the Function-Key group to use when the system is in Browse Entry Mode. **<F2>** will generate a PickList of available Function Key groups. For this example, select **Default DEScreen**.

When the cursor advances past the Function Keys field, the Data Entry Window will scroll, displaying other fields that did not fit on the previous view. The balance of the Display Structure window is shown below. To have this display similarly on the computer screen, press **<Ctrl-Pg Dn>**.



> **Note:** The upward-pointing arrow on the lower-right hand corner of the window border indicates that there is more on this data entry screen above the window's current viewing area.

For the next part of this exercise, use the **<Up Arrow>** key to move the cursor back to the BHelp Prompt.

If a particular help topic exists that would apply to this Browser, Enter the topic name in **BHelp**.

Example System

Step 2: Design the Browser

<F2> can provide a PickList of help topics. For this example, Enter **Browsers**.

BColors refers to the name of the color structure to use when the Browser is in use. **<F2>** will generate a PickList of available colors. In this case, select **Browser**.

Example System

Step 2: Design the Browser

Browser Keys contains the name of the Function-Key group to use when the system is in Browse (standard) Mode. **<F2>** will generate a PickList of available Function Key groups. For this example, select **Default Browser**.

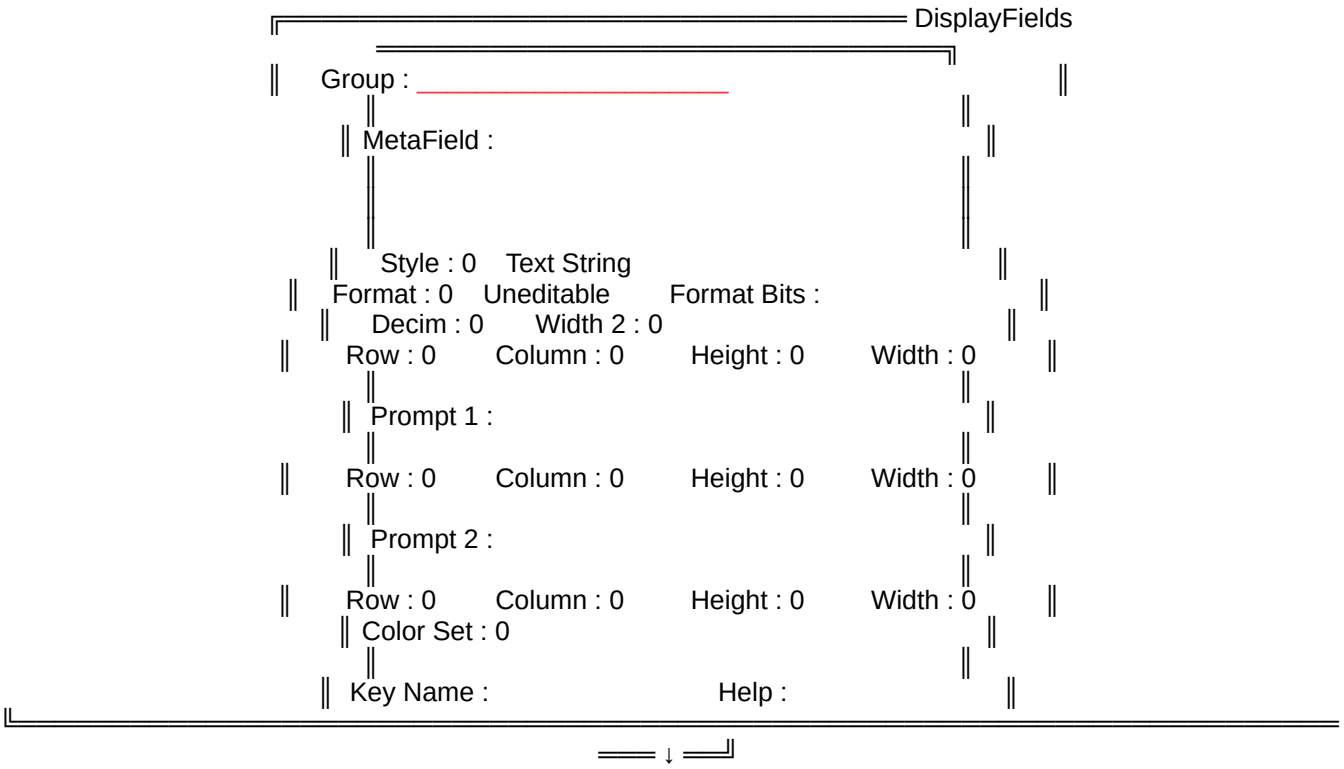
The **BFooter** contains the cue lines to display when the browser is active in Standard (non-Entry) Mode. **<Enter>** past this field.

The **Search** field contains the name of the Search used to limit the browser. **<Enter>** past this field. **<F2>** will generate a PickList of all available Search Structure names to select from. **<F4>** will generate a BrowseList, which acts like a PickList, but allows Browser functions, such as record adding, within the same process.

Press **<Ctrl-Enter>** to save the display structure record, and then **<Esc>** to leave the display structure browser and return to the VariForm System submenu.

While the display structure record has established that a browser named "ClientInfoBrow" exists, VariForm needs to know what information to display for each record. This information is stored in the Display Fields database.

To enter the Display Fields, select the "Field Records" line on the VariForm System / System submenu, either by pressing **<F>**, or using the <arrow> keys and pressing **<Enter>**. This will bring up the Display Field Records browser. Press **<F3>** to bring up a blank record for editing. There will be one display field record for each field on the browser.



Group is the field that attaches this record to the Display Structure system (where the matching field is called "Display". All field records for this browser will be in the

"ClientInfoBrow" group.

MetaField contains the name of the database field to display at this location. The first field will be "ClientID".

Example System

Step 2: Design the Browser

The **Style** field determines how the field's contents will be translated into display text: must a number be converted to text, or a Yes/No scenario, et cetera. **<F2>** brings up a PickList of available field styles. The first record of this example will be a style 2, or Character String (ASCIIZ) field.

Format tells VariForm about any special formatting requirements or limitations to put on the field. This is actually a fully used set of bit fields. **<Enter>** past this field.

The **Format Bits** field is nested, and pressing **<Enter>** will bring up a subscreen that delineates all of the yes/no combinations that make up the Format Field. Turn the "Editable" option to "Yes", and **<Ctrl-Enter>** to save the subscreen. The Format field will redisplay as a 1 - Editable.

Decim, or the decimals field, is used for floating-point numbers, multiline character fields, multi-bit fields, and nested fields. **<Enter>** past this field.

Width2 is used for multi-bit fields only. **<Enter>** past this field.

The Location and Size Fields, **Row**, **Column**, **Height**, and **Width**, determine the position and size of this field, with all numbers relative to this screen only.

In browsers, display fields have their prompts on one set of rows, and their fields on another set. Since the record space (item height) is 1 for this example, all prompts will be on row 0, while all fields will be on row 1. If this were a record space 2 browser (2 lines per record), the prompts would be on rows 0 and 1, with the corresponding fields on rows 2 and 3.

For this example, (field) Row will be 1, Column will be 1 (indent 1 from edge), Height will be 1 (always 1, except for multiline fields), and width will be 6.

Prompt1 contains the text of the primary field prompt, which will be "Client" for this example.

The next set of Location and Size Fields, **Row**, **Column**, **Height**, and **Width** refer to the positioning of the Prompt. For this example, (prompt1) Row will be 0, Column will be 1 (keep prompts left-aligned with character strings), Height will be 1, and Width will be 6.

The alternative prompt information: **Prompt2**, **Row**, **Column**, **Height**, and **Width** refer to a second prompt for data entry screens, or a trigger character for menus, but are generally not used on browsers. **<Enter>** past these fields.

Color Set contains the numeric value of the alternative coloring choices to use for the alternative prompt. Valid values are between 0 and 4, inclusive. **<Enter>** past this field.

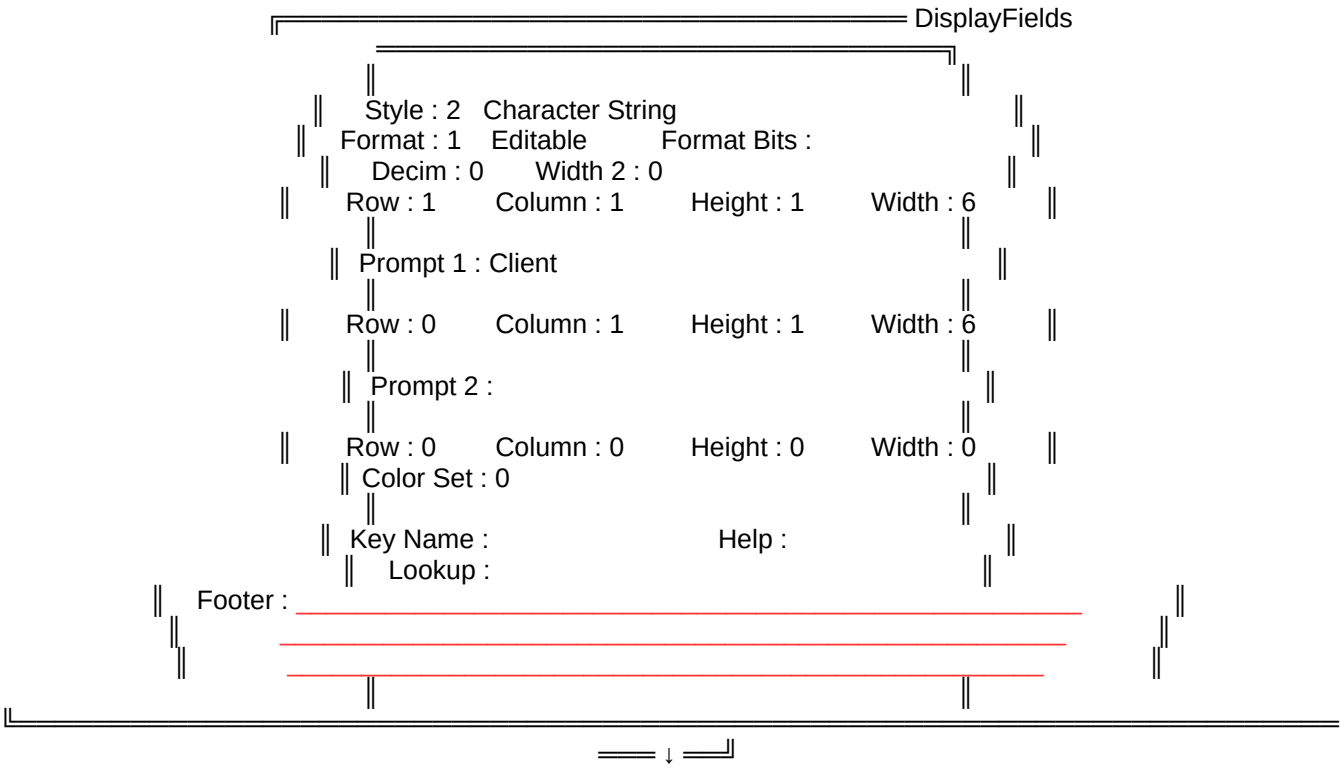
The **Key Name** field contains the group name for function keys associated with this specific field. This field is only effective when the browser is in Entry Mode. **<Enter>** past these fields. **<Enter>** past this field.

Help is for a field-specific response to the **<F1>** (help) key, when it is pressed on this specific field. This field is only effective when the browser is in Entry Mode. **<Enter>** past this field. This will cause the window to scroll.

Example System

Step 2: Design the Browser

To see the rest of the display fields data entry screen, press **<Ctrl-PgDn>**. The resulting window should appear as follows:



The **Lookup** field contains the name of the Search Structure to use to perform lookups on the MetaField. This process happens automatically whenever the field is changed. On Browsers, lookups can cause dramatic slowdowns on the browser's movement and redrawing speed. **<Enter>** past this field.

The **Footer** field contains the field-specific cue lines that will display when this MetaField field is made current. This field is only effective when the browser is in Entry Mode. **<Enter>** past this field. This will cause the window to scroll back to the top.

Press **<Ctrl-Enter>** to save this record, and the display field browser will become active.

Positioning the records in a single-line browser is done by adding the previous field's column and width, plus one, to get the next field's starting column.

The values for the other field records are:

MetaField	Style	Format	Column	Width	Prompt 1 Text	Width
Last Name	2	1	8	20	Last Name	9
Zip	2	1	29	12	Zip Code	8
FirstPurch	7	49	42	10	1st Purch.	9
SalesRegion	2	1	53	4	Reg.	4

Enter and save these field records, all with a Group Name of ClientInfoBrow, and then **<Esc>** from the Display Field Browser. This concludes section two of the example system: the database and a basic browser have been entered into the system.

Example System

Step 2: Design the Browser

Test the Browser

To make sure that the browser and the database structures are correct, there is a simple way to load the browser: call it from the Command Line Interface. From the VariForm Menu area, or any System-related Default DE Screen, press **<F3>**, to bring up the Command Line Interface windows.

The cursor will be located on the bottom window. Type **Browser("ClientInfoBrow")**, and press **<Ctrl-Enter>**. If all goes well, a 58-column by 15 row browser should appear in the center of the viewing area. If there are error messages, see Appendix 2 (page) for a list of corrective actions. Eventually, a browser will come up, or abort out.

If the browser appears, press **<Esc>** to leave the browser.

The Command Line Interface will ask for 'any key to continue'. Press **<Space>**, and the CLI Windows will redisplay. Press **<Esc>** to leave the Command Line Interface.

Take whatever corrective actions are necessary. Fix the bad field offsets in the database of database fields browser, for example, and re-run the quick test until the browser appears without error messages. Once this has been accomplished: once the browser can be accessed 'cleanly', it can be added to the menu system.

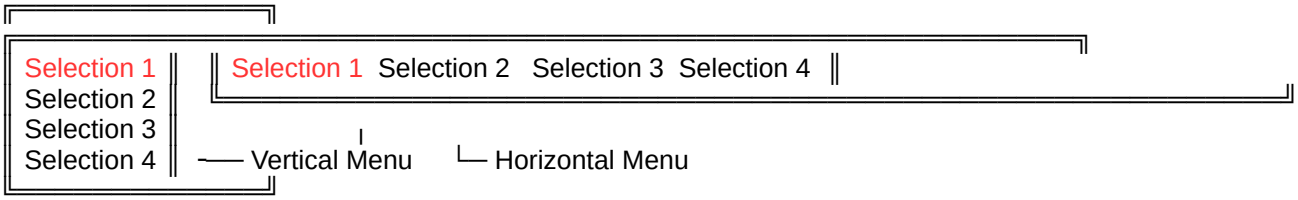
Example System

Step 3: Add a Menu for the System

This addition will require two steps. First, a menu for all of the example data systems will be added to the VariForm System menu. Second, a line to call the newly-entered browser will be added to the menu.

Menu Strategy

A menu is a special use of the display system: it is a list of "Text String" fields, which display the contents of the prompts. When the user makes a selection, the contents of the MetaField entry of the selected field record will be processed. This allows menus to run procedures, call reports, open browsers, and so forth.



There are two types of menus: Vertical and Horizontal. The difference between them is the arrow keys that move the menu's cursor bar. Vertical menus use the <Up> and <Down> arrow keys, while horizontal menus use <Left> and <Right>. <Left> and <Right> on vertical menus 'jump' to the adjacent menu, while <Up> and <Down> on horizontal menus act like <Enter>, selecting the current menu item.

One other difference between the two types of menus is the default starting position. horizontal menus start on the left edge, while vertical menus start at the top. Finally, as a point of style, horizontal menus do not generally have borders, while vertical menus do.

Example System

Step 3: Add a Menu for the System

Adding a Menu

Adding a menu is similar to adding a browser with one extra step: menus have to be called from somewhere. Users do not appreciate having to drop to the Command Line Interface every time they want to access a few browsers. As with all other systems, a little forethought can save a lot of editing time later.

The original example problem asked for four databases: Client Info, Sales History, Items, and Sales Region. It also asked for total purchases calculations for all clients, and some reports. 4 Databases + 1 separation line + 1 calculation + 1 submenu for reports = 7 menu items. The new menu should be 7 rows high. The longest name will probably be "Sales History", which should have a space before and after it: 13 characters + 2 spaces = 15 columns wide.

Bring up the **Display Structures** browser, and press **<F3>** to create a blank record for editing.

The **Name** of the structure must be unique, to keep it separate from other data structures. It is also a good idea to use naming conventions that help the designer remember at a glance what each structure does. In this example, the name "ExampleMenu" makes sense, since it is the Menu for the Example System.

The **Database** field refers to the database the browser will be editing. Since this is a menu, no database will be used. Either leave this field blank, or type **Virtual**, which is the name of the in-memory database that maintains the virtual field list.

Usage refers to how the Display Structure will be used: as a database, browser, data entry screen, et cetera. Since this is a menu, type **5** and **<Enter>** past this field.

The **Display** field is used to allow different display structures, such as a limited and unlimited browser, to share the same display fields. Press **<F3><Enter>**.

If a particular **Help** topic exists that would apply when this Menu is current, Enter the topic name here. **<F2>** can provide a PickList of help topics. For this example, type **Menu System**, and press **<Enter>**.

The **Title** is the text that will be displayed, automatically centered, on the top of the display window's border, if one exists. **<Enter>** past this field.

The **Footer** contains the cue lines to display when this menu is current. Type **Example System from the Manual** and press **<Enter>**.

Footer Offset is how many lines (0-4) above the base line to display the screen's cue lines. **<Enter>** past this field.

The **Footer Colors** field contains the name of the color structure record to use when drawing the cue lines, even if there are no cue-lines to draw. **<F2>** provides a PickList of color structures to select from. The default for vertical menus is "HelpPlain" which refers to the fact that there is no separation line between the Screen- and Field-specific cue lines.

The **DEScreen** field is only for browsers. **<Enter>** past this field.

Example System

Step 3: Add a Menu for the System

The next four fields require a little mathematics for aesthetics: they refer to the location and size of the menu on the computer screen, excluding borders. Since this menu will be attached to the horizontal VariForm System menu, the location should be 2 lines down from the top (allowing for a border), and at column 60 (lining up with the VariForm System line to be inserted).

Row : **2**
Column : **60**
Height : **7**
Width : **15**

The **True Height** and **True Width** fields should always match the Height and Width fields for menus. Only on oversized data entry screens and reports should the True values exceed the displayed values. True Height and True Width represent the absolute size of a display structure, and will nullify all values outside of their range. Set these values to **7** and **15**, respectively.

Item Height is for browsers and sub-reports. **<Enter>** past this field.

Colors refers to the name of the color structure to use on this menu. **<F2>** will generate a PickList of available colors. In this case, select **MenuVertical**, which will provide consistant colors with the surrounding menus, and a double-line border around the menu.

Function Keys contains the name of the Function-Key group to use when the system is in Browse Entry Mode. **<F2>** will generate a PickList of available Function Key groups. For this example, select **Default Menu**.

The balance of the fields on the Display Structure entry screen are for browsers and reports. Press **<Ctrl-Enter>** to save the display structure record, and then **<Esc>** to leave the display structure browser and return to the VariForm System submenu.

The display structure record has established that a menu named "ExampleMenu" exists. VariForm also needs to know what information to display on the menu. This information is stored in the Display Fields database.

Example System

Step 3: Add a Menu for the System

Adding a Menu Line

The term 'menu line', is a technical misnomer: it should be called 'a menu selection or unit of displayed area', but for some reason, the phrase 'menu line' is easier to remember. Each menu line, whether it is a literal line, such as on a vertical menu, or a block of columns, as on horizontal menus, is represented by a record in the Display Fields database.

To enter the Display Fields, select the "Field Records" line on the VariForm System submenu, either by pressing **<F>**, or using the <arrow> keys and pressing **<Enter>**. This will bring up the Display Field Records browser. Press **<F3>** to bring up a blank record for editing. There will be one display field record for each line on the menu. Only two lines are currently needed: the line that will call the ClientInfoBrow browser, and the line that will visually separate the browser calls from everything else.

Group is the field that attaches this record to the Display Structure system (where the matching field is called "Display". All field records for this menu will be in the "ExampleMenu" group.

MetaField contains the line that will be processed when this item is selected. For this line, the item will be: **Browser("ClientInfoBrow")**. This will call the ClientInfoBrow browser, when this menuline is selected.

The **Style** field determines how the field's contents will be translated into display text: must a number be converted to text, or a Yes/No scenario, et cetera. All menu lines are Text Strings, so leave this field at **0**.

Format tells VariForm about any special formatting requirements or limitations to put on the field. Since this is a "Text String" field, leave the value at **0**.

The **Format Bits** field is nested, so go past this field with **<Tab>**.

Decim, or the decimals field, indicates whether this line item is 'blocked', or totally inaccessible. Inaccessible lines are usually headings, explanations or separation lines, and are signified by a -1 in this field. Since the user will want access to this field, leave it a **0**.

Width2 is used for multi-bit fields only. **<Enter>** past this field.

The Location and Size Fields, **Row**, **Column**, **Height**, and **Width**, determine the position and size of this field, with all numbers relative to this screen only.

In Menus, display field positioning simply maintains the order of the choices within the menu. For this example, (field) Row and Column will be 0, to keep this line first, Height will be 0 (always 0 or 1, except for multiline fields), and width will be 0 (no field to draw).

Prompt1 contains the text of the primary field prompt, which will be " Client Info" for this example. Note the leading space : it will help visually separate the text from the border. Also, each choice line (displayfield record) should be the full width of the menu's window, so that the cursor bar can move smoothly up and down the menu without changing size.

Example System

Step 3: Add a Menu for the System

The next set of Location and Size Fields, **Row**, **Column**, **Height**, and **Width** refer to the positioning of Prompt1. For this example, (prompt1) Row will be 0, Column will be 0 (keep prompts left-aligned on vertical menus), Height will be 1, and Width will be 15.

Prompt2 has a special use for menus: it represents the 'hot key' for this menu selection. It should correspond to one character from prompt1, and be positioned similarly. For this example, Prompt2 will contain the letter '**C**'.

The alternative prompt positioning: **Row**, **Column**, **Height**, and **Width** should be 0, 1, 1, and 1, respectively. Column 1 overlays the second character (the C) of prompt1, since prompt1 begins column 0.

Color Set contains the numeric value of the alternative coloring choices to use for the alternative prompt. By selecting an alternative color set for prompt2, the 'hot key' will show up as a different color on the menu. Valid values are between 0 and 4, inclusive. Most VariForm menus use color set 1 for menus.

The **Key Name** field contains the group name for function keys associated with this specific field. This field is only effective when this selection is current, and is not usually used on VariForm menus. **<Enter>** past this field.

Help is for a field-specific response to the **<F1>** (help) key, when it is pressed on this specific field. This field is only effective when this selection is current. **<Enter>** past this field.

The **Lookup** field contains the name of the Search Structure to use to perform lookups on the MetaField. Since this is meaningless on menu lines (there are no fields), **<Enter>** past this field.

The **Footer** field contains the field-specific cue lines that will display when this selection field is current. **<Enter>** past this field. This will cause the window to scroll back to the top.

Press **<Ctrl-Enter>** to save this record, and the display field browser will become current.

The second field is a 'separation line', which visually breaks up the menu into two parts: the upper portion will be for selecting browsers, and the lower portion will be for reports and calculations. To put the line on the menu, add a display field record with the following values:

Group	"ExampleMenu"
MetaField	Leave Blank
Style	0
Format	0
Format Bits	Leave Blank (**<Tab>** to advance)
Decim	-1 to make the line inaccessible
Width2	0
Row	4
Column	0
Height	0
Width	0

Example System

Step 3: Add a Menu for the System

Prompt1	---------- (15 minus signs or dashes)
Row	4
Column	0
Height	1
Width	15
All others	Leave blank, or as value 0

Press **<Ctrl-Enter>** to save this record, and the display field browser will become current.

Example System

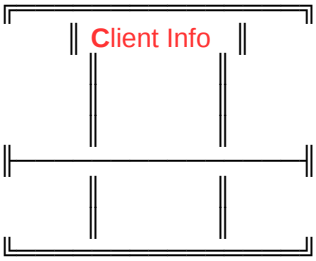
Step 3: Add a Menu for the System

Test the New Menu

To make sure that the size and colors are correct, there is a simple way to load the menu: call it from the Command Line Interface. From the VariForm Menu area, or any System-related Default DE Screen, press **<F3>**, to bring up the Command Line Interface windows.

The cursor will be located on the bottom window. Type **Menu("ExampleMenu")**, and press **<Ctrl-Enter>**. If all goes well, a 15-column by 7 row menu should appear in the upper right region of the viewing area. If there are error messages, see Appendix 2 (page) for a list of corrective actions. Eventually, the menu will either come up, or abort out.

If the menu appears, it should resemble the following;



If the menu appears, press **<Enter>** to test the browser call. If the browser appears, everything is A-OK. Press **<Esc>** to leave the browser, and **<Esc>** again to leave the menu.

The Command Line Interface will ask for 'any key to continue'. Press **<Space>**, and the CLI Windows will redisplay. Press **<Esc>** to leave the Command Line Interface.

Take whatever corrective actions are necessary. Fix the spelling on the browser call, for example, or correct the size entries, and re-run the quick test until the menu appears without error messages. Once this has been accomplished: once the menu can be accessed 'cleanly', and it can call the browser without complaint, this menu can be added to the VariForm menu.

Example System

Step 3: Add a Menu for the System

To add this menu to the VariForm menu, add a display field record with the following values:

Group	"System"
MetaField	Menu("ExampleMenu")
Style	0
Format	0
Format Bits	Leave Blank (**<Tab>** to advance)
Decim	0 to make the line inaccessible
Width2	0
Row	0
Column	61
Height	0
Width	0
Prompt1	" Menu Example " (quotes are to show spaces: do not enter them)
Row	0
Column	61
Height	1
Width	14
Prompt2	M
Row	0
Column	61
Height	1
Width	1
All others	Leave blank, or as value 0

Press **<Ctrl-Enter>** to save this record, and the display field browser will become current.

Do not expect the menu to automatically change after the last record has been saved. The VariForm System submenu must be re-loaded from disk for the change to take effect. This is accomplished by pressing **<Esc>** to leave the System submenu, and **<Esc>** again to leave the VariForm menu. This returns control to the small, centered 'Main' menu.

Select VariForm from the main menu, and the VariForm menu should now look like this:

The Example menu can now be selected either by pressing **<M>**, or moving the cursor bar to the 'Menu Example' section, and pressing either **<Enter>** or **<Down Arrow>**. This will cause the new menu to appear directly beneath the "Menu Example" phrase. This concludes section three of the example system: the menu that calls the database browser has been entered into the system, with room for the other databases.

Example System

Step 4: Design the Data Entry Screen

Browsers allow users to 'paint with a broad brush': users can view, delete, and copy existing records, as well as edit any fields displayed by the browser. Most databases have more fields than would readily display on a browser, and character fields can easily exceed the screen width.

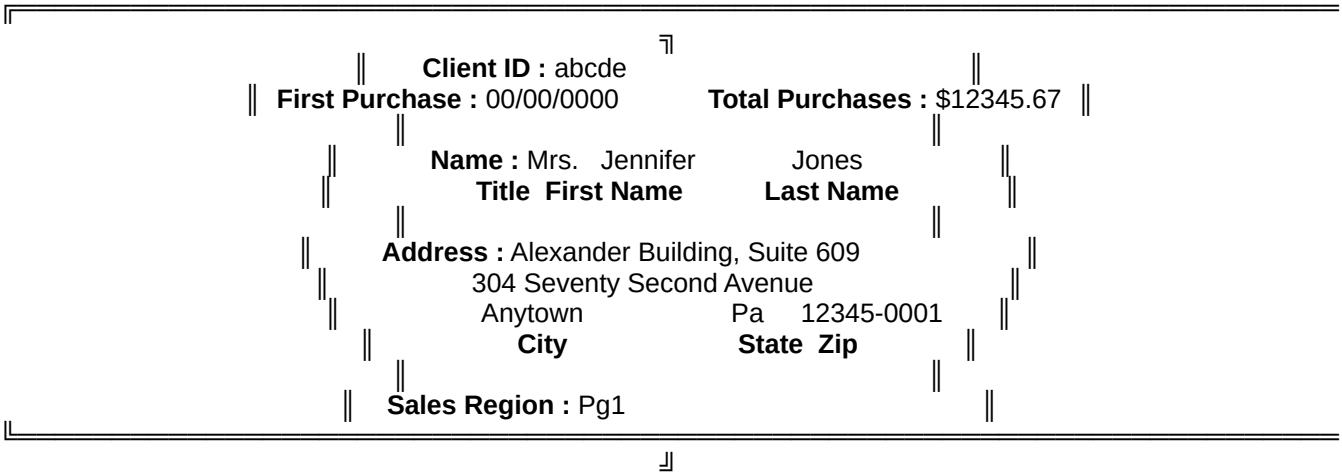
Data Entry Screens allow the user to perform sophisticated, detailed editing on a single database record at a time, with no limit on the number or size of the fields being edited. Browsers scroll to allow the user to view other records, while Data Entry Screens scroll to allow the user to view other fields within a record, if there are too many fields to fit in the display window at one time.

Data Entry Screen Strategy

A data entry screen was the original basis for design of the Display Structure / Display Field system: to provide a straightforward, data-based method for designing data entry screens for sophisticated user needs. Each data entry screen is described by a single Display Structure entry. The content of the screen is described by Display Fields.

The best way to design the data entry screen is to draw a picture of what it should look like, and use that drawing to determine placement and sizing for the fields. Often, a designer lays out is presented with some initial drawings of data entry screens to determine what the database needs to contain.

A general drawing of the Client Info data entry screen would be as follows, with the prompts in bold, for visibility:



Points of aesthetics:

- Whenever possible, the longest field prompt of the left-hand column should be at or near the top of the data entry screen.

- Prompts should be organized into columns, to enable experienced users to move

around quickly with the **<Up Arrow>** and **<Down Arrow>** keys.

● Fields with related topics, such as elements of a name, or parts of an address, should be grouped together, and often look better if they 'share' a prompt for the group.

Example System

Step 4: Design the Data Entry Screen

Return to the VariForm System submenu, and bring up the **Display Structures** browser, and press **<F3>** to create a blank record for editing.

The **Name** of the structure must be unique, to keep it separate from other display structures. It is also a good idea to use naming conventions that help the designer remember at a glance what each structure does. In this example, the name "ClientInfoEntry" makes sense, since it is the Data Entry screen for the Client Info database. The name must match the corresponding entry in the browser that will call this screen (see page : this name has already been entered in the browser).

The **Database** field refers to the database the data entry screen will be editing. Type **ClientInfo**, and press **<Enter>**. This field should always match the database name in the corresponding browser.

Usage refers to how the Display Structure will be used: as a database, browser, data entry screen, et cetera. Since this is data entry screen, type **1** and **<Enter>** past this field.

The **Display** field is used to allow different display structures, such as a limited and unlimited browser, to share the same display fields. Press **<F3><Enter>**, to copy the Display Structure name here.

If a particular **Help** topic exists that would apply for this entire data entry screen, Enter the topic name here. **<F2>** can provide a PickList of help topics. For this example, type **Data Entry**, (there is a space between "Data" and "Entry") and press **<Enter>**.

The **Title** is the text that will be displayed, automatically centered, on the top of the display window's border, if one exists. Type **Client Information** and press **<Enter>**.

The **Footer** contains the cue lines to display when this menu is current. Type **Client Information Data Entry Screen** and press **<Enter>**.

Footer Offset is how many lines (0-4) above the base line to display the screen's cue lines. To prevent the data entry screen's cue lines from moving vertically, type **2** and press **<Enter>**.

The **Footer Colors** field contains the name of the color structure record to use when drawing the cue lines, even if there are no cue-lines to draw. **<F2>** provides a PickList of color structures to select from. The default for vertical menus is "HelpPlain" which refers to the fact that there is no separation line between the Screen- and Field-specific cue lines.

The **DEScreen** field is only for browsers. **<Enter>** past this field.

The next four fields require a little mathematics for aesthetics: they refer to the location and size of the data entry window on the computer screen, excluding borders. This should be calculated to center the window on an 80 column by 21 row viewing area. From the drawing, the screen should be 12 Rows of 70 columns. To calculate the upper-left hand corner position, subtract the size of the window from the size of the computer screen, and divide the result by two:

Row : **4** (21 - 12) / 2 = 4.5 ≈ 4

Example System

Step 4: Design the Data Entry Screen

Column : **5** (80 - 70) / 2 = 5
Height : **12**
Width : **70**

The **True Height** and **True Width** fields should match the Height and Width fields whenever possible. Only on oversized data entry screens (where the number of rows is greater than about 20), and on reports should the True values exceed the displayed values. True Height and True Width represent the absolute size of a display structure (the screen size), and will nullify all values outside of their range. Set these values to **12** and **70**, respectively.

Item Height is for browsers and sub-reports. **<Enter>** past this field.

Colors refers to the name of the color structure to use on this window. **<F2>** will generate a PickList of available colors. In this case, select **DEScreen** which will provide consistant colors with the other data entry screens, and a double-line border around the window.

Function Keys contains the name of the Function-Key group to use when the current data entry field does not support the Function Key pressed. **<F2>** will generate a PickList of available Function Key groups. For this example, select **Default DEScreen**.

The balance of the fields on the Display Structure entry screen are for browsers and reports. Press **<Ctrl-Enter>** to save the display structure record, and then **<Esc>** to leave the display structure browser and return to the VariForm System submenu.

The display structure record has established that a data entry screen named "ClientInfoEntry" exists. VariForm also needs to know what information to display on the screen. This information is stored in the Display Fields database.

To enter the Display Fields, select the "Field Records" line on the VariForm System submenu, either by pressing **<F>**, or using the <arrow> keys and pressing **<Enter>**. This will bring up the Display Field Records browser. Press **<F3>** to bring up a blank record for editing. There will be one display field record for each field to be edited.

Group is the field that attaches this record to the Display Structure system (where the matching field is called "Display"). All field records for this menu will be in the "**ClientInfoEntry**" group.

MetaField contains the name of the field that will be edited through this record. For this field, the item will be: **ClientID**. This will access information from the ClientID field of the current database record.

The **Style** field determines how the field's contents will be translated into display text: must a number be converted to text, or a Yes/No scenario, et cetera. All data entry screen fields' styles should the corresponding database's field type. In this case, enter a **2** for character

string.

Format tells VariForm about any special formatting requirements or limitations to put on the field. Since this is field should be editable change the value to **1**.

The **Format Bits** field is nested, so go past this field with **<Tab>**.

Decim, controls formatting for multibit, multiline, and floating-point fields. Since ClientID is none of these, leave the field at **0**.

Width2 is used for multi-bit fields only. **<Enter>** past this field.

The Location and Size Fields, **Row**, **Column**, **Height**, and **Width**, determine the position and size of this field, with all numbers relative to this screen only. According to the "Drawing" made previously (page), the field starts at row 0, column 19, 1 line high, 5 characters wide. Remember to count the first space inside a border as 0.

Prompt1 contains the text of the primary field prompt, which will be "**Client ID :**" for this example.

The next set of Location and Size Fields, **Row**, **Column**, **Height**, and **Width** refer to the positioning of Prompt1. For this example, (prompt1) Row will be 0, Column will be 7 (leave a space between the prompt and the field), Height will be 1, and Width will be 12. While these numbers can be derived from the drawing, they can be automatically generated by pressing **<F6>** on the Prompt1Row field.

Prompt2 will not be used for this field, so enter past it, **Row**, **Column**, **Height**, **Width** and **Color Set**.

The **Key Name** field contains the group name for function keys associated with this specific field. This field is only effective when this selection is current, and is not usually used on VariForm menus. **<Enter>** past this field.

Help is for a field-specific (read: context-sensitive) response to the **<F1>** (help) key, when it is pressed on this specific field. This field is only effective when this selection is current. **<Enter>** past this field.

The **Lookup** field contains the name of the Search Structure to use to perform lookups on the MetaField. **<Enter>** past this field.

The **Footer** field contains the field-specific cue lines that will display when this selection field is current. **<Enter>** past this field. This will cause the window to scroll back to the top.

Press **<Ctrl-Enter>** to save this record, and the display field browser will become current.

Example System

Step 4: Design the Data Entry Screen

The other display field records are enumerated below, followed by an explanation of the "Shared Prompt" concept. The fields are listed in the order they appear in the database structure.

Group	ClientInfoEntry	ClientInfoEntry	ClientInfoEntry
MetaField	Title	FirstName	LastName
Style	2 Character String	2 Character String	2 Character String
Format	1 Editable	Use Format Bits	Use Format Bits
F. Bits	Skip this field	Editable, Capitals	Editable, Capitals
Decim	0	0	0
Width2	0	0	0
Row	3	3	3
Column	19	26	48
Height	1	1	1
Width	5	19	19
Prompt1	Title	First Name	Last Name
Row	4	4	4
Column	19	26	48
Height	1	1	1
Width	5	10	9
All others	See "NameShare"	See "NameShare"	See "NameShare"

Group	ClientInfoEntry	ClientInfoEntry	ClientInfoEntry
MetaField	Address1	Address2	City
Style	2 Character String	2 Character String	2 Character String
Format	1 Editable	1 Editable	1 Editable
F. Bits	Skip this field	Skip this field	Skip this field
Decim	0	0	0
Width2	0	0	0
Row	6	7	8
Column	19	19	19
Height	1	1	1
Width	39	39	25
Prompt1	Address :	Address :	City
Row	6	6	9
Column	9	9	19
Height	1	1	1
Width	9	9	4
All others	Blank / 0	Blank / 0	See "Address"

Group	ClientInfoEntry	ClientInfoEntry	ClientInfoEntry
MetaField	State	Zip	SalesRegion
Style	2 Character String	2 Character String	2 Character String
Format	1 Editable	1 Editable	1 Editable
F. Bits	Skip this field	Skip this field	Skip this field
Decim	0	0	0
Width2	0	0	0
Row	8	8	11

Example System

Step 4: Design the Data Entry Screen

Column	36	41	19
Height	1	1	1
Width	3	11	3
Prompt1	State	Zip	Sales Region :
Row	9	9	11
Column	35	41	4
Height	1	1	1
Width	5	3	14
All others	See "Address"	See "Address"	Blank / 0

Example System

Step 4: Design the Data Entry Screen

Group	ClientInfoEntry	ClientInfoEntry
MetaField	FirstPurch	TotalPurch
Style	7 Date	4 Standard Numeric
Format	Use format bits	Use format bits
F. Bits	See "Date"	See "Purchase"
Decim	0	2
Width2	0	0
Row	2	2
Column	19	59
Height	1	1
Width	10	11
Prompt1	First Purchase :	Total Purchases :
Row	2	2
Column	2	41
Height	1	1
Width	16	17
All others	Blank / 0	Blank / 0

Prompt Sharing, etc.

When several fields are grouped together, they may need to have a common prompt. This is true on the Client Info data entry screen, as well as the VariForm Color Structure data entry screen. This leads to the concept of 'Shared' prompts: several fields that have prompts with the same content in the same location.

When VariForm is drawing a data entry screen, it will redraw all the fields. When the user is performing data entry within the screen, however, only the current field's prompt is re-drawn. This can create the illusion of a 'disappearing' prompt, a 'constant' or regional prompt, or a prompt that changes color. All are useful visual effects that can help the end user to understand what field is current at a given point in time.

The point of the next two additions is to show various uses for the alternate prompt (prompt2):

NameShare: Since Title, FirstName, and LastName are all part of a 'name', they should all light up a communal prompt2, indicating this. To do so, give all three fields a Prompt2 of "Name :", on Row 3, at Column 12, with a Height of 1 and a Width of 6. Leave the ColorSet field at 0.

Address: Since Address1, Address2, City, State, and Zip are all part of an 'address', they should all have access to a common prompt as well. Since 2 fields are actually the 'Address' lines, they should light up 'Address' as a normal prompt. The other three fields should light up Address in a different way. City, State, and Zip need a Prompt2 of "Address :" at Row 6, Column 9, Height 1, Width 9, Colorset 1, 2, and 3. This will provide a different color for the Address prompt for each of City, State, and Zip.

Extra Information about Field Format Bits:

Date: Date fields are usually Editable, Padded with zeroes, using slash (/) separators. All of these options are available on the Format Bits nested field. Set Editable, and Padded to "Yes",

and make sure the Pad Character is the code that means zero, and the Separator Character is the code that means slash. Both of the character fields have the <F2> key generating appropriate PickLists.

Purchase: Most floating point numbers need some decimals, and in this case, a dollar sign would also be nice (see drawing, page). Set Editable, Decimals, and Dollar Sign to "Yes" on the Format Bits sub-screen, and make sure the Dollar Character is the code that means $. The Dollar Character field has an <F2> PickList generator that lists all possible dollar characters.

Once these entries have been correctly made, step 4 is almost complete: all inputs into this database have been entered. All that remains of this step is to test the data entry screen.

Press **<Esc>** from the Display Fields Browser, and **<Esc>** from the VariForm System submenu. Go to the Menu Example submenu, either by pressing **<M>**, or by moving the cursor to the Menu Example entry, and pressing **<Enter>** or **<Down Arrow>**.

To test the data entry screen, select the menu entry for this database (for this example, select "Client Info" from the menu), and when the Browser comes up, press **<F3>** to bring up a blank data entry screen. Type in a record (make up a name, address, et cetera). Look for the following potential problems:

- Are the prompts lined up, and changing colors appropriately
- Are the field characters (extra underscore characters) touching adjacent prompts
- Do the First Name and Last Name fields correctly capitalize data entry
- Is the Total Purchases field supporting and truncating input to 2 decimals

Once these problems have been addressed and answered, step 4 is complete: The target database has all standard inputs: Data Structure, Database Browser, Data Entry Screen.

Example System

Step 5: Repeat 1-4 until done

The original requirement (on page), has three other databases, which need database structures, browsers, and data entry screens. Take the following information, and input it to complete the database set. Test each step in this 'input cycle', as recommended in the previous steps: while data entry errors can always be fixed, prevention is still the best method of bug-fixing.

Data Set 2: Sales Regions

Database Structure

Database Name	SalesRegion
File Mask	SalesReg.Dat
Number of Buffers	2
Buffer Size	1024
Tree Height	11
Characteristics	0

Fields within a database

Database	OrderName	SizeOff. Type	Sub Type
SalesRegion	0 RegionID	6 0 0 Char / ASCIIZ	2
SalesRegion	1 RegionName	40 6 0 Char / ASCIIZ	0

Index 0	Abbrev	RegionID: 6
Index 1	Name	RegionName:40

Sales Regions Browser Structure

Name	SalesRegionBrow
Database	SalesRegion
Usage	0
Display	SalesRegionBrow
Help	Browser Entry
Title	Sales Regions
Footer	Browser Entry Mode: Sales Regions Reference
Footer Offset	0
Footer Colors	"HelpPlain"
DEScreen	SalesRegionEntry
Row	3
Column	15
Height	15
Width	50
True Height	15
True Width	50
Item Height	1
Colors	DEScreen

Example System

Function Keys DefaultDEScreen
BHelp Browsers
BColors Browser
Browser Keys DefaultBrowser
BFooter Sales Regions Reference Records
Search

Example System

Step 5: Repeat 1-4 until done

Sales Regions Browser Field Values

Remember: all browser fields will be in the SalesRegionBrow group, with Field Row 1, and Prompt1 Row 0.

MetaField	Style	Format	Column	Width	Prompt 1 Text	Width
RegionID	2	1	1	6	Region	6
RegionName	2	1	8	40	Name of Region	14

Sales Regions Menu Line

Group	ExampleMenu
MetaField	Browser("SalesRegionBrow")
Style 0	
Format	0
Decim	0
Width2	0
Row	1
Column	0
Height	0
Width	0
Prompt1	" Sales Region "
Row	1
Column	0
Height	1
Width	15
Prompt2	S
Row	1
Column	1
Height	1
Width	1
Color Set	1
Key Name	
Help	
Lookup	
Footer	Load the Sales Region Browser

Example System

Step 5: Repeat 1-4 until done

Sales Regions Data Entry Structure

Name	SalesRegionEntry
Database	SalesRegion
Usage	1
Display	SalesRegionEntry
Help	Data Entry
Title	Sales Regions
Footer	Sales Regions Data Entry Screen
Footer Offset	2
Footer Colors	HelpPlain
DEScreen	
Row	9
Column	15
Height	3
Width	50
True Height	3
True Width	50
Item Height	0
Colors	DEScreen
Function Keys	DefaultDEScreen
All Others	Blank or 0

Sales Region Data Entry Field Records

Group	SalesRegion	SalesRegion
MetaField	RegionID	RegionName
Style	2 Character String	2 Character String
Format	1	1
F. Bits		
Decim	0	0
Width2	0	0
Row	0	2
Column	9	9
Height	1	1
Width	3	39
Prompt1	Code :	Name :
Row	0	2
Column	2	2
Height	1	1
Width	6	6
All others	Blank / 0	Blank / 0

Example System

Step 5: Repeat 1-4 until done

Data Set 3: Item Information

Database Structure

Database Name ItemInfo
File Mask ItemInf.Dat
Number of Buffers 2
Buffer Size 1024
Tree Height 11
Characteristics 0

Fields within a database

Database	OrderName	SizeOff. Type	Sub Type
ItemInfo	0 ItemNumber	12 0	0 Char / ASCIIZ 2
ItemInfo	1 ItemDescription	200 12	0 Char / ASCIIZ 0
ItemInfo	2 ItemPrice	8 212	4 Floating Point 0

Index 0 Abbrev ItemNumber: 12
Index 1 Description ItemDescription: 50

Item Information Browser Structure

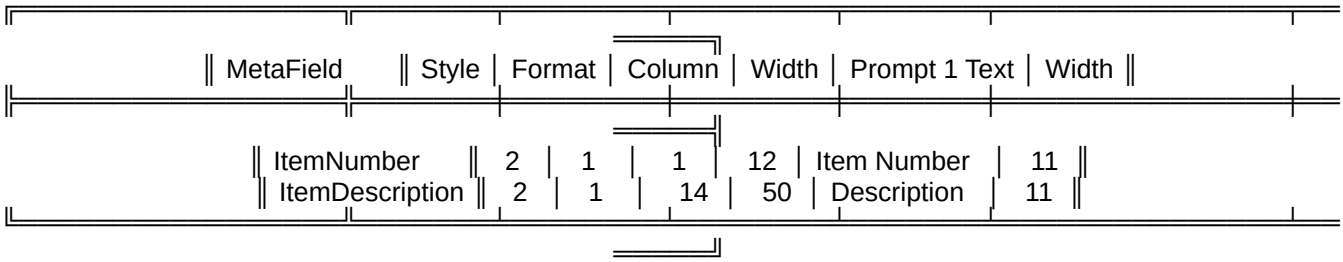
Name ItemInfoBrow
Database ItemInfo
Usage 0
Display ItemInfoBrow
Help Entry Mode
Title Item Information
Footer Entry Mode: Item Information Records
Footer Offset 0
Footer Colors "HelpPlain"
DEScreen ItemInfoEntry
Row 4
Column 1
Height 15
Width 78
True Height 15
True Width 78
Item Height 1
Colors DEScreen
Function Keys DefaultDEScreen
BHelp Browsers
BColors Browser
Browser Keys DefaultBrowser
BFooter Item Information Records
Search

Example System

Step 5: Repeat 1-4 until done

Item Information Browser Field Records

Remember: all browser fields will be in the ItemInfoBrow group, with Field Row 1, and Prompt1 Row 0.



Item Information Menu Line

Group		ExampleMenu
MetaField		Browser("ItemInfoBrow")
Style	0	
Format		0
Decim		0
Width2		0
Row		2
Column		0
Height		0
Width		0
Prompt1		" Item Info "
Row		2
Column		0
Height		1
Width		15
Prompt2		I
Row		2
Column		1
Height		1
Width		1
Color Set		1
Key Name		
Help		
Lookup		
Footer		Load the Item Information Browser

Example System

Step 5: Repeat 1-4 until done

Item Information Data Entry Structure

Name	ItemInfoEntry
Database	ItemInfo
Usage	1
Display	ItemInfoEntry
Help	Data Entry
Title	Item Information
Footer	Item Information Data Entry Screen
Footer Offset	2
Footer Colors	HelpPlain
DEScreen	
Row	8
Column	1
Height	5
Width	78
True Height	5
True Width	78
Item Height	0
Colors	DEScreen
Function Keys	DefaultDEScreen
All Others	Blank or 0

Item Information Data Entry Field Records

Group	ItemInfoEntry	ItemInfoEntry	ItemInfoEntry
MetaField	ItemNumber	ItemDescription	ItemPrice
Style	2 Character String	2 Character String	4 Standard Numeric
Format	1 Editable	1 Editable	Use Format Bits
F. Bits	Skip this field	Skip this field	Editable, Decimals Dollar Sign : $
Decim	0	0	2
Width2	0	67 (see below)	0
Row	0	2	0
Column	8	8	66
Height	1	3 (see below)	1
Width	11	200	10
Prompt1	No. :	Desc :	Price :
Row	0	2	0
Column	2	1	58
Height	1	1	1
Width	5	6	7
All others	Blank / 0	Blank / 0	Blank / 0

Multi-Line Fields: There are many situations where a very long text string is required: dos file pathnames, complex descriptions, road or driving directions, and so forth. While this information could be arranged into a set of shorter fields, like the address fields in Client Info, it may be desireable to have 'flowing' or 'wrap-around' text. This is accomplished by using a long character string field (70-299 characters), and designing the display as a multi-line field.

Example System

Step 5: Repeat 1-4 until done

Multi-Line fields are designed by entering a field height > 1. The total width to display is in the field Width, and the width of the individual lines is in the Decimals entry. This process works in all display field situations. The size of the data-entry layouts do not affect the contents of the string being entered. For best results, use a line width (decimals) of (total width / height) + 1 or 2.

Example System

Step 5: Repeat 1-4 until done

Data Set 4: Sales History

Database Structure

Database Name	SalesHistory
File Mask	SaleHist.Dat
Number of Buffers	2
Buffer Size	1024
Tree Height	11
Characteristics	0

Fields within a database

Database	Order	Name	Size	Off.	Type	Sub Type
SalesHistory	0	ClientID	6	0	0 Char / ASCIIZ	0
SalesHistory	1	Date	4	6	7 Date	0
SalesHistory	2	Quantity	4	10	3 Long Integer	0
SalesHistory	3	ItemID	12	14	0 Char / ASCIIZ	0
SalesHistory	4	SalesRegion	6	26	0 Char / ASCIIZ	0
SalesHistory	5	PurchaseAmount	8	32	4 Floating Point	0

Index 0	Client	ClientID: 6, Date: 4, ItemID: 12
Index 1	Item	ItemID: 12, Date: 4, ClientID: 6
Index 2	Region	SalesRegion: 6, Date:4, ClientID: 6, ItemID: 12
Index 3	Date	Date: 4, ClientID: 6, ItemID: 12

Example System

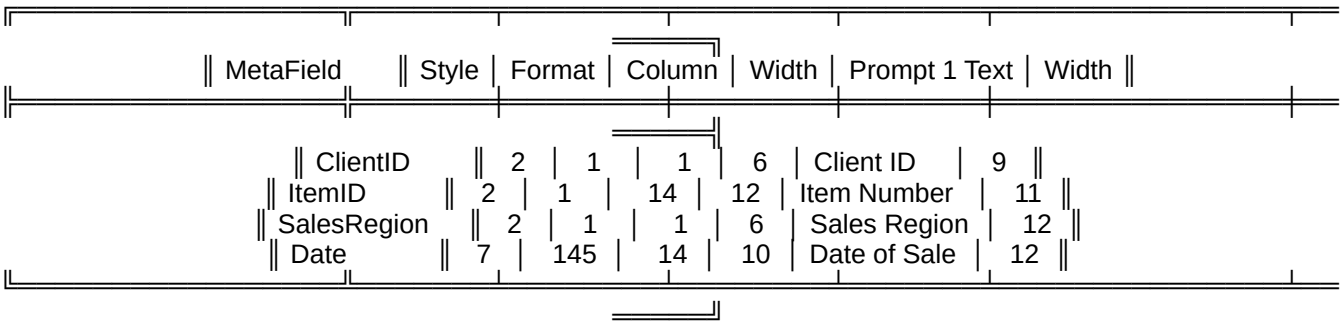
Step 5: Repeat 1-4 until done

Sales History Browser Structure

Name	SalesHistoryBrow
Database	SalesHistory
Usage	0
Display	SalesHistoryBrow
Help	Entry Mode
Title	Sales History
Footer	Entry Mode: Sales History Records
Footer Offset	0
Footer Colors	"HelpPlain"
DEScreen	SalesHistoryEntry
Row	4
Column	21
Height	15
Width	38
True Height	15
True Width	38
Item Height	1
Colors	DEScreen
Function Keys	DefaultDEScreen
BHelp	Browsers
BColors	Browsers
Browser Keys	DefaultBrowser
BFooter	Sales History Records
Search	

Sales History Browser Field Records

Remember: all browser fields will be in the SalesHistoryBrow group, with Field Row 1, and Prompt1 Row 0.



MetaField	Style	Format	Column	Width	Prompt 1 Text	Width
ClientID	2	1	1	6	Client ID	9
ItemID	2	1	14	12	Item Number	11
SalesRegion	2	1	1	6	Sales Region	12
Date	7	145	14	10	Date of Sale	12

Example System

Step 5: Repeat 1-4 until done

Sales History Menu Line

Group		ExampleMenu
MetaField		Browser("SalesHistoryBrow")
Style	0	
Format		0
Decim		0
Width2		0
Row		3
Column		0
Height		0
Width		0
Prompt1		" Sales History "
Row		3
Column		0
Height		1
Width		15
Prompt2		I
Row		3
Column		1
Height		1
Width		1
Color Set		1
Key Name		
Help		
Lookup		
Footer		Load the Sales History Browser

Example System

Step 5: Repeat 1-4 until done

Sales History Data Entry Structure

Name	SalesHistoryEntry
Database	SalesHistory
Usage	1
Display	SalesHistoryEntry
Help	Data Entry
Title	Sales History Line Items
Footer	Sales History Data Entry Screen
Footer Offset	2
Footer Colors	HelpPlain
DEScreen	
Row	7
Column	1
Height	6
Width	78
True Height	6
True Width	78
Item Height	0
Colors	DEScreen
Function Keys	DefaultDEScreen
All Others	Blank or 0

Sales History Data Entry Field Records

Group	SalesHistoryEntry	SalesHistoryEntry	SalesHistoryEntry
MetaField	ClientID	Date	SalesRegion
Style	2 Character String	7 Date	2 Character String
Format	1 Editable	145	1 Editable
F. Bits	Skip this field	Skip this field	Skip this field
Decim	0	0	0
Width2	0	0	0
Row	0	1	1
Column	14	14	66
Height	1	1	1
Width	5	10	3
Prompt1	Client ID :	Sale Date :	Region :
Row	0	1	1
Column	2	2	57
Height	1	1	1
Width	11	11	8
All others	Blank / 0	Blank / 0	Blank / 0

Example System

Step 5: Repeat 1-4 until done

Sales History Data Entry Field Records

Group	SalesHistoryEntry	SalesHistoryEntry	SalesHistoryEntry
MetaField	ItemID	Quantity	PurchaseAmount
Style	2 Character String	4 Standard Numeric	4 Standard Numeric
Format	1 Editable	1 Editable	Use Format Bits
F. Bits	Skip this field	Skip this field	Editable, Decimals
			Dollar Sign : $
Decim	0	0	2
Width2	0	0	0
Row	3	5	5
Column	14	14	66
Height	1	1	1
Width	11	6	10
Prompt1	Item ID :	Quantity :	Purch Amount :
Row	3	5	5
Column	4	3	51
Height	1	1	1
Width	9	10	14
All others	Blank / 0	Blank / 0	Blank / 0

Safety Precaution: Before advancing to step 6, make sure that <u>all</u> systems entered pass the following test criteria:

- Are all of the menu lines visible? Are the 'hot key' selections visible
- Will the menu cursor bar move properly up and down through the menu
- Will the menu cursor bar skip over the separation line
- Do the 'hot keys' work
- Do the browsers come up without complaint
- Do the data entry screens come up without complaint
- Are the browser and data entry screen colors consistant
- Are the fields and prompts on the browsers and data entry screens at least one character away from the left and right borders
- Can sample records be entered into each database: Add two or three records to each database, watching the browser for proper displays of the entered data.
- Will records copy and delete properly from the browsers

Once all database systems have passed these tests, delete all sample entries, and pack all of the databases. **<F4>**+**<Y>** deletes the record the browser bar is on, **<F10>**+**<Y>** packs the databases.

Example System

Step 6: Relations, Nested Fields, Lookups

Steps 1 through 5 have generated a series of databases that can be used as 'file card' boxes. The databases designed are independant, unrelated, without external references. The purpose of this step is to 'connect the loose ends' between the databases.

The difference between 'a series of databases' and a database system should be clarified in this step: a database system relates, correlates, and explains information in terms of one or several databases. A database system, then, is a harmonious organization of information: making entry simpler to perform, and more comprehensible to read.

Relationships versus References

There are two major methods used to connect databases: a dependancy relationship, and a definition reference. In general, a relationship provides connection between database records of two databases with some mandatory correspondance: loss of records in one database either changes or eliminates the meaning of the corresponding records in the other database. A reference, or what some database systems refer to as a 'dictionary', or a 'table', provides specific meanings for one field or group of fields within a database record.

A Relationship is a direct connection between two databases: records in one database are related to records in another database. These relationships are enforced by an entry in the Search Structures database, and are defined by appropriately matching values in certain fields between the records in two databases. In this example, the ClientID field in the Sales History databases will be used to relate Sales History records to Client Info records.

A reference is an implied connection between two databases: the contents of a field in one database are given meaning by another database. References are often used to provide explanations of a field's contents, shown with lookups on a data-entry screen, and accessed through PickLists or BrowseLists. Some users may consider this an expanded definition of a data dictionary: VariForm data dictionaries are expressions of part or all of a database, using a Search Structure to define the parameters and scope of the dictionary.
In this example, there are several references. The Sales History database has two references: Item Number, which pulls in a description and price for an item, and Sales Region, which pulls in a description of the region where the sale occurred. The Client Info database also has a Sales Region reference field.

One other reference is available: the Sales History database can use the ClientID field, which defines its relationship to the Client Information database, to show information about the Client making the purchase. While this is technically a reverse-relationship, it will be used and examined as another use of a reference.

Example System

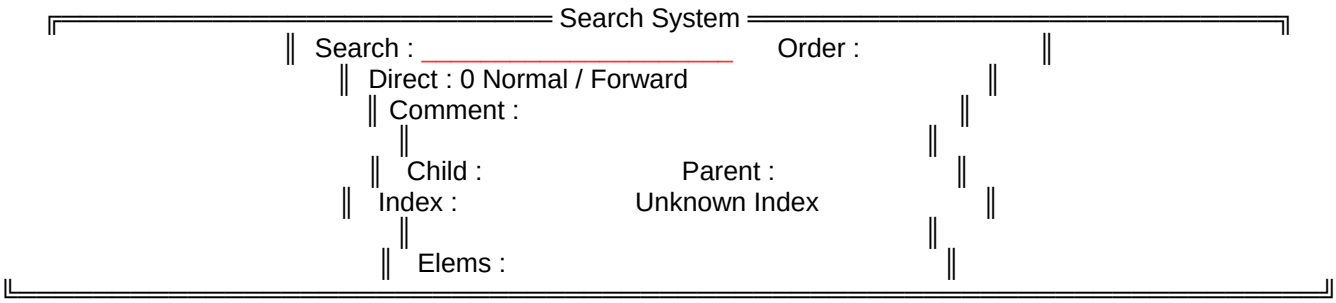
Step 6: Relations, Nested Fields, Lookups

Establishing a Relationship

Statement of relationship: All sales in the Sales History database will be made by clients in the Client Info database. There will be a one-to-many relationship, with the parent database, Client Info having a unique field (ClientID), that attaches a client record to all sales made by the client. All entries in the Sales History database, will have the appropriate client value in a corresponding ClientID field.

Implementation: A Search Structure record will be added. This record will be loaded automatically whenever the Client Info (parent) database is opened, and will establish the relationship to Sales History. The Sales History database will be automatically opened as a side-effect. Deleting a Client Info record will automatically cause all associated Sales History records to be deleted. Changing the ClientID field of a Client Info record will automatically change the ClientID field of all corresponding Sales History records.

To enter the relationship: Go to the VariForm System submenu, and select the Search Structures choice. This will bring up the Search Structures browser. Press **<F3>** to add a Search Structure record.



Search is the unique 'group name' that identifies the Search Structure and its associated Search Field records. Enter **ExampleRelation**.

Order is used for extremely complex search relationships. Leave this field as **0**.

Direct controls the direction of the search being performed: forward, with respect to the index, or reverse, against the direction of the index. Leave this field as **0**.

Comment provides space for a comment or description of what this search does. Enter **Relationship between ClientInfo and SalesHistory**.

Child contains the name of the child database in a relationship, or the subject of a search. Press **<F2>** for a PickList of databases, and select the **SalesHistory** database.

The **Parent** field is only used for relationship definitions, and contains the name of the parent database in a relationship. Press **<F2>** for a PickList of databases, and select the **ClientInfo** database.

The **Index** contains the name of the index within the child database to use for searching. Press **<F2>** for a PickList of Indexes, and select the **ClientID** index. This index makes the

most sense, since it is the ClientID field, which is sorted by this index, which will be used in the relationship.

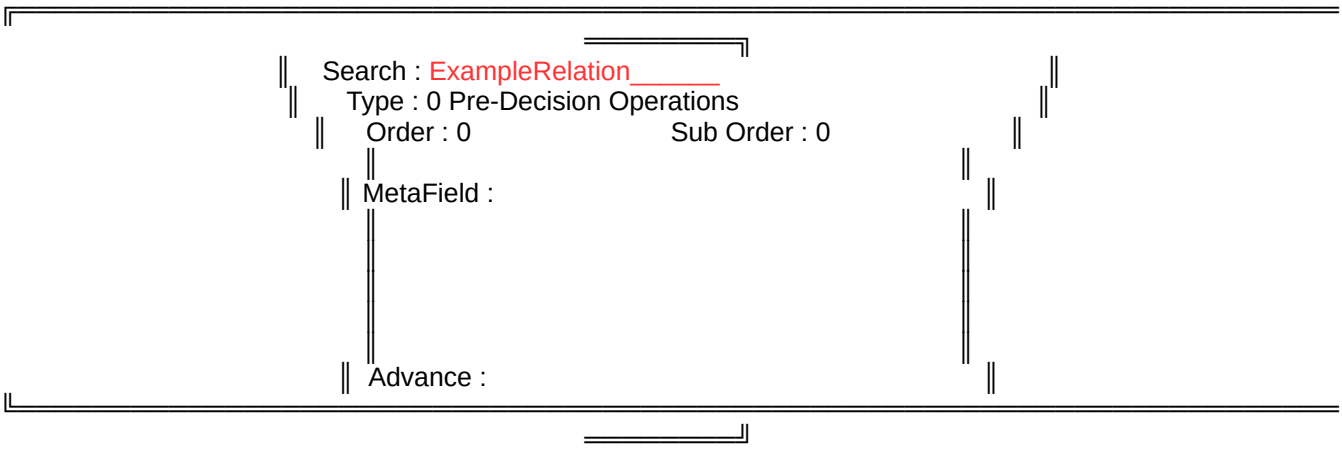
Elems is a nested field that provides access to the Search Field records database. Press **<Enter>** on this field, and the Search Field browser will appear.

Example System

Step 6: Relations, Nested Fields, Lookups

A Search structure needs 2 Search Fields to maintain a database relationships: a type 1 (Decision Logic) field that establishes what the relationship is, and a type 4 (Enforcement Action) field that tells the search how to create relationships for new child records.

Press **<F3>** on the Search Field browser.



Search is the 'group' field for the Search Field records, and will already be filled in correctly. **<Enter>** past this field.

Type determines when and how this Search Field record is used within the search. For more information on Search Field Types, see the table on page . Enter a **1** for this record.

Order, along with **Sub Order**, keep all of the Search Field records organized within a particular Search's Search Field Type. Enter **0** for both of these fields.

MetaField contains the formula, equation, or actions to perform at this point. Since the relationship will be established by the matching of ClientID fields, enter:

>ClientID == <ClientID.

The 'angle brackets' next to the field names tell the Search System whether to pull the designated field from the parent or child database, or whether to look in the virtual field area. A right angle bracket (alias a greater than sign) tells the search system to check the child database; a left angle bracket (a less than sign) checks the parent database; no bracket (just a field name) checks the 'virtual database', or region of virtual fields.

Advance is only used on extremely long type 1 records, where the search decision requires more than one search field records to establish the equation. Advance contains logical operators, such as and, or, xor, et cetera. **<Enter>** past this field.

Press **<Ctrl-Enter>** to save this record, and add the following type 4 record:

Search **<Enter>** past this field
Type **4** (Enforcement Actions)
Order 0
Sub Order **0**
MetaField **>ClientID = <ClientID**
Advance **<Enter>** past this field

This enforcement action will bring the ClientID of the Parent record into each Child record, when adding child records. This process is actually being shown by the Search Field records 'Search' field being filled in properly when adding child records to the ExampleRelation Search Structure.

Example System

Step 6: Relations, Nested Fields, Lookups

To make the search efficient, there are two more record types that will help: Type 5 (Start Conditions), and Type 6 (Stop Conditions). These record types allow the search process to begin close to the first possible record, and to stop as soon as the last possible record has been found. The effects of type 5 and 6 records are not often obvious until the child database has several dozen entries.

Search **<Enter>** past this field
Type 5 (Start Conditions)
Order 0
Sub Order 0
MetaField >ClientID = <ClientID
Advance **<Enter>** past this field

This record forces the search to start at the first Sales History record where the ChildID field is correct. While it seems redundant, it does save time as the child database grows.

Search **<Enter>** past this field
Type 6 (Stop Conditions)
Order 0
Sub Order 0
MetaField >ClientID > <ClientID
Advance **<Enter>** past this field

This record tells the search to stop when it reaches a record that has a ClientID field that is greater than the searched-for value. Since the ClientID index will keep the ClientID fields of the records in ascending (normal) order, this will stop the search after the first impossible record has been found.

Example System

Step 6: Relations, Nested Fields, Lookups

Implementing a Relationship

To make use of the newly established relationship, add a limited browser to the child database, and access that browser through a nested field on the parent database's data entry screen.

Instead of creating and sizing a whole new browser, this implementation will simply re-use the already-defined SalesHistoryBrow browser. To do this, go to the Display Structures browser, and locate the SalesHistoryBrow entry.

- With the browser bar on the SalesHistoryBrow record, press **<F5>** - the copy key.
  A new record, named SalesHistoryBrowA will appear below the browser bar.

- Move down to the SalesHistoryBrowA record, and press **<Enter>** to edit it.

- Press **<Ctrl-Page Down>** to get to the Search Field, and press **<F2>**, to generate a list of Search Structures. Select the **ExampleRelation** structure that was entered previously.

- Press **<Ctrl-Enter>** to save the new, limited browser.

Accessing the limited browser will require a nested field on the ClientInfo data entry screen. This new field would fit nicely if the ClientInfoEntry screen and window were 1 row larger:

- Locate the ClientInfoEntry Display Structure, and press **<Enter>** to edit it.

- Change the Height and TrueHeight fields to 13

- Press **<Ctrl-Enter>** to save the new, slightly larger data entry screen.

To add the new field, go to the Display Fields Browser, and add the following record:

Group	ClientInfoEntry
MetaField	
Style	8 Nested Field
Format	0 Uneditable
Decim	4
Width2	0
Row	12
Column	19
Height	1
Width	14
Prompt1	Sales History :
Row	12
Column	3
Height	1
Width	15
Lookup	SalesHistoryBrowA
All others	Blank / 0

The lookup tells the Data Entry system what browser to load. SalesHistoryBrowA tells the Browser system what limiting search structure to use. ExampleRelation, in turn loads the ClientID field of new child record appropriately.

Example System

Step 6: Relations, Nested Fields, Lookups

Test the Relationship

Two sets of sample data are provided to test the relationship just established. Entering this data will show if the relationship exists, and if it is implemented properly.

Client ID: SmJ-1
Name: Mrs. Joan Smith
4321 Woodhill Drive
(leave second address line blank)
Anytown, Pa. 16543
First Sale: Jan 1, 1993
Total Purchases: (leave blank)

Press **<Enter>** on the Sales History nested field, and a blank SalesHistory browser should appear. Press **<F3>** on the SalesHistory Browser, and a blank Sales History Entry record should appear, with the ClientID already filled in as SmJ-1. Enter these 3 sales:

Date: Jan 1, 1993
Item: PUR-LEA-3
Quantity: 2

Date: Jan 15, 1993
Item: KCH-LEA-4
Quantity: 2

Date: Feb 21, 1993
Item: WAL-LEA-1
Quantity: 3

The second client, Mr. John Smith, will be tracked separately. He has the same address as Mrs. Smith, so use the **<F5>** (copy) key to make a copy of Mrs. Smith's record. Edit the client record to read:

Client ID: SmJ-2
First Sale: February 5, 1993

Press **<Ctrl-Enter>** to save the changes, then **<Enter>** on this record to re-edit it. Go down to the Sales History field, and press **<Enter>**. Note that a copy of Mrs. Smith's sales were made for Mr. Smith's record. This is because by default, relationships copy child records when the parent record is copied.

On Mr. Smith's Sales History browser, delete the three sales records with the **<F4>** (delete) key. Then add the following two sales:

Date: February 5, 1993
Item: JCK-LEA-1
Quantity: 2

Example System

Step 6: Relations, Nested Fields, Lookups

Date: March 10, 1993
Item: CAP-LEA-1
Quantity: 9

From the Example Menu, the regular (unlimited) Sales history browser should show 5 active records, and 3 deleted records (**<F9>** Info will initially show 8 records).

Example System

Step 6: Relations, Nested Fields, Lookups

Reference Displays: Lookups

In the previous section, sales were entered on some item numbers. What are those items? What follows is the appropriate data entry for the Item Info database:

Item: PUR-LEA-3
Desc: 3-Sided Leather Purse
Price: $149.50

Item: KCH-LEA-4
Desc: Standard Keychain with Square Leather Fob
Price: $9.95

Item: WAL-LEA-1
Desc: Handmade Leather Wallet
Price: $39.95

Item: JCK-LEA-1
Desc: Black Gloss Leather Biker's Jacket
Price: $249.30

Item: CAP-LEA-1
Desc: Black Gloss Leather Sports Cap
Price: $79.75

This seems to be very useful information: it could make the Sales History database a great deal more meaningful it the item description appeared next to the item number field, for example. This is the function of a lookup.

What is a lookup? A lookup is an extra display option available to any Display Field record that describes a field that is neither nested, nor uses decimal places. Lookups can be used on Browsers, Data Entry Screens, PickLists and Reports.

A lookup is actually a special use of a Search Structure. The content of the field being 'looked up' is passed to the Search Structure in a field called #Target#. The Search performs its work, and alters the value of #Target#. The lookup process then displays the content of #Target# to the right of the original field.

Types of Lookups

Simple (1-field) lookups are reusable: the same lookup can be used in a variety of data entry screens, or many times on the same data entry screen. This is because of the #Target# field: If the Search System had to use the specific field name that was being looked up, the overhead would be enormous. Example: the Color Structures data entry screen looks up the meaning of 15 different color values, through 1 lookup. This means only one search structure has to be loaded for all fifteen fields. The savings in space and time can be enormous.

Compound lookups: the result is determined by more than one field's value. Searches can use any and all specific fields from a database to perform a single lookup, by using the 'Parent Indicator' (<) before the field name within the Search Fields' structures. Example: A data entry screen may display a suggested price based on both the Item and the Quantity being purchased.

Complex lookups: the result is determined by more than one database. For example, in a 3-database relationship, a bottom-layer record may be more meaningful if it displayed some information from the top-layer record: this would require the lookup to reverse 2 relationships, which means performing 2 Searches to get the appropriate #Target# value for display.

Compound-Complex lookups: the result is determined by several fields used to search more than one database. For example, 2 fields might describe an unrelated 2-layer database, with one field accessing a parent record, and one field accessing a particular sub-record, if it exists. While this sounds rather esoteric, it is surprising common among multiple-catalog vendors' point-of-sale pricing systems, as well as certain accounting statistics, such as uneven depreciation on a variable value asset.

This manual provides an introduction to VariForm, and so the example only shows simple lookups. For further assistance in more sophisticated lookups, get our companion manual, *Advanced VariForm Programming Styles and Structures: The philosophy of compiled data*, which provides several multilayer databases that demonstrate all forms of lookups and relationships.

Example System

Step 6: Relations, Nested Fields, Lookups

Implementing a Lookup

There are two steps to implementing a lookup: Enter a search structure to do the lookup, and call the lookup from a Display Field record. The search structure has 2 mandatory elements: type 1 (Search Condition) and type 2 (Positive Action), and 1 recommended element: type 3 (Negative Action).

Search Structure Record

Search **ExampleLookup**
Order **0**
Direct **0**
Comment **Lookup for Description of Item Number**
Child **ItemInfo**
Parent (Leave blank: lookups don't care)
Index **Abbrev**

Search Field Records

Search **<Enter>** past this field
Type **1** (Decision)
Order **0**
Sub Order **0**
MetaField **>ItemNumber == #Target#**
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type **2** (Positive Action)
Order **0**
Sub Order **0**
MetaField **#Target# = >ItemDescription**
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type **3** (Negative Action)
Order **0**
Sub Order **0**
MetaField **#Target# = "No Such Item Number"**
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type **5** (Start Condition)
Order **0**
Sub Order **0**
MetaField **>ItemNumber = #Target#**
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type **6** (Stop Condition)
Order **0**

Sub Order **0**
MetaField **>ItemNumber > #Target#**
Advance **<Enter>** past this field

Example System

Step 6: Relations, Nested Fields, Lookups

To add the lookup to the SalesHistoryEntry data entry screen, go to the Display Fields browser, and locate the "ItemID" field within the SalesHistoryEntry group.

- Press **<Enter>** on the browser to edit this record

- Go down to the Lookup Field, and enter **ExampleLookup**

- If, for some reason, the length of the lookup needs to be limited, such as an additional field to the right of the field being looked up, put the desired length of the lookup, in characters, in the decimals field of this display field record.

- Press **<Ctrl-Enter>** to save the change made.

Test the Lookup

Go to the SalesHistory database in either direction: direct from the Example Menu, or through the Client Address database's Nested field. In either event, if the Search Structure and the Display Field were entered and edited correctly, there should be a description string to the right of the Item Number field.

Example System

Step 6: Relations, Nested Fields, Lookups

Other recommended Lookups

Two other lookups, used appropriately, will add to the comprehensibility of the Client Info and Sales History data entry screens: A lookup to clarify the Sales Region field in both screens, and one for the ClientID field in the Sales History screen.

Search Structure Record

Search **ExampleLookup2**
Order 0
Direct 0
Comment **Lookup for Description of Sales Region**
Child SalesRegion
Parent (Leave blank: lookups don't care)
Index Abbrev

Search Field Records

Search **<Enter>** past this field
Type 1 (Decision)
Order 0
Sub Order 0
MetaField >RegionID == #Target#
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type 2 (Positive Action)
Order 0
Sub Order 0
MetaField #Target# = >RegionName
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type 3 (Negative Action)
Order 0
Sub Order 0
MetaField #Target# = "No such Sales Region"
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type 5 (Start Condition)
Order 0
Sub Order 0
MetaField >RegionID = #Target#
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type 6 (Stop Condition)
Order 0

Sub Order **0**
MetaField **>RegionID > #Target#**
Advance **<Enter>** past this field

Add this lookup to the SalesRegion display fields of ClientInfoEntry and SalesHistoryEntry.

Example System

Step 6: Relations, Nested Fields, Lookups

Search Structure Record

Search **ExampleLookup3**
Order **0**
Direct **0**
Comment **Lookup for Description of Client ID**
Child **ClientInfo**
Parent (Leave blank: lookups don't care)
Index **ClientID**

Search Field Records

Search **<Enter>** past this field
Type **1** (Decision)
Order **0**
Sub Order **0**
MetaField **>ClientID == #Target#**
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type **2** (Positive Action)
Order **0**
Sub Order **0**
MetaField **#Target# = (>Title + " " + >FirstName + " " + >LastName)**
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type **3** (Negative Action)
Order **0**
Sub Order **0**
MetaField **#Target# = "No Such Client ID"**
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type **5** (Start Condition)
Order **0**
Sub Order **0**
MetaField **>ClientID = #Target#**
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type **6** (Stop Condition)
Order **0**
Sub Order **0**
MetaField **>ClientID > #Target#**
Advance **<Enter>** past this field

Add this lookup to the ClientID display field of SalesHistoryEntry.

Example System

Step 6: Relations, Nested Fields, Lookups

PickLists and BrowseLists

Lookups expand the meaning of an abbreviation, but that doesn't insert the correct values into the end-user's head. PickLists and BrowseLists provide the user with a list of appropriate meanings to select from. When the user selects the appropriate meaning, the correct value is inserted into the field.

There are two differences between a PickList and a BrowseList: PickLists can not change the lookup database, so **<Enter>** makes the selection; Browselists are browsers, and can change the database being looked up, so a **<Select()>** key makes the selection. If the browselist uses the DefaultBrowseList Function Key set, Select is both **<F7>** and **<F8>**.

PickLists and BrowseLists are called by Function-Keys within data entry screens, so entering these list types will also demonstrate how function keys are attached to fields.

It is important to remember that PickLists and BrowseLists should use very similar search logic to that of any associated lookup. The difference is that a list routine will not test for an exact field match: On a simple lookup, the list will show all records in a database; on a compound lookup, the list will show all records that qualify, based on the values of the other fields used within the search.

A simple and useful picklist/browselist combination would speed data entry in the SalesHistory Database, for example: selecting an item number from a list of item descriptions through the picklist, and adding items to the item database through the browselist when necessary.

The lists will require a search to facilitate item selection: when the User makes a selection, what field should VariForm return from the target database's selected record. The following Search Structure will answer that:

Search **ExampleList**
Order **0**
Direct **0**
Comment **List Search for Item Description**
Child **ItemInfo**
Parent (Leave blank: lookups don't care)
Index **Description**

Search Field Records

Search **<Enter>** past this field
Type **1** (Decision)
Order **0**
Sub Order **0**
MetaField **1**
Advance **<Enter>** past this field

Search **<Enter>** past this field

Type 10 (Loop #1 Positive Action)
Order 0
Sub Order 0
MetaField #Target# = >ItemNumber
Advance <Enter> past this field

Example System

Step 6: Relations, Nested Fields, Lookups

The purpose behind using the Description Index should be obvious: if the user knew the item number, they would not ask for a list of items. Searching by description, which is what the lookup will later display, makes more sense.

In this example, the PickList and BrowseList will actually use the same Display Structure, and share the Display Field group with the ItemInfoBrow browser. It is more common, but beyond the scope of this guide, to provide separate PickList and BrowseList Display Structures: this allows for separate help lines, different color schemes, et cetera.

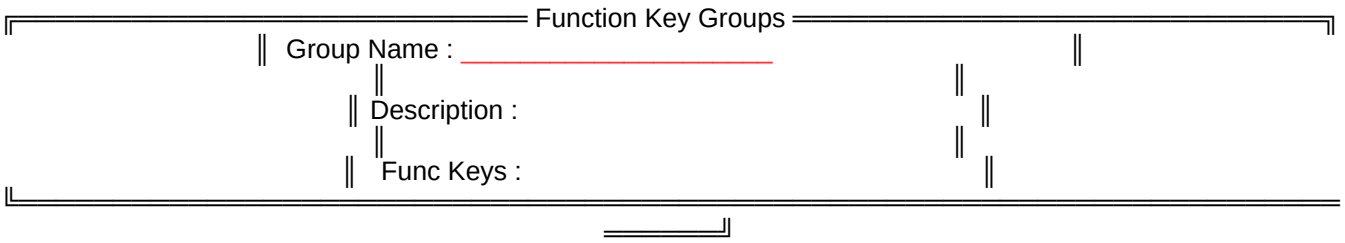
To make the extra record, go to the **Display Structures** database and make a copy of the ItemInfoBrow display structure. Edit that copy, with the following changes:

Name	**ItemInfoBList**
Usage	Leave at **0**
Browser F. Keys	**DefaultBrowserList**
Browser Colors	**BrowserList**
Search	**ExampleList**

<Ctrl-Enter> to save the changes. A BrowseList / PickList structure is now available. The next step is to define function keys to call the structure, and to attach those keys to the appropriate data entry screen's display field.

Creating a Function Key Group

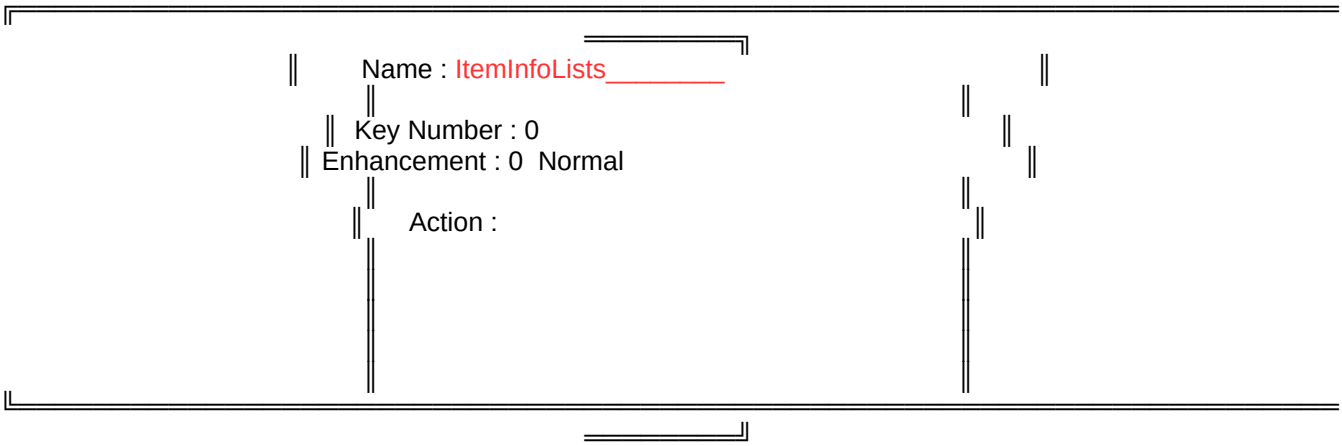
Select the **Function Key Group** choice on the VariForm System submenu. Press **<F3>** to create a blank record for editing.



Group Name contains the unique identifier for this group of function keys. Enter **ItemInfoLists**.

Description contains a text description of what this key-group does. This is for the function key picklists. Enter **F2-PickList, F4-BrowseList of ItemInfo**.

Func Keys is a nested field that brings up the actual function key entries for this group. Press **<Enter>** on this field to bring up the Function Key Browser.



Name contains the (relationship enforced) group name for this Function Key Group. **<Enter>** past this field.

Key Number is the function number for this key. E.g.: <F2> would be a value 2. Enter a **2**.

Enhancement is the extended function id for this key, so <Shift-F1> would be key 1, enhancement 1 (shift). **<Enter>** past this field.

Action tells VariForm what to actually do when this function key is pressed. For this record, enter **PickList("ItemInfoBList")**.

<Ctrl-Enter> to save this record. Now, whatever field uses the ItemInfoLists function keys will have a <F2> key that generates an ItemInfo PickList.

The second record for this Function Key Group is <F4> - BrowseList. The record should contain:

Name	**ItemInfoLists**
Key Number	**4**
Enhancement	**0**
Action	**BrowseList("ItemInfoBList")**

To add this function key grouping to the SalesHistory data entry screen, go to the Display Field records, and edit the ItemNumber field of the SalesHistoryEntry field group.

Go down to the "Function Keys" field, and press **<F2>**, for a PickList of available function key groups. Select the Item Info key grouping just entered. Then go to the cue line field, and enter **F2 - PickList F4 - BrowseList**. That text will appear in the cue lines at the bottom of the computer screen whenever this field is current. When adding function keys, it is best to follow a standard: users find consistant interfaces easier to learn and remember. VariForm System function key conventions are listed in the Tables appendix, on page .

Example System

Step 7: Procedures

The problem with straight database systems is that systemized information is static. The real world's information flow is dynamic: records will be added to Sales History, and the Total Purchases field for the Client Info database must be maintained. After a certain time, inactive clients should be deleted, along with their sales histories, to make space for newer, active clientele.

Just as relationships, lookups, and lists systemize information, procedures can consolidate, analyze, and modify information to make it more useful. Procedures tell VariForm how to perform calculations, modify large clusters of information, and have the computer do the grunt work with very little effort on the part of the end-user.

Technically, every operation is some expression of a procedure: a menu appears because the command line processor was sent a Menu() command; a browser comes up because a user selected a Menu Line or Function Key that sent a Browser() command to the command line processor. These types of operations can be enhanced greatly through the creation of customized procedure commands.

The VariForm procedure system is a hybrid between a 'macro' language and a full-fledged programming language. The CommandLine System can duplicate any set of user actions it is programmed to do, including calling Dos Shells, and running other programs within that shell. While working with this step, reviewing the section on the CommandLine Interface that starts on page may be useful.

There will be three procedure examples in this section: a data entry screen procedure to calculate the total of a sale line item, a browser procedure to calculate the total purchases for a client, and a menu procedure to eliminate 'old' clients.

Example System

Step 7: Procedures

Procedure 1: line item calculations

There is a field in the Sales History database that can be calculated by VariForm : Purchase Amount. Mathematically, this would be expressed as:

PurchaseAmount = Quantity * (Price per item)

Price per item is in parentheses because VariForm does not have this information available on the Sales History database. This calculation will require a quick search to get the price of the item from the Item Info database, so the steps for this procedure will be:

- determine what item is in this record
- get the item price
- do the math
- redraw the screen, to show the new price

In VariForm's procedure language, this is expressed as:

Procedure: DoPriceCalc

```
#Target# = ItemID
#FloatAnswer# = 0
DoSearch( "GetItemPrice" )
PurchaseAmount = (Quantity * #FloatAnswer#)
Redraw()
```

This assumes that there is a search named "GetItemPrice", which will find the Item record in the ItemInfo database with the correct ID#, and will return the price in the #FloatAnswer# field. Since this search is very similar to the "ExampleLookup" search, which returns Item Description based on Item ID, it makes sense to expand the functionality of the search, instead of creating a whole separate one. Go to the Search Structures browser, and bring up the ExampleLookup Search Record. Press **<Enter>** on the nested field to bring up the Search Field records, and press **<F3>** to add the following records to the search:

Search **<Enter>** past this field
Type **2** (Positive Action)
Order **1**
Sub Order **0**
MetaField **#FloatAnswer# = >ItemPrice**
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type **3** (Negative Action)
Order **1**
Sub Order **0**
MetaField **#FloatAnswer# = 0**
Advance **<Enter>** past this field

Example System

Step 7: Procedures

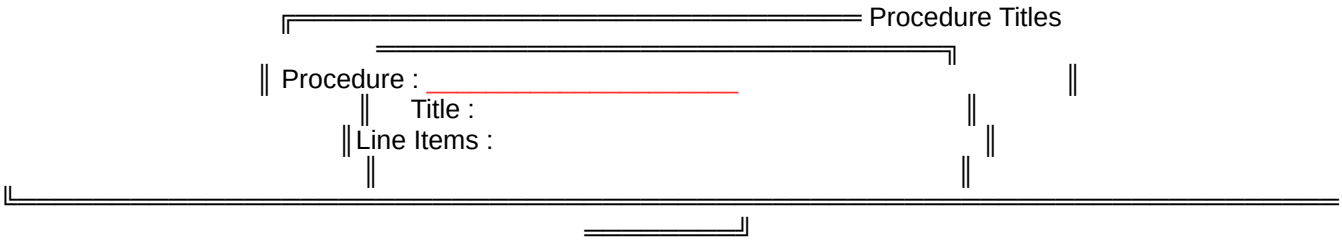
With those two records added, the ExampleLookup now performs two functions during a lookup: get an item description, as well as the item's unit price. The next step is to add the procedure to VariForm.

Example System

Step 7: Procedures

Adding a Procedure

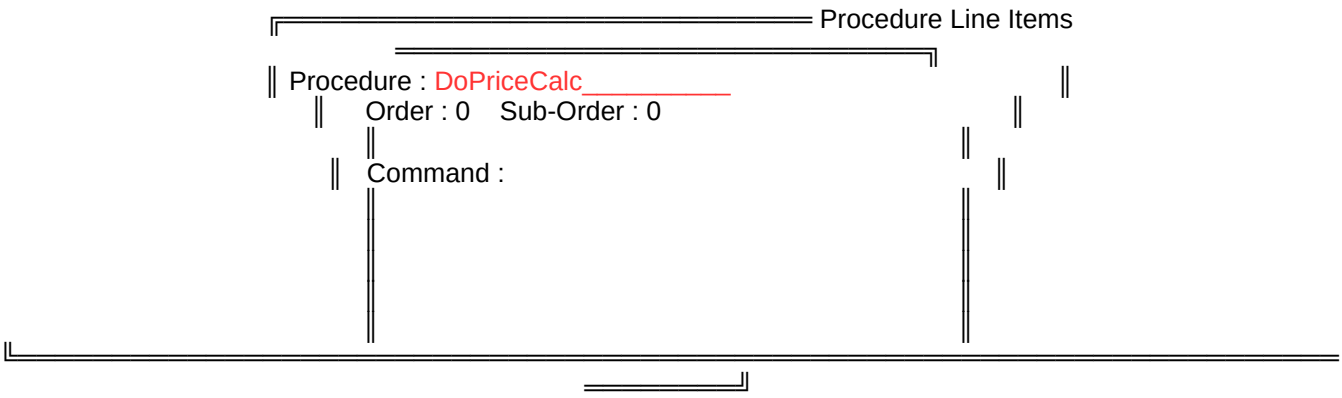
Go to the VariForm System submenu and select **Procedures**. This will bring up the Procedure Titles browser. Press **<F3>** to create a blank record for editing.



Procedure contains the unique name of the procedure for calling purposes. Enter **DoPriceCalc** here.

Title contains a text description of what this procedure will do when called. Enter **Load Price from Item Info database & multiply out**.

Line Items is a nested field that accesses the procedure's line item records. Press **<Enter>** to bring up the Procedure Line Items browser. Press **<F3>** to generate a blank record for editing.



Procedure is the name of the procedure to associate this line with. It had already been filled in by the Nested Field relationship. **<Enter>** past this field.

Order and **Sub-Order** are the fields that keep the lines in sequence. The first record should be **Order 1, Sub-Order 0**.

Command is the active MetaField that contains the operation to be performed. From the previous page, it should be clear that the first line is: **#Target# = ItemID**. Since this is a procedure that will be called from a data entry screen, the field name (ItemID) needs no parent/child designation.

Press **<Ctrl-Enter>** to save this record.

Example System

Step 7: Procedures

A Procedure Entry Short-Cut

There are four more lines to add, as per page . These records can be entered manually, but that requires manually entering the line numbers. VariForm provides an option for the computer to maintain the numbering order. This is described as incremental copying. In certain situations, such as numeric fields on a limited browser, the copy command will increment the last numeric field on the index, which in this case is Sub-Order.

From the Procedure Line Items browser, press **<F5>** (copy) to make a copy of the record just entered. The new record will be Order 1, SubOrder 1. Move to the new record and make another copy. The new record will be Order 1, SubOrder 2. Continue this until there are four copies of the first record. The browser should look as follows:

Procedure Line Items

Procedure	Order	Sub	Command Line
DoPriceCalc	1	0	#Target# = ItemID
DoPriceCalc	1	1	#Target# = ItemID
DoPriceCalc	1	2	#Target# = ItemID
DoPriceCalc	1	3	#Target# = ItemID
DoPriceCalc	1	4	#Target# = ItemID

Press **<Ctrl-F8>** (Renumber by 1's). After a few seconds, the browser will redraw will all of the records numbered in order, with sub order being 0.

Now press **<F8>** (Entry Mode). The browser will change colors, the first field of the browser bar will turn into a field prompt (like those on a data entry screen), and the 'Procedure' column title will become the PromptSelect color (default = white).

Procedure Line Items

Procedure	**Order**	**Sub**	**Command Line**
DoPriceCalc	1	0	#Target# = ItemID
DoPriceCalc	2	0	#Target# = ItemID
DoPriceCalc	3	0	#Target# = ItemID
DoPriceCalc	4	0	#Target# = ItemID
DoPriceCalc	**5**	**0**	**#Target# = ItemID**

Use **<Up Arrow>** to move the Entry Mode's Browser Bar up to the first copy: Order 2, Sub 0. Type right over the record, using the **<Tab>** or **<Enter>** key to advance across the field columns. Change the record's Command Line to **#FloatAnswer# = 0**, and press **<Down Arrow>** to save the record. Edit the lines as follows:

Ord	Sub.	Command Line
3	**0**	**DoSearch("ExampleLookup")**
4	**0**	**PurchaseAmount = (Quantity * #FloatAnswer#)**
5	**0**	**Redraw()**

After making the changes to the last line, press **<Ctrl-Enter>** to save the last record and return to the Standard Browse Mode.

This process can be much more convenient for function keys and small procedure entries than constantly bringing up a data entry screen. It is also useful for database field structures, to see the surrounding fields and offsets.

Example System

Step 7: Procedures

Make the procedure 'callable'

To make the procedure available to the user, it must be run from either a menu selection or a function key. Since this function is for the Sales History data entry screen, it makes the most sense to add this as a function key on the Purchase Amount field.

Add a function key group as follows:

Function Key Group

Group **DoPriceCalc**
Description **F3 calls the Price Calculation routine**

Function Keys

Group **<Enter>** past this field
Function **3**
Enhancment **0**
Action **Call("DoPriceCalc")**

Finally, go to the Display Fields database, and edit the Purchase Amount field of the SalesHistoryEntry group, making the following changes:

Function Keys **DoPriceCalc**
Cue Line **F3 - Generate Price from ItemInfo database**

That is all there is to adding procedures to data entry screens:

- Lay out the procedure
  - Determine what searches, if any, are needed, and what they must do.
- Enter or modify the necessary searches
- Enter the procedure
- Create a function key to call the procedure
  - In a field group that is or will be attached to the appropriate field
- Add the function key group to the display field, if it is not already there
- Add a cue line to the same display field, to notify the user of the function key

Adding procedures to browsers, or as menu lines requires only some small changes in the location of the calling function: Browsers take a copy of the default browser's function keys, while menu lines can call procedures directly.

Example System

Step 7: Procedures

Procedure 2: Calculate Total Purchases

The Client Info database has two fields that require dynamic management: Total Purchases and Date of First Purchase. These fields should reflect the running total of all related Sales History records, as well as the date of the first Sales History record for this client.

While this procedure could be readily run from the Client Info data entry screen, this example will show how to attach it to the Client Info browser. When this procedure is run, the current Client Info record (where the browser bar is) will be searched and corrected, then the total sale will appear at a "Press Any Key" prompt at the bottom of the screen.

Procedure: ClientTotal

```
#Target# = ClientID
#FloatAnswer# = 0
#DateAnswer# = (Today)
MultiSearch( "GetClientTotal", 1 )
TotalPurch = #FloatAnswer#
FirstPurch = #DateAnswer#
#Target2# = (ClientID + #FloatAnswer#)
Pause( #Target2# )
EditRecord()
```

The MultiSearch call makes sure that all matching records are processed, no just the first record.

Instead of creating a Search structure from scratch, make a copy of the ExampleRelation search structure. Rename the copy to GetClientTotal, and remove the Parent Database field contents. Remember: Relations are the only Search Structures with Parent Database fields filled in, and 100% duplicate relationships can create severe confusion for VariForm.

Enter the Search Field Browser. Delete the type 4 (Mapping) Search Field record, and change the remaining records' MetaFields from <ClientID to #Target#. Add the following actions:

Search **<Enter>** past this field
Type **10** (Loop #1: Positive Action)
Order **1**
Sub Order **0**
MetaField **#FloatAnswer# = (#FloatAnswer# + >PurchaseAmount)**
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type **10** (Loop #1: Positive Action)
Order **2**
Sub Order **0**
MetaField **#DateAnswer# = IfIs((#DateAnswer# == Today), >Date, #DateAnswer#)**
Advance **<Enter>** past this field

Example System

Step 7: Procedures

Explanation of the Order 2 Type 10 record: Before the Search is called, #DateAnswer# is initialized to (Today). During the Search, the first record found will be the earliest date for this client, and will change #DateAnswer# to a (probably) non-today value. From that point on, the line will evaluate to #DateAnswer# = #DateAnswer.

Example System

Step 7: Procedures

Adding a Procedure to a Browser

Most browsers have a wide array of existing function keys: Add, Delete, Copy, Sort, Pack, and so forth. Instead of entering all of these functions by hand, make a copy of the Default Browser's Function Key Group: DefaultBrowser, and rename the copy to ClientInfoBrowKeys.

The recommended first function key, according to the table on page , is Shift-F7, so add the following Function Key record to the ClientInfoBrowKeys group.

Group **<Enter>** past this field (it should say ClientInfoBrowKeys)
Function **7**
Enhancment **1** (Shift)
Action **Call("DoClientTotal")**

After saving this record, go to the Display Structures database and edit the ClientInfoBrow record:

Browser F. Keys **ClientInfoBrowKeys**
Cue Line **sh-F7 : Run Client Totaling Procedure**

Example System

Step 7: Procedures

Procedure 3: Delete old Client Sales

The final example procedure is another typical problem: removing old 'deadwood' information. Every year, the client wants to eliminate sales from two years prior, as well as all clients who made no purchases the previous year. This is a multi-step process:

- Kill old sales
- Recalculate all client totals
- Kill all clients with 0 Total Purchases
- Pack Client Info and Sales History databases

Also, since the client may vary his fiscal year, the procedure should recommend a destruction date, but should offer the user the ability to change that date, as well as offer the ability to abort the procedure, in case it was started accidentally.

These features incorporate three specialized concepts into this example:

- Virtual Fields will be loaded to provide some 'workspace' for the procedure.

- A Requestor will offer the user the option control over the target dates.

- A Confirmation Request offers the user a last-chance process abort.

While these specific elements have not been covered before, they all have similarities to the work done so far:

- Virtual Fields are entered similarly to database fields, except that positioning is calculated by VariForm.

- A requestor is a specialized data entry screen, whose display fields operate exclusively on virtual fields, instead of database fields.

- A confirmation request is a standard function call that highlights the use of the conditional process call: If(oktodo) doit.

Questions to ask for designing this procedure are:

- What is the first sales history date that will not be killed
- Should the Client Info records' Total Purchases fields be recalculated
- Should the Client Info records with Total Purchases == 0 be deleted
- Should the Sales History database be packed
- Should the Client Info database be packed

- Is the user absolutely certain that this procedure should be run

Example System

Step 7: Procedures

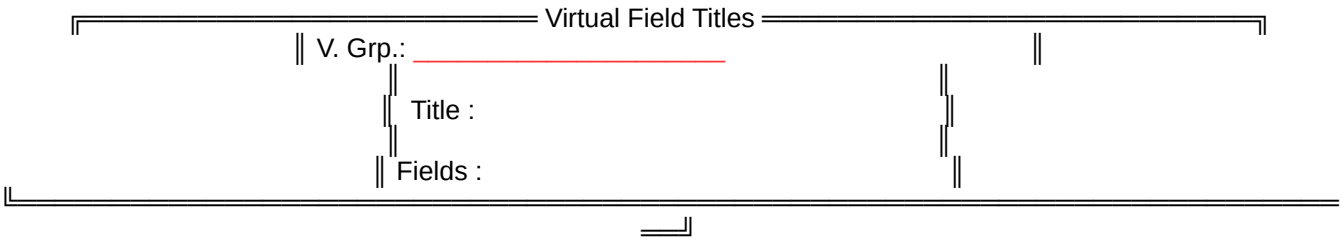
To hold answers to these questions, VariForm will need a few temporary variables. The simplest way to load and pre-assign (initialize) values to more than one variable is to make a group of Virtual Fields, and load them within the procedure.

In this example, the Variables will be:

Variable Name	Field Type	Initial Value	Comment
FirstGoodDate	Date	01\01\1901	Unlikely date
RecalcTotals	Integer	0	'No'
DeleteBlanks	Integer	0	'No'
PackSales	Integer	0	'No'
PackClient	Integer	0	'No'
Confirm	Integer	0	'No'

This group of variables / virtual fields will be called "ClientDeletionVars".

These variables must be entered into the Virtual Field List, so go to the Virtual Field browser, and press **<F3>** to create a blank data entry screen for editing.

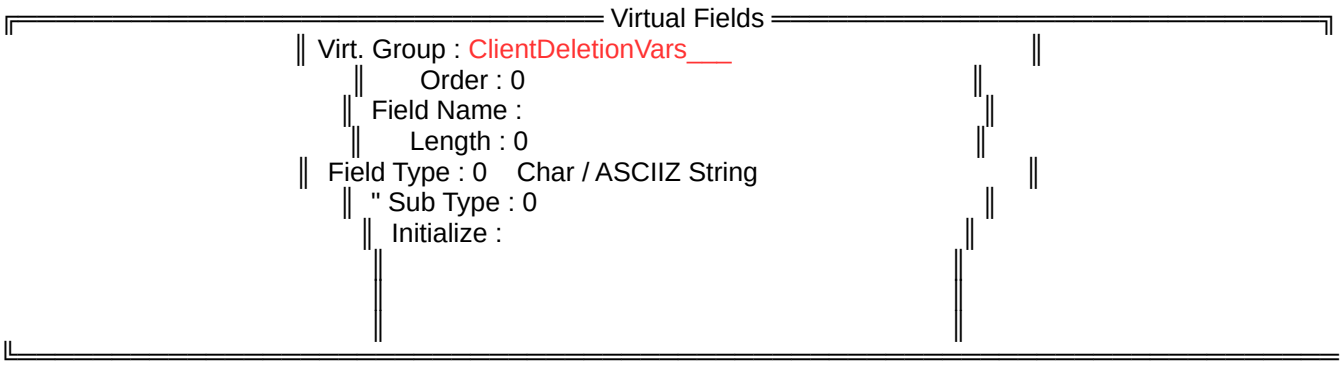


V. Grp. contains the unique name of the Virtual Field Group. For this example, enter **ClientDeletionVars**.

Title contains text that describes what the Virtual Field Group is used for. Enter **Fields for ClientInfo/SalesHistory Purging**.

Fields is a nested field that accesses the individual Virtual Field records. Press **<Enter>** on this field and the Virtual Field record browser will appear.

Press **<F3>** to create a blank record for editing:



Example System

Step 7: Procedures

Virt. Group attaches the Virtual Field records to the Virtual Field Title record with the same group name. This field should already be filled in. **<Enter>** past this field.

Order keeps the fields in a consistant sequence. The first record should be **1**.

Field Name is the name for the virtual field: it should be unique from all other virtual fields that can be loaded at the same time. Enter **FirstGoodDate**.

Length, as in databases, is the length in bytes of the field. Date fields always consume 4 bytes, so enter **4**. To verify the length of any given fieldtype, see the chart on page .

Field Type is the numeric code that tells VariForm what kind of information will be stored in this field. Date fields are type **7**. There is also a PickList **<F2>** that shows all available field types.

Sub Type is the type-modifier, as is true in database fields. This will almost always be **0**.

Initialize contains the value that will be inserted into the field when it is loaded by a procedure. This saves the calling process from having to initialize variables all of the time. Enter **01\01\1901** for this field.

Press **<Ctrl-Enter>** to save this record. The other virtual fields are:

Order	Field Name	Length	Field Type	Sub Type	Initialize
2	RecalcTotals	2	2	0	0
3	DeleteBlanks	2	2	0	0
4	PackSales	2	2	0	0
5	PackClient	2	2	0	0
6	Confirm	2	2	0	0

With the virtual fields entered, lay out the procedure code to determine the missing pieces:

Procedure: ClientDeletion:

```
LoadVirtuals( "ClientDeletionVars" )
Requestor( "ClientDeletionReq" )
Confirm = YesNo( "This may delete records.  Continue?", "N" )
If( Confirm == 1 ) {
        LoopSearch( "KillOldSalesHist" )
        If( RecalcTotals == 1 ) {
                MultiSearch( "DoAllClientTotals", 1 )
        }
        If( DeleteBlanks == 1 ) {
                LoopSearch( "KillInactiveClient" )
        }
        If( PackSales == 1 ) {
```



```
                PackFile( "SalesHistory" )
        }
        If( PackClient == 1 ) {
                PackFile( "ClientInfo" )
        }
}
Else {
        Pause( "Client Deletion Process Aborted" )
}
KillVirtuals( "ClientDeletionVars" )
```

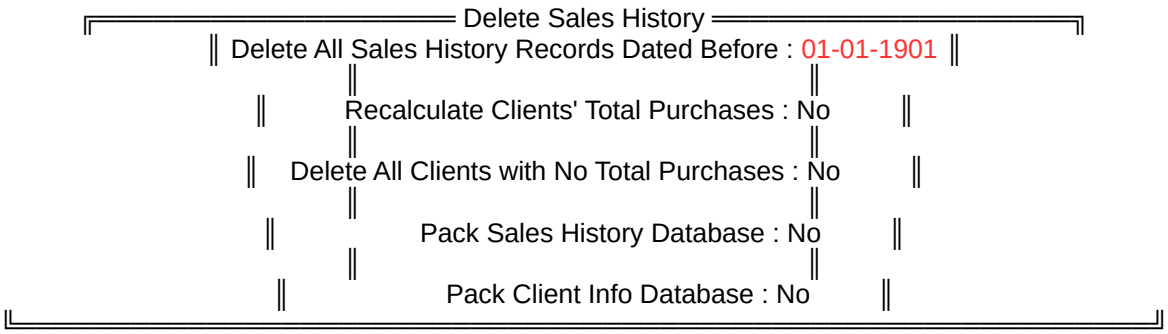

Example System

Step 7: Procedures

Requestors

Requestors are data entry screens that operate exclusively on virtual fields. They do not generally read or write database records, and can be called from procedures, searches, or reports. Requestors give the end-user some level of control over how the search or procedure will function: how much control is actually given is determined by the sophistication of the procedure.

A rough layout of the ClientDeletionReq requestor might look like this:



Client Deletion Requestor

Name	ClientDeletionReq
Database	Virtual
Usage	1
Display	ClientDeletionReq
Help	Data Entry
Title	Delete Sales History
Footer	Requestor to Kill Old Sales History Records
Footer Offset	0
Footer Colors	HelpPlain
DEScreen	
Row	6
Column	10
Height	9
Width	60
True Height	9
True Width	60
Item Height	0
Colors	Requestor
Function Keys	DefaultDEScreen
All Others	Blank or 0

Example System

Step 7: Procedures

Client Deletion Requestor Display Field Records

Group	ClientDeletionReq	ClientDeletionReq	ClientDeletionReq
MetaField	FirstGoodDate	RecalcTotals	DeleteBlanks
Style	7 Date	1 Yes/No Field	1 Yes/No Field
Format	145	1 Editable	1 Editable
F. Bits	Skip this field	Skip this field	Skip this field
Decim	0	0	0
Width2	0	0	0
Row	0	2	4
Column	49	49	49
Height	1	1	1
Width	10	3	3
Prompt1	(See drwg above)	(See drwg above)	(See drwg above)
Row	0	2	4
Column	1	10	4
Height	1	1	1
Width	47	38	44
All others	Blank / 0	Blank / 0	Blank / 0

Group	ClientDeletionReq	ClientDeletionReq
MetaField	PackSales	PackClient
Style	1 Yes/No Field	1 Yes/No Field
Format	1 Editable	1 Editable
F. Bits	Skip this field	Skip this field
Decim	0	0
Width2	0	0
Row	6	8
Column	49	49
Height	1	1
Width	3	3
Prompt1	(See drwg above)	(See drwg above)
Row	6	8
Column	19	21
Height	1	1
Width	29	27
All others	Blank / 0	Blank / 0

This information will generate the requestor shown above. The next step is installing the search logic:

KillOldSalesHist:

Search Sales History in date order, killing all records where the Date field is less than FirstGoodDate

DoAllClientTotals:

Go through the client database, recalculating the TotalPurch field for each record

Example System

Step 7: Procedures

KillInactiveClient:

Go through the client database, killing all records where the TotalPurch field is less than or equal to zero.

Example System

Step 7: Procedures

Search Structure Record

Search **KillOldSalesHist**
Order 0
Direct 0
Comment **Delete Sales History records < FirstGoodDate**
Child SalesHistory
Parent Leave blank
Index Date

Search Field Records

Search **<Enter>** past this field
Type 1 (Decision)
Order 0
Sub Order 0
MetaField (>Date < FirstGoodDate)
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type 2 (Positive Action)
Order 0
Sub Order 0
MetaField KillRec()
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type 6 (Stop Condition)
Order 0
Sub Order 0
MetaField (>Date >= FirstGoodDate)
Advance **<Enter>** past this field

Notes on the Search Strategy:

There is no Search Type 5 record, because the search is working in the Date index order. The earliest date, which is most likely to be deleted, will be pulled first.

Because this Search contains a KillRec statement, it should never be called as a MultiSearch: VariForm will try to find the record following a just-deleted record, which is always an error. Searches will KillRec() commands should be called through either DoSearch() or LoopSearch(), which always re-initialize the search at the beginning of the database.

Example System

Step 7: Procedures

Search Structure Record

Search **DoAllClientTotals**
Order **0**
Direct **0**
Comment **Recalc TotalPurch for all Client Info records**
Child **ClientInfo**
Parent Leave blank
Index **ClientID**

Search Field Records

Search **<Enter>** past this field
Type **1** (Decision)
Order **0**
Sub Order **0**
MetaField **1**
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type **10** (Loop #1 Positive Action)
Order **0**
Sub Order **0**
MetaField **#Target# = >ClientID**
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type **10** (Loop #1 Positive Action)
Order **1**
Sub Order **0**
MetaField **#FloatAnswer# = 0**
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type **10** (Loop #1 Positive Action)
Order **2**
Sub Order **0**
MetaField **MultiSearch("GetClientTotal", 1)**
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type **10** (Loop #1 Positive Action)
Order **3**
Sub Order **0**
MetaField **>TotalPurch = #FloatAnswer# = 0**
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type **10** (Loop #1 Positive Action)

Order 4
Sub Order 0
MetaField EditRec()
Advance <Enter> past this field

Example System

Step 7: Procedures

Search Structure Record

Search **KillInactiveClient**
Order **0**
Direct **0**
Comment **Delete Client Info records <= $0 in TotalPurch**
Child **ClientInfo**
Parent Leave blank
Index **ClientID**

Search Field Records

Search **<Enter>** past this field
Type **1** (Decision)
Order **0**
Sub Order **0**
MetaField **(>TotalPurch <= 0)**
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type **2** (Positive Action)
Order **0**
Sub Order **0**
MetaField **KillRec()**
Advance **<Enter>** past this field

Notes on the Search Strategy:

This Search is considered exahustive, because there is neither a Fast Start (type 5) nor a Fast Stop (type 6) element. This is usually the case when searches are based on unindexed fields, or when a search must use an index that does not properly separate the database for the query being performed.

A faster variation of this Search would exist if the Client Info database contained an index on the TotalPurch field. In that situation, a Fast Stop element could be used that would terminate the Search when it encountered a record with a TotalPurch field value greater than zero.

Because this Search contains a KillRec statement, it should never be called as a MultiSearch: VariForm will try to find the record following a just-deleted record, which is always an error. Searches will KillRec() commands should be called through either DoSearch() or LoopSearch(), which always re-initialize the search at the beginning of the database.

Because of the sophistication of this procedure, it will probably be best to put it directly on the Example Menu. Add a Display Field Record to call the procedure:

Example System

Step 7: Procedures

Group: ExampleMenu
MetaField: Call("ClientDeletion"), Row: 5, Column: 0
Prompt1: " Purge Clients", Row: 5, Column: 0, Width: 15
Prompt2: "P", Row: 5, Column 1, Width 1.

Example System

Step 8: Reports

The next step in basic system design is output. Whether this output takes the form of Xbase data files, printed information summaries, or mailing labels, the process for selecting, designing, and generating this output is the same. The general process of creating these output files is called reporting or report generation.

Under VariForm, reports are another use of the Display System: A display structure can pop a requestor, access a search, and define a page; the display field records define the text output, its location within the page, and may call sub-reports. Reports actually use the same nested-field mechanism for sub-reports that is available to the data entry system for calling limited browsers.

Designing a report is similar to laying out a data entry screen. Designing a sub-report is more like building a browser. Reports have two main variations from the on-computer display system:

- Reports are colorless: the colors of the on-screen display do not affect the output that is extracted to the file.

- Reports may use special 'attribute' characters, which, if used properly, can cause prompts or fields to print out with special character strings that tell the printer to use different fonts (Pica vs. Elite), different character sizes (10 pitch, 12 pitch, et cetera), or different character attributes (underline, bold, compressed).

The example client wants two separate outputs: a label generator that will print out labels for all clients, and a regional summary report that takes a selected sales region and reports on total sales by client within that region. (Both the Client and Sale record must be in the targeted region to qualify for the second report).

Before beginning any report design, some information must be gathered:

- How many lines per inch does the printer print by default

- How many characters per inch within a line

- What is the size of the receiving media, in characters and lines

- Where is the printer connected / what is it's "file address"

The most common answers are:

- 6 lines per inch: 8 lines in condensed mode

- 10 or 12 characters per inch: often misnamed 'Pica' and 'Elite'

Example System

- Media dependant:
    - 8½" x 11" paper: 80 or 96 characters by 66 lines on line printers
    - On a laser printer, page length will be 60-64 lines
    - Labels: 40 or 48 characters by 5 or 4 lines with a 1 line gap

- Most PC systems will accept "LPT1" as a valid printer destination

Example System

Step 8: Reports

Report 1: Label Generator

Multiple Item Strategy

The initial layout of a label generator is done in the same fashion as a multi-line browser: Define an item, determine its height, and generate field records that reflect that height.

The distinction between a report and a browser lies in the use of the display fields. In a browser, the prompts for the field 'columns' are kept within the associated field record. This facilitates the Browser Entry Mode, where prompts need to be highlighted to show the user what the current field is.

Reports, on the other hand, treat field prompts differently: there is a routine in the report generator which 'moves' the field set around on the display screen, based on the editability of the field. If a field is uneditable, it is only displayed once per screen. So in terms of columnar reports, there will be one or more 'title' fields, which will be uneditable text strings, displayed at the top of each screen, and several 'mobile' fields, which will be editable.

In this example, the labels will have no titles. Assume the lables are 1-up, with 11 to an 8½ x 5½ page, with the label occupying the central 4½". Applying the 'most common printer answers' from before, the following values are calculated:

Screen (media) height: 66 lines
Screen (media) width: 55 characters
Label (item) height: 6 lines (5 printable, 1 space)
Label (item) width: 50 characters (5 indent, then 45)

Label Generator Display Structure

Name	ClientLabels
Database	ClientInfo
Usage	2 Report
Display	ClientLabels
Help	
Title	
Footer	
Footer Offset	0
Footer Colors	HelpPlain
DEScreen	
Row	0
Column	15
Height	18
Width	50
True Height	66
True Width	50
Item Height	6
Colors	DEScreen2
Function Keys	

Example System

Step 8: Reports

Search	ClientLabels
All Others	Blank or 0

Example System

Step 8: Reports

Client Label Report Display Field Records

Group	ClientLabels	ClientLabels	ClientLabels
MetaField	#Target#	Address1	Address2
Style	2 Character String	2 Character String	2 Character String
Format	1 Editable	1 Editable	1 Editable
F. Bits	Skip this field	Skip this field	Skip this field
Decim	0	0	0
Width2	0	0	0
Row	0	1	2
Column	5	5	5
Height	1	1	1
Width	39	39	39
All others	Blank / 0	Blank / 0	Blank / 0

Group	ClientLabels
MetaField	#Target2#
Style	2 Character String
Format	1 Editable
F. Bits	Skip this field
Decim	0
Width2	0
Row	3
Column	5
Height	1
Width	39
All others	Blank / 0

Note that #Target# and #Target2# are used instead of 3 name fields and city/state/zip. The reason for this is to allow the actual fields to 'blend' together. Remember that in data entry screens and browsers, the fields displayed are always the full field width. The virtual fields will be assembled by the Search Condition.

Search Structure Record

Search **ClientLabels**
Order **0**
Direct **0**
Comment **Select & Preprocess Client Info records for labeling**
Child **ClientInfo**
Parent Leave blank
Index **ClientID**

Search Field Records

Example System

Step 8: Reports

Search **<Enter>** past this field
Type 1 (Decision)
Order 0
Sub Order 0
MetaField **1** [This selects all ClientInfo records]
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type 10 (Loop #1 Positive Action)
Order 0
Sub Order 0
MetaField **#Target# = (Title + " " + FirstName " " + LastName)**
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type 10 (Loop #1 Positive Action)
Order 1
Sub Order 0
MetaField **#Target2# = (City + " " + State + " " + Zip)**
Advance **<Enter>** past this field

Now that the display structure, display fields, and the search is in place, this report is ready to be used. Attach this report to the example menu by adding the last available menu line (for the current size of ExampleMenu) to the Display Field database:

Group: ExampleMenu
MetaField: ReportCall("ClientLabels", "LPT1"), Row: 6, Column: 0
Prompt1: " Client Labels", Row: 6, Column: 0, Width: 15
Prompt2: "L", Row: 5, Column 8, Width 1.

Selecting the Client Labels option from the example menu will now send out neatly formatted labels for all Client Info records to the printer attached to the "LPT1" port.

If it is necessary to test this report, drop to the Command Line Interface (<Shift-F3>) from the menu system, and enter: ReportCall("ClientLabels", ""). This will generate the label report to the computer screen, pausing at the end of each 'page'.

Example System

Step 8: Reports

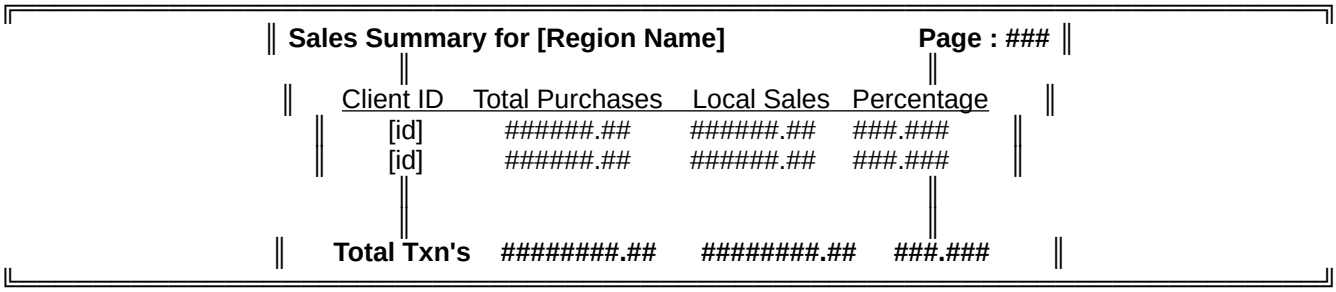
Report 2: Region Summary

Complex Report Strategy

Complex reports are reports with many layers. They require a little more planning than do simple multi-item reports. Basically, a complex report is one or more multi-item sub reports inside a single-item report.

The first display structure defines the outside page boundaries. If the sub reports exceed this boundary, they will be truncated. If the sub reports fit in the master page, but the amount on information exceeds the sub report's boundaries, VariForm will complete the page, and make a new page for the balance of the sub report.

To design the report, start with a generic 'layout' the describes how the report should appear. The following layout shows the highlights of the report, but the repetetive section has been abbreviated:



Observe the area of repetitive information: from "Client ID" down to the totals: this is the sub-report. The primary page includes the top line, and the Total Transactions line. Also note the different 'fonts' used: fields on the primary page are in **bold**, while the secondary report's header is underlined.

Design the primary report first, keeping in mind that the primary report will determine the 'page' size.

Example System

Step 8: Reports

Primary Report for Sales Summary

Name	SalesSummary1
Database	SalesRegion
Usage	2 (Report)
Display	SalesSummary1
Help	
Title	
Footer	
Footer Offset	0
Footer Colors	HelpPlain
DEScreen	
Row	0
Column	0
Height	3
Width	70
True Height	66
True Width	70
Item Height	0
Colors	DEScreen2
Function Keys	
Search	OneShot
All Others	Blank or 0

On this report, the SalesRegion database is be used, since this report will be run from the SalesRegionBrow browser. The 'OneShot' search provides for the primary page to be run one time, as opposed to a multi-region report.

Display Field Records

Group	SalesSummary1	SalesSummary1	SalesSummary1
MetaField	RegionName	Page	
Style	2 Character String	4 Standard Numeric	8 Nested Field
Format	1 Editable	1 Editable	0 Uneditable
F. Bits	Skip this field	Skip this field	Skip this field
Decim	0	0	4
Width2	0	0	0
Row	0	0	2
Column	1	64	5
Height	21	1	1
Width	40	3	0
Prompt1	\BSales Summary for	Page :	
Row	0	0	0
Column	1	57	0
Height	1	1	0
Width	19	6	0
Prompt2		\B	
Row	0	0	0
Column	0	68	0
Height	0	1	0

Example System

Step 8: Reports

Width	0	2	0
Lookup			SalesSummary2
All others	Blank / 0	Blank / 0	Blank / 0

Example System

Step 8: Reports

Group	SalesSummary1	SalesSummary1	SalesSummary1
MetaField	#SSFloat1#	#SSFloat2#	#SSFloat3#
Style	4 Standard Numeric	4 Standard Numeric	4 Standard Numeric
Format	See Format Bits	See Format Bits	See Format Bits
F. Bits	Editable, Decimals	Editable, Decimals	Editable, Decimals
	Right-Justify, 0's	Right-Justify, 0's	Right-Justify, 0's
Decim	2	2	3
Width2	0	0	0
Row	0	60	60
Column	60	38	54
Height	21	1	1
Width	11	11	7
Prompt1	\BTotal Txn's		
Row	60	0	0
Column	6	0	0
Height	1	0	0
Width	13	0	0
Prompt2			\B
Row	0	0	60
Column	0	0	68
Height	0	0	1
Width	0	0	2
All others	Blank / 0	Blank / 0	Blank / 0

Note the placement of the \B format codes. Character formatting codes should always be paired, so that the report page finishes printing in the same state that it began.

Observe the Nested Field: it is a type 4 nested field, just like a browser. The name of the next layer of reporting is in the lookup field, also like a browser.

The location of the nested field determines where on the main report the sub report will appear. This affects both the uppermost row (line 2), and the leftmost column (sub-report column 0 will be placed on the main report's column 5).

The sub report design is a little more complicated. The editable status of a display field determines whether or not the field is relocated and redrawn for each applicable record: Uneditable display fields are 'immobile', while editable fields will be moved down 1 itemheight number of rows for each record to be displayed.

Example System

Step 8: Reports

Sub Report for the Sales Summary

Name	SalesSummary2
Database	ClientInfo
Usage	2 (Report)
Display	SalesSummary2
Help	
Title	
Footer	
Footer Offset	0
Footer Colors	HelpPlain
DEScreen	
Row	2
Column	5
Height	56
Width	64
True Height 56	
True Width	60
Item Height 1	
Colors	DEScreen2
Function Keys	
Search	SalesSummary2A
All Others	Blank or 0

Note that this report is using ClientInfo for the central access: the numbers being accumulated are only for clients within the sales region. The designation of 'SalesSummary2A' is based on the need for a subsidiary search to accumulate the total sales within the region for a client. The results of the subsidiary search will be returned in the #SSFloat## fields, for display on both the secondary report and the primary report's baseline totals.

Display Field Records

Group	SalesSummary2	SalesSummary2	SalesSummary2
MetaField		ClientID	TotalPurch
Style	0 Text String	2 Character String	4 Standard Numeric
Format	0 Uneditable	1 Editable	1 Editable
F. Bits	Skip this field	Skip this field	Skip this field
Decim	0	0	2
Width2	0	0	0
Row	0	1	1
Column	0	8	23
Height	0	1	1
Width	0	4	9
Prompt1	\U (see drawing)		
Row	0	0	0
Column	0	0	0
Height	1	0	0
Width	60	0	0
Prompt2	\U		
Row	0	0	0

Example System

Step 8: Reports

Column	61	0	0
Height	1	0	0
Width	2	0	0
All others	Blank / 0	Blank / 0	Blank / 0

Example System

Step 8: Reports

Group	SalesSummary2	SalesSummary2
MetaField	#SSFloat4#	#SSFloat5#
Style	4 Standard Numeric	4 Standard Numeric
Format	1 Uneditable	1 Editable
F. Bits	Skip this field	Skip this field
Decim	2	3
Width2	0	0
Row	1	1
Column	40	54
Height	1	1
Width	9	7
All others	Blank / 0	Blank / 0

The first search, SalesSummary2A, assumes a 'parent' database within the report structure of 'SalesRegion' will provide the Region to Search for. Searches within subreports are treated as multisearch(search, 1) calls. Actions will be defined as Type 10 / Type 11 action records. The first search is as follows:

Search Structure Record

Search **SalesSummary2A**
Order **0**
Direct **0**
Comment **Locate clients in #Target# Sales Region**
Child **ClientInfo**
Parent Leave blank
Index **HomeRegion**

Search Field Records

Search **<Enter>** past this field
Type **1** (Decision)
Order **0**
Sub Order **0**
MetaField **(>SalesRegion == #Target#)**
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type **5** (Start Condition)
Order **0**
Sub Order **0**
MetaField **>SalesRegion = #Target#**
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type **6** (Stop Condition)
Order **0**
Sub Order **0**

MetaField **(>SalesRegion > #Target#)**
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type **10** (Loop #1 Positive Action)
Order 0
Sub Order **0**
MetaField **#SSFloat1# = (#SSFloat1# + >TotalPurch)**
Advance **<Enter>** past this field

Note: #SSFloat1# is the running total for the primary report

Search **<Enter>** past this field
Type **10** (Loop #1 Positive Action)
Order 1
Sub Order **0**
MetaField **#SSFloat4# = 0**
Advance **<Enter>** past this field

Note: #SSFloat4# will recieve the total regional sales history per client, and so must be initialized between clients.

Search **<Enter>** past this field
Type **10** (Loop #1 Positive Action)
Order 2
Sub Order **0**
MetaField **MultiSearch("SalesSummary2B", 1)**
Advance **<Enter>** past this field

Note: This routine will search the SalesHistory database and generate a regional total for the current client within the #Target# region.

Search **<Enter>** past this field
Type **10** (Loop #1 Positive Action)
Order 3
Sub Order **0**
MetaField **#SSFloat5# = (>TotalPurch / #SSFloat4#)**
Advance **<Enter>** past this field

Note: This maintains the overall average percentage for the primary report.

Search **<Enter>** past this field
Type **10** (Loop #1 Positive Action)
Order 4
Sub Order **0**
MetaField **#SSFloat2# = (#SSFloat2# + #SSFloat4#)**
Advance **<Enter>** past this field

Note: This maintains the running regional total for the primary report.

Example System

Step 8: Reports

Search **<Enter>** past this field
Type **10** (Loop #1 Positive Action)
Order **5**
Sub Order **0**
MetaField **#SSFloat3# = (#SSFloat2# / #SSFloat1#)**
Advance **<Enter>** past this field

Example System

Step 8: Reports

As noted above, the secondary search, SalesSummary2B, will collate the appropriate SalesHistory records for the current client within the #Target# region.

Search **SalesSummary2B**
Order 0
Direct 0
Comment **Locate sales history records for sales summary report**
Child SalesHistory
Parent Leave blank
Index Client

Search Field Records

Search **<Enter>** past this field
Type 1 (Decision)
Order 0
Sub Order **0**
MetaField **((>ClientID == <ClientID) and (>SalesRegion == #Target#))**
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type 5 (Start Condition)
Order 0
Sub Order **0**
MetaField **>ClientID = <ClientID**
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type 6 (Stop Condition)
Order 0
Sub Order **0**
MetaField **(>ClientID > <ClientID)**
Advance **<Enter>** past this field

Search **<Enter>** past this field
Type 10 (Loop #1 Positive Action)
Order 0
Sub Order **0**
MetaField **#SSFloat4# = (#SSFloat4# + >PurchaseAmount)**
Advance **<Enter>** past this field

Example System

Step 8: Reports

The displays, fields and searches define the report. The searches and the report use a few non-standard virtual fields: #SSFloat1# - #SSFloat5#. These need to be installed as a virtual field group named "SSFloaters", all 5 fields are floating point (8 bytes long), and should be initialized to 0.

This report will be attached to the Sales Region browser with a Function Key <Shift-F7>. Since SalesRegionBrow currently uses the DefaultBrowser function key set:

- Go the the Function Key Browser
- Make a copy of the DefaultBrowser record
- Rename the copy to "SalesRegionBrowKeys"
- Add a key record for Key 7, Enhancement 1, Action: Call("DoSalesSummary")

- Go to the Display Structures Browser
- Edit the SalesRegionBrow record
- Change the Browser F. Keys field to "SalesRegionBrowKeys"

- Go to the Procedures Browser
- Create a procedure called "DoSalesSummary"
  - The procedure can be described as "Call Sales Summary Report"
  - This will be a four line procedure:

```
LoadVirtuals( "SSFloaters" )
#Target# = <RegionID
ReportCall( "SalesSummary1", "LPT1" )
KillVirtuals( "SSFloaters" )
```

After installing the function key and procedure, the <Shift-F7> key will generate a SalesSummary report for the current record, and send it to the printer on LPT1.

This concludes the Example Database. A more detailed example is available from Parker Software, in the "Programming Style" book, which provides guidelines on making a more comprehensible user interface, and more sophisticated examples, such as report requestors, and large-scale batch jobs.

Example System

Step 9: Help Topics

Programmers and System Developers have been hounded in the past because they failed to supply or emphasize a help system. Books, training videos, and tutorial classes can help the end-user know the basics. There will be times when the manual is out of reach, the user gets distracted, and when the user turns back to the browser, data entry screen, or menu, he will draw a blank: why is he here, what is he doing, what function keys are available.

VariForm provides rudimentary help through the cue lines at the bottom of the computer screen. All display structures have a cue line, which can represent one or two lines of function keys and area description. Individualized fields: data entry fields, menu selections, and browser fields when in Entry Mode, have a cue line that can provide one or two lines of additional, field-specific information.

A key that is in every 'default' function key setting is F1 - Help. The help key allows the user to a specialized multi-colored window full of information about a particular 'topic' or concept. The location of the user, and the content of various help fields in the display system will determine what help group is provided initially:

- From a menu:
  - If the current field has an entry in 'help', it will appear
  - Otherwise, if the display structure of the menu has an entry
  - Otherwise, the group called "Menu System" will be displayed

- From a browser in standard (browse) mode:
  - If the browser's display structure has an entry in BHelp
  - Otherwise, the group called "Browsers"

- From a browser in Entry Mode:
  - If the current field has an entry for help, it will appear
  - Otherwise, if the browser's display structure has an entry in Help
  - Otherwise, the group called "Browser Entry"

- From a data entry screen:
  - If the current field has an entry for help, it will appear
  - Otherwise, if the screen's display structure has an entry in Help
  - Otherwise, the group called "Data Entry"

- From the help window:
  - The group called "Help on Help"

This example will supply help, with cross-referencing, for the Client Info data entry screen.

Example System

Step 9: Help Topics

First, lay out the text that should be there, and what field it should be attached to.

(ClientInfoGeneral):

The ClientInfo database contains records that identify specific clients, their name and address, and some basic sales related information. This data entry screen also provides access its child database: SalesHistory, through a Nested Field at the bottom of the screen.

ClientID: The ClientID field contains a unique value that will always be used for this client. All sales history records with a matching ClientID field will be attached to the record as children. See also ClientInfoGeneral.

First Purchase:

The FirstPurch field is maintained by the 'Client Calculator' procedures, and represent the earliest dated Sales History record attached to this client.

Total Purchase:

The TotalPurch field is maintained by the 'Client Calculator' procedures, and represent the total of all Sales History records attached to this client.

Name: The three fields named Title, First Name, and Last Name, make up the Client's name. Title refers to Mr., Mrs., Ms., or M/M. as a prefix for mailing labels and mass mailings.

Address: The five fields named Address1, Address2, City, State, and Zip, make up the client's address. The address is used for mailing labels. If there is only one line of address, use the upper line (address1).

Sales Region:

The sales region field contains an abbreviation that indicates what area the client is located in. This is used for determining whether the client habitually purchases locally, or out of his sales region.

Sales History:

This is a Nested field. Pressing <Enter> on this field will access the Sales History database, and bring up a browser that shows only the records associated with this client. To leave this field, use the <Tab> or <Arrow / Page> keys.

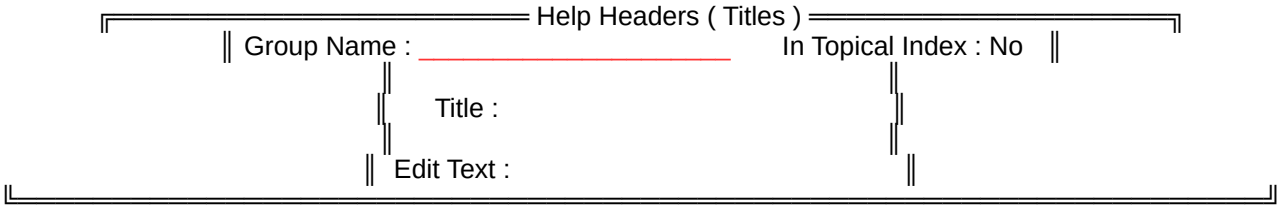
Next, establish a naming convention so that these individual help groups will appear together in the help database: this makes re-editing later easier.

In VariForm, a common naming convention is a three or four character database id, followed by a number (for sequence). The only exceptions to this convention are groups that will be cross-referenced by other help topics, such as ClientInfoGeneral. Since this help is for the ClientInfoEntry screen, the values for the remaining fields, would be CIE1 through CIE7.

Example System

Step 9: Help Topics

To enter this information, go to the Help System selection on the VariForm System submenu. When the help browser appears, press <F3> to generate a blank Help Header record.



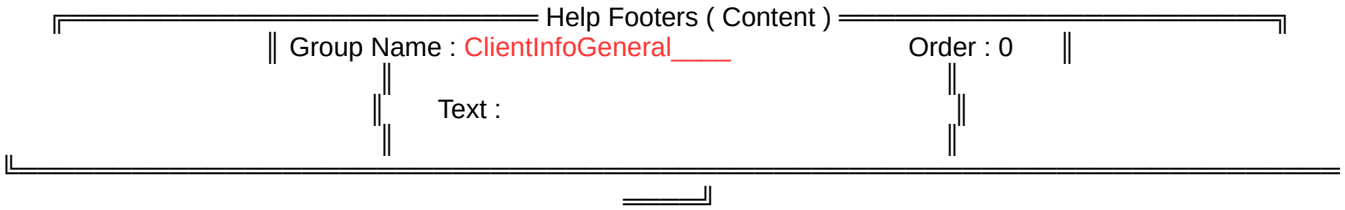
Group Name contains the unique identifier for this help call. Enter **ClientInfoGeneral**.

In Topical Index refers to whether or not this help should be listed in the "Master Index of Help Topics". Type **<Y>**, and the field should become 'Yes'.

Title contains the title for the help window, when this group is current. Enter **General Information about the Client Info D-E Screen**.

Edit Text is a nested field that brings up the Help Footers (Contents) browser. Press **<Enter>**.

When the Help Footers browser appears, press **<F3>** to add a record.



Group Name attaches the help footers to the help header record. It will already be filled in appropriately. **<Enter>** past this field.

Order is a numeric field that is used to keep the help text in a specific sequence. Because of the order field, the 'short-cut' that was shown on page can be used here:

- Enter a blank record
- Make enough copies to complete the entry
- Use the renumbering function (shift-F9 or shift-F10 from the browser)
- Complete the entries using Browser Entry Mode

Text is for the actual contents of the help line to appear. For the short-cut method of entry, leave this blank and save the record.

Make copies of this record, and use the Entry Mode to fill in the text from the previous page. **<Esc>** from this browser when done, and **<Ctrl-Enter>** to save the parent record on the way out.

The balance of the data entry should be straightforward. Remember to surround the word

Example System

Step 9: Help Topics

"ClientInfoGeneral" in ClientID (Help: CIE1) with /i statements, to vary the color and to tell the Help System that the phrase is a valid "index" reference.

Example System

Step 9: Help Topics

Once the eight help groups have been entered into the Help System database, their references must be put in the appropriate display fields. Go to the Display Field section of the VariForm System submenu, and locate the ClientInfoEntry group with the browser.

Change the "Help" fields in this fashion:

MetaField Name	Help field
ClientID	CIE1
FirstPurch	CIE2
TotalPurch	CIE3
Title	CIE4
FirstName	CIE4
LastName	CIE4
Address1	CIE5
Address2	CIE5
City	CIE5
State	CIE5
Zip	CIE5
SalesRegion	CIE6
(Blank)	CIE7

Note: since all of the fields have help, there is no need to add help for the ClientInfoEntry Display Structure. While it would seem to make sense to do so, there are no fields without help, so there is no way for the user to access ClientInfoGeneral help through the normal help system channel. A function key could be added to the screen that would call Help("ClientInfoGeneral"), which could be useful, but the main intent for that help was for the master index, and a cross-reference under ClientID.

This concludes the example system, as well as the main manual.

Appendix 1: Shortcuts

Rebuilding Indexes Vs. Packing a Database

Neither operation is trivial: Rebuilding and Packing can take more than 20 minutes per database if the database is large, if there are many indexes, or if the hard drive doing the work is slow. **Neither operation should ever, ever be attempted unless absolutely certain there are no other users accessing the database at that time.**

In general, rebuild is a quick-fix for minor trouble: corrupt index, accidental reboots, power failures, et cetera, where as pack is a sure-fire, self-contained cleanup system. Packing a database re-sequences records, removes deleted records, attempts to correct damaged records in a database, then rebuilds indexes. An index rebuild is faster, because it does less than a database pack: rebuilding is the last step of packing.

Reasons to Pack:

If more than 10% of the total number of records are erased (Use the info() key on an appropriate browser), the database should be packed.

If any records are listed as damaged on a standalone system, or if more than 2 records are listed as damaged on a multiuser system, the database should be packed.

If a record appears at the beginning or end of any index, and its contents are machine garbage (untypeable characters), the database should be packed.

Why Rebuild:

If a VariForm System Database, such as User Data, is damaged, VariForm will not run until the index is rebuilt.

Rebuilding is up to 2 times faster than packing a database.

Rebuilding does not make a .bak file, so uses less disk space.

MultiUser systems can access a rebuilding database, once the rebuild has begun working on the last index in the database.

To Pack a Database:

If possible, go to the database's browser and press the Pack (default = F10) key.

Otherwise, drop to the CommandLine Interface, and enter the command CallPack(databasename).

Alternatively, enter the name of the database(s) to be packed in the "Packing List" database, drop to the CommandLine, and enter the BatchPack() command, or select BatchPack from the VariForm System Menu.

Appendix 1: Shortcuts

Rebuilding Indexes Vs. Packing a Database

To rebuild indexes:

If possible, go to the database's browser and press the Rebuild (default = Shift-F10) key.

Otherwise, drop to the CommandLine Interface, and enter the command CallRebuild(databasename, indexname), or RebuildAll(databasename).

Alternatively, or if VariForm will not run, delete the index files from the DOS command prompt, and then run VariForm.

Deleting Index Files:

1: Get to the DOS command prompt. It should look like:

C:\>
(There may be a path name involved: Ignore it.)

2: Get on the appropriate drive. If the Drive Letter is different from your command prompt, type:

<Drive-letter>:<Enter>
example: for Drive D, type:
D:<Enter>

3: Make the appropriate directory current. If the directory is "c:\VariForm", then type:

cd c:\VariForm<Enter>

4: The DOS prompt should now read:

C:\VARIFORM\>

5a: If you know the filename of the database in trouble, type:

ERASE <filename>.0*<Enter>

Note: Insert the actual file name where the example shows <filename>. Omit the <> signs, they are only for clarity in the manual. Some DOS Versions want DELETE instead of ERASE.

5b: If you don't know the filename of the database, type:

ERASE *.0*<Enter>

Note: Some DOS Versions want DELETE instead of ERASE.

6) Rerun VariForm. It will sense the missing indexes when it accesses the database, and will rebuild them.

Appendix 1: Shortcuts

Index Calculations

Generic Suggestions

Heavy editing: If the database is doing a lot of editing work, it is better to have several small buffers than a few large ones:

If a database has 4 indexes,
10 - 512 byte buffers will edit as fast or faster than 5 - 1024s.

Light editing: If the database is doing a lot of adding/deleting work, and not so much editing, it is better to have some large indexes:

If a dabase has 4 indexes,
5 - 1024 byte buffers will add/delete as fast or faster than 10 - 512's.

Little/No editing: If the database is a seldom-changed 'reference table', it is better to have a few huge indexes:

Regardless of the number of indexes,
2 - 3192 byte buffers will search faster than 3 - 2048's.

Specific Formulas

Free RAM consumed by indexes:
(Number of Buffers + 1) * (Buffer Size + (4 * Tree Depth))

Maximum number of record entries a buffer can hold (MNE):
(Buffer Size - 10) / Size of Largest Index

Note: To find size of the largest index, go through the index records, and for each record, add all of the field sizes together. Take the larges result, and add 14 for the above formula.

Hint: For editing, which must reorganize this list, you want a small Buffer, since most edits cause entries to 'jump buffers'. Adding and deleting records reorganize Buffers more often than entries, so it is better to give them larger buffers that will be jumped less often.

Maximum number of records that the Largest Index can hold:
(MNE + 1) * (((MNE / 2) + 1) ^ (Tree Depth - 1))

Note: The carat (^) above means "to the power of". MNE refers to the Maximum Number of Entries in a buffer.

Making Changes

To change the number of Buffers, simply change the value in the Database of Databases record; VariForm will use the new size the next time the database is opened.

To change either the Tree Depth or the Size of a buffer, the database must be either packed, or all indexes must be rebuilt immediately following the change.

Appendix 1: Shortcuts

Time-Saving Procedure Keys

Database of Database Fields:

Renumber field orders to be sequential with <Shift-F10>

Recalculate all field offsets correctly with <Ctrl-F10>

Warning: May not work properly with Unioned Fields

Database of Database Indexes:

Load default field size by pressing <F3> on the appropriate Field#Name.

To Generate "Quick" Display Units of a Database:

From the Database of Databases Browser, <Shift-F10> will load a requestor that asks for display type to generate, generates a recommended name (pre-checked for uniqueness), provides user an option to change the name, and generate the structure and sub-structures. Structures are as follows:

- Browser: Makes a browser with first field from each index, no search conditions, default color schemes and function keys.

- DEScreen: Makes a data entry screen with one field per line, with the field name as the prompt, default color schemes, no field-function keys.

- Report: Generates a report format Display Structure scaled as a data entry screen. Uses default printer's page settings for sizing.

- SubReport: Generates a report format Display Structure scaled as a browser. Uses default printer's page settings for sizing.

Display Structure Records for Data Entry:

Display is often the same as Name. <F3> on either field will copy the Name field contents into the Display Field.

Centering can be a tiring task. <F3> on any of Row, Column, Height, or Width, will correct Row and Column to screen center (80 x 21 basis), as per Height and Width.

Display Field Records:

Re-calculating Prompt positions can become mind-numbing. <F3> on Prompt1Row will generate appropriate Row, Column, Width, and Height Settings.

To save character-counting, <F4> on Prompt1Text will update Prompt1Width.

Everything that is true for Prompt1 is true for Prompt2, too.

Procedure Records:

Renumber the procedure lines automatically, with <Shift-F10> from the detail browser. Renumbers all records to start from 1, sub 0, counting in whole numbers, all sub 0.

Search Records:

Renumber searchfield records automatically within a SearchField type, with <Shift-F10> on any record of the targeted type. Renumbers all records to start from 1, sub 0, counting in whole numbers, all sub 0.

Appendix 2: The Error System

To help diagnose problems, VariForm has a complex error reporting system : each error or displayed message can, in the worst case, be stored to the Error/Action Database to help track down errors during development or testing of an application.

Errors contain 5 parts, that answer 5 questions:

Severity How important / significant is this error or message. This value tells VariForm whether to display the message, Pause the Message (press any key to continue...), or store the message in the Error/Action Database.

Region Which module in VariForm is doing the complaining? This value tells the Developer more precisely what VariForm was trying to do when the problem occurred.

Message This is the number that controls the specific text of the message: what precisely was the error?

Text If VariForm has enough memory available, it will load a text message from the ErrorMessage section of the Charts Database, fill in the blanks, and send that message to the user and/or the file. This helps pinpoint what specific step was occurring when the error occurred.

User Name of the user (as logged in at VariForm Startup) having or finding the error. On standalone systems, this will almost always be "System Default".

Appendix 2: The Error System

Error Severity

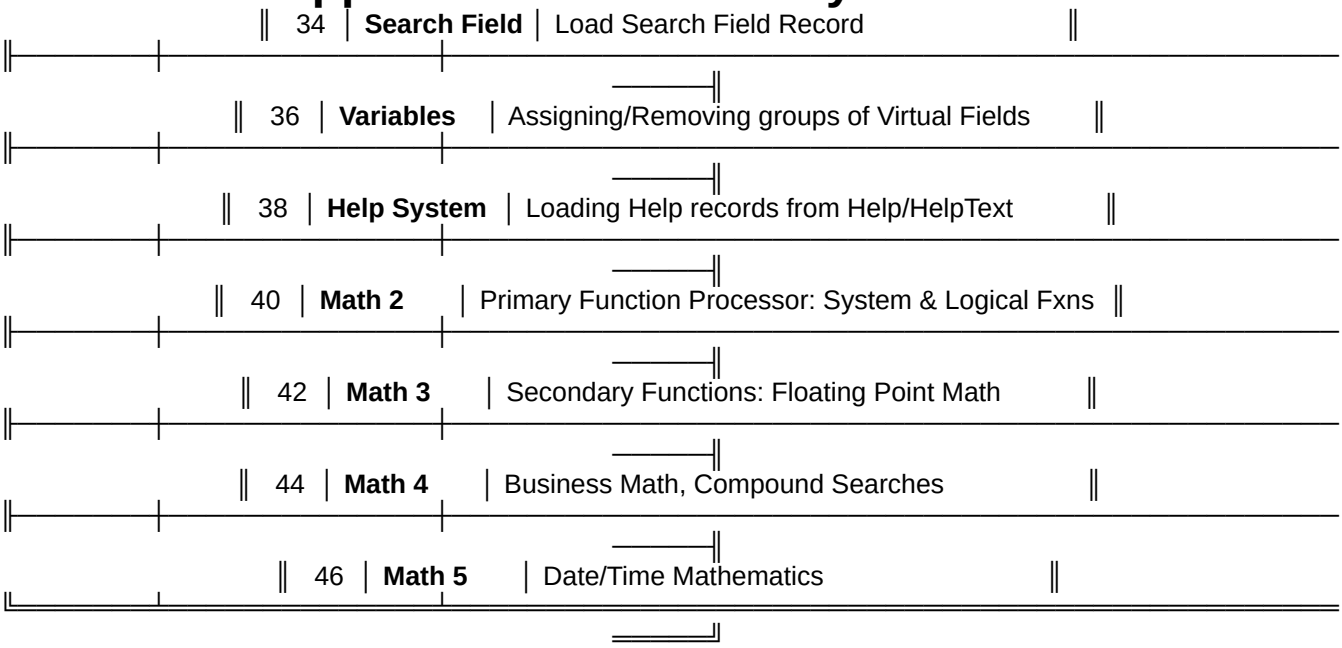
Value	Official Name	Meaning
0	**Detail**	Startup and ShutDown statements
		Every Single statement of a MultiSearch/Map Counter
2	**OverView**	MultiSearch/Map Entry statements
4	**Text Info**	User and VariForm Text Displays
		Out of Memory / Low on Memory Statements
6	**Bad Reference**	No such search, No such FKey
		Search Failures that can be worked around
10	**Simple Error**	Impossible Screen Size
		Non-Unioned Fields over/underlap
12	**Action Blocked**	No such display system, No Fields in display
		Critical out of memory
20	**Act. Imposible**	Allocate -1 bytes, No such Field to remove
		Rename/Delete nonexistant file
30	**External Block**	Dos Error prevented action
		Out of file handles, bad file, damaged index
		Simple sharing violation
40	**External Reset**	Dos Multiple error: cleanup and reset
		Nonexistant file handle, corrupt index
		Extreme sharing violation
50	**External Abort**	Dos requested immediate shutdown
		Repetitive errors, FAT damaged
		EMS location errors

Appendix 2: The Error System

Error Region

Value	Module Name	Action Occuring
0	**BPlus**	Index Processing
1	**Unknown**	Memory (Allocation) Related
2	**Browser**	Load Browser or SubBrowser
4	**Color**	Load Color Record
6	**Database**	Load / Unload Database Record
8	**DBField**	Load / Unload Database Field Element
10	**DBIndex**	Load Index Record or Build .0* file
12	**DEStruct**	Load Display Record & Build Structure
14	**DisplayField**	Load DisplayField Record
16	**Drawing Sys.**	Display Record Info in a Display Structure
18	**Globalio**	Disk I/O including Disk Caching System
20	**Function Key**	Load Function Key Group or Record Element
22	**Math Core**	Process and CommandLine / Command Line Interface
24	**Menu**	Load Menu record, process Menu Arrow Keys
26	**Procedure**	Process If/ElseIf/Else/Do/While, Load Procedure
28	**Relate**	Load Search Record, Load Child Database
30	**Report**	During Reporting: Load subreport. Else Stack.
32	**Search**	Database or index error. Impossible Search Set.

Appendix 2: The Error System



Appendix 2: The Error System

Error Messages

Preface to Error Messages: Error messages are designed to tell the user not only what was wrong, but specifically where and why the problem(s) occurred. Most of the time, the text messages in the second column will display, but if VariForm runs into critical/shutdown errors, or runs completely out of memory, only the numbers for Severity, Region and Message will show.

Complaints from DOS: There are several sets of errors which VariForm will report that relate to either DOS or Network settings. They are shown by the words:

Name	VariForm #'s	Meaning
Extended	92 - 182	Dos Extended Error: See DOS Manual
Err Class	183 - 195	Specific Error Category
Err Locus	196 - 200	Broad Error Category
Err Action	201 - 207	Suggested Remedial Action
Std	213 - 248	Dos Basic Error: See DOS Manual

Most VariForm errors can be remedied with VariForm changes; most DOS errors relate to more involved problems. DOS errors appear when there are not enough FILES or File_Handles, or when a user attempts to access a network drive that they are not currently connected to. DOS errors also signify hardware problems, such as a damaged hard drive, bad memory chips, faulty network connections, and so forth.

There is one special exception, where VariForm errors can generate tons and tons of DOS error messages: corrupt indexes. If a user made changes to a database, and OnNetwork was 0, and they didn't flush files, all indexes that should have changed may be corrupted. If a user rebooted while running a procedure that edits, adds, or deletes records, or if the power to the computer was turned off during such a procedure, all indexes on databases that have been changed may be corrupted.

Corrupt indexes will show themselves with DOS file errors, such as "Error 0", or "Invalid File Handle". The best remedy is to pack the damaged databases, if the browser or CommandLine Interface are accessible. If not, see the section on "Rebuilding Indexes" in Appendix 1 (page).

Special Note: These messages, both here and in the Charts Database (see the ErrorMessage Section of the Charts database) there are characters preceded by a percent (%) sign. These tell VariForm to add Text (%s) or numeric (%d) information before displaying the string, to further clarify the error. While users may change the position of these 'format characters', they should never add or remove % signs or re-arrange their order within an error message. To do so defeats the purpose of the error display system, and could lead to errors that would be impossible to track.

Appendix 2: The Error System

Error Messages

Value Meaning
Diagnostic Info

0 Cannot Load Browser Display (%s)
This is usually preceded by a message that explains why the browser didn't load: out of memory, bad data, disk error. If this is the first message, then the display named in %s does not exist in the Display Database.

1 Load Browser Requires a Search Structure
Sub-browsers, PickLists, BrowseLists and Sub-Reports all require a Search Structure. The search structure may limit the portion of the database, and must tell the Pick/Browse List what to return if the user makes a selection. Fix by filling in the "Search" field in the Display Record for the browser.

2 No field to search in index %s
When browsing a limited index, the Browser Search routine will use the first non-mandatory field. That is, it counts the number of type 5 entries in the Search Structure, and counts that many elements off of the current index. Fix by either removing some type 5 entries, or adding a field for searching to the end of the named index.

3 Cannot load '%s' browser
See error 0

4 Invalid / Unlimited Sub-browser (%s)
A sub-browser, as called from a Nested Field, MUST contain a Search Structure. Since sub-browsers are for use in related databases, this search should include mapping information (type 4 records. Fix by filling in the Search field in the sub-browser's Display record.

10 Cannot Access Display Fields Database
This is a serious error, and will prevent a Display from loading. Check the DOS FILES=### setting, and the File_Handles setting (if on a network).

11 Cannot Access Display Structure Database
See error 10.

12 Load Display: No %s Database
When the Display System tried to load a Display, it either could not find the named database, or could not open it. If the named database is in the Database of Databases and no other errors displayed, see error 10.

13 Load Display: No %s Color

The named color set was not found in the color database. The Display System requires that colors exist for Help, even if no help exists, and for Display. Browsers also should have a distinct Browser Color set. Suggested colors: Help: "HelpPlain", Display = "DEScreen" or "DEScreen2" (reports - no borders), and Browser = "Browser". For menus, VariForm uses "Menu" for Horizontal Menus, and "MenuVertical" otherwise.

14 No Such DisplayList as %s

The specified Display Screen was not found in the Display Database. For Browsers, check the "DEScreen" field in the Display record, otherwise, check whatever routine made the call.

15 Nonexistant field name '%s'

A MetaField in a DisplayField contained a field Name that was neither in the Display's Database, nor in the Virtual Field list At that time.

16 Could not create window for %s

Usually due to low-memory, or impossible values in the named Display Structure.

17 Do Sub Screen: missing screen name or reference

For nested fields, the screen name must be contained in the Lookup Field of the nested field's displayfield record. If not, The name of the field or formula that generates the display name must be in the MetaField area of that record.

18 Invalid field %s

This is another implementation of error 15, and is also displayed by the Search System when it has invalid field names in any of its SearchField records. If this occurs during Browsing, check the browser's search definition. If this occurs during Full-Screen editing, check the lookups displaying or changing on the window at that time.

19 No field to break down

This is a complaint from bitfields, multibit fields, and subscreens that are breaking up strings into integer/character pieces. Check the MetaField entries for the display fields: This is the most common typo in all of VariForm.

20 Bad Target Field Name (%s)

During editing, this means a DisplayField's MetaField does not exist. If the complaint is on a calculated field, change the field to "Uneditable". Otherwise, it is a MetaField misspelling.

21 Could not Build Sub Screen

Usually out of memory, or impossible field size (<1, >299) cause this error. Check this DisplayField's width.

22 Bad Nested Field Type (%d)

Nested fields have a secondary type indicator which is in the 'Decimals' field of the DisplayField record. The value should be between 0 and 4. See the DisplayField section of the Primary Database chapter.

30 Invalid Field Name (%s) in DisplayList

See error 20.

31 Field Name not in DisplayList

See error 20.

32 BCD Formats not Installed

To have access to BCD-encoding schemes, there is a separate set of program modules (.exe, .ovl, or .dll files) that are available separately from Parker Software.

33 Numeric style is only for numeric fields

The database system has listed the current field as a non-numeric type (date, string, et cetera). Either update the database entry for the field, or correct the displayfield record's 'Style' field.

34 Invalid field name (%s) in DisplayList

See error 20.

40 Could not make/create index file (%s)

This error comes indirectly from DOS: either there were not enough file handles, not enough memory, or the drive or filename was not accessible. This requires making changes to either the User, Directory system, or to DOS startup settings.

60 Could not open %s file for %s

Database file access error: see error 40.

61 Load Field: Not enough RAM for all fields

Some RAM can be freed with the "Flush Memory" function.

62 Field %s: entered %d: Calculated %d

The Database System is trying to load a standard (non-union) field, and will use the calculated position, instead of the entered position in the record. Check the entries in the Database Field records for this database.

63 Unknown field %s in %s=>%s index

The Database System has found a bad field name when loading an

index record. Check the named index record for the named database: a field name is probably misspelled.

64 Unknown field type %d in %s=>%s index

The Database System does not recognize the field type number used in the named field in the Database Field record. This often follows an error 73 (see below).

65 Could not access field database

See error 10. Instead of blocking a display from loading, this will prevent a database from loading.

66 Could not access index database

See error 10. Instead of blocking a display from loading, this will prevent a database from loading.

67 Could not access database database

See error 10. Instead of blocking a display from loading, this will prevent a database from loading.

68 NewOpen: cannot find %s in database database

The Database System was given a name (mentioned) that does not exist in the Database of Databases.

69 %s was not Opened

This is a secondary error, explaining the result of a previous error: the named database could not be opened.

70 %s BootStrap Failed

This is a secondary error, caused primarily by either low-memory or too many positioning errors.

71 Tried to close NULL database

Internal severe error: this is a clue that a field is being "Overloaded", such as forcing 200 characters into a 2-byte field. Users should quit to DOS and restart on this message.

72 Tried to close non-chained database (%s)

See error 71.

73 Unknown field type (%d)

The Database System does not recognize the field type number used in a particular field in the Database Field records.

74 Pack: Out of memory

If the system is in "Sensitive" mode, the user should abort the pack, free up some memory, and re-attempt the pack.

80 Enforce Relations: Unindexed unique %s in %s

All fields that are declared 'unique' in the database of database fields should have an index that is comprised of only that field in the corresponding database of indexes.

81 Not enough memory to enforce uniques

Low memory condition: unique record will not be checked, and some add/copy routines may be blocked.

82 Not enough memory to copy child records

Low memory condition: all copy routines on relational databases will be blocked.

83 Kill request (%ld) is out of range

The user, or a procedure, tried to delete a deleted record, or an impossible record number. This is a common occurrence when KillSafe is Off on a slow browser, or when two or more users are co-purging a database. The kill request will be denied, and there are no further consequences from this message.

84 Not enough RAM to Kill records from %s

Low memory condition: Relational system cannot allocate memory to Kill Child Records. If this is happening from a browser, try flushing memory.

85 Not enough RAM to safely process edit

Low memory condition: Relational system cannot rename child records to maintain their connection to their newly renamed parent.

86 Not enough RAM to load relations for %s

Low memory condition: User has less than 4 K free at this point, and relational deletes and kills will kill parent records only.

87 Could not load %s relation for %s

Secondary error: something was wrong with the %s search, which prevented it from loading.

90 HC %d CC %d Curr %d

Disk-Caching message: Hard Cache Block Count, Cache Controllers available count, and Current Cache Controller number.

91 Block %d (%d) has %d ptrs (%d) blank using %d bytes

Disk-Caching message, for viewing the efficiency of a specific Cache Block.

DOS Extended Errors

92 Extended 00: No error

93 Extended 01: Function Number Invalid

Appendix 2: The Error System

Error Messages

94 Extended 02: File Not Found
95 Extended 03: Path Not Found
96 Extended 04: Too many open files (no handles available)
97 Extended 05: Access Denied
98 Extended 06: Invalid Handle
99 Extended 07: Memory Control Block Destroyed
100 Extended 08: Insufficient Memory
101 Extended 09: Memory block address invalid
102 Extended 10: Environment Invalid (>32K long)
103 Extended 11: Format Invalid
104 Extended 12: Access Code Invalid
105 Extended 13: Data Invalid
106 Extended 14: Reserved
107 Extended 15: Invalid Drive
108 Extended 16: Attempted to Remove Current Directory
109 Extended 17: Not same device
110 Extended 18: No More Files
111 Extended 19: Disk Write Protected
112 Extended 20: Unknown Unit
113 Extended 21: Drive Not Ready
114 Extended 22: Unknown Command
115 Extended 23: CRC Data Error
116 Extended 24: Bad Request Structure Length
117 Extended 25: Seek Error
118 Extended 26: Unknown Media Type (non-dos disk)
119 Extended 27: Sector Not Found
120 Extended 28: Printer Out of Paper
121 Extended 29: Write Fault
122 Extended 30: Read Fault
123 Extended 31: General Failure
124 Extended 32: Sharing Violation
125 Extended 33: Lock Violation
126 Extended 34: Disk Change Invalid
127 Extended 35: FCB Unavailable
128 Extended 36: Sharing Buffer Overflow
129 Extended 37: Reserved
130 Extended 38: Cannot complete file operation
131 Extended 39: Reserved
132 Extended 40: Reserved
133 Extended 41: Reserved
134 Extended 42: Reserved
135 Extended 43: Reserved
136 Extended 44: Reserved
137 Extended 45: Reserved
138 Extended 46: Reserved
139 Extended 47: Reserved
140 Extended 48: Reserved
141 Extended 49: Reserved
142 Extended 50: Network Request Not Supported

143 Extended 51: Remote Computer Not Listening
144 Extended 52: Duplicate Name on Network
145 Extended 53: Network Name Not Found
146 Extended 54: Network Busy
147 Extended 55: Network Device No Longer Exists
148 Extended 56: Network BIOS Command Limit Exceeded
149 Extended 57: Network Adapter Hardware Error
150 Extended 58: Incorrect Response from Network
151 Extended 59: Unexpected Network Error
152 Extended 60: Incompatible Remote Adapter
153 Extended 61: Print Queue Full
154 Extended 62: Queue Not Full
155 Extended 63: Not Enough Space to Print File
156 Extended 64: Network Name was Deleted
157 Extended 65: Network: Access Denied
158 Extended 66: Network Device Type Incorrect
159 Extended 67: Network Name Not Found
160 Extended 68: Network Name Limit Exceeded
161 Extended 69: Network BIOS Session Limit Exceeded
162 Extended 70: Temporarily Paused
163 Extended 71: Network Request Not Accepted
164 Extended 72: Network print/disk Redirection Paused
165 Extended 73: Invalid Network Version
166 Extended 74: Account Expired
167 Extended 75: Password Expired
168 Extended 76: Login Attempt Invalid at this time
169 Extended 77: Disk Limit Exceeded on Network Node
170 Extended 78: Not Logged In to Network Node
171 Extended 79: Reserved
172 Extended 80: File Exists
173 Extended 81: Reserved
174 Extended 82: Cannot Make Directory
175 Extended 83: Fail on Int 24h
176 Extended 84: Too Many Redirections
177 Extended 85: Duplicate Redirection
178 Extended 86: Invalid Password
179 Extended 87: Invalid Parameter
180 Extended 88: Network Write Fault
181 Extended 89: Function Not Supported on Network
182 Extended 90: Required System Component Not Installed

DOS Extended Error Class

183 Err Class 00: Out of resource (storage or I/O Channels)
184 Err Class 01: Temporary Situation (file/record lock)
185 Err Class 02: Authorization (Access Denied)
186 Err Class 03: Internal (System Software Bug)
187 Err Class 04: Hardware Failure

Appendix 2: The Error System

Error Messages

188 Err Class 05: System Failure (Configuration Missing or Incorrect)
189 Err Class 06: Application Program Error
190 Err Class 07: Not Found
191 Err Class 08: Bad Format
192 Err Class 09: Locked
193 Err Class 10: Media Error
194 Err Class 11: Already Exists
195 Err Class 12: Unknown

DOS Extended Error Locus

196 Err Locus 0: Unknown or Not Appropriate
197 Err Locus 1: Block Device (Disk Error)
198 Err Locus 2: Network Related
199 Err Locus 3: Serial Device (Timeout)
200 Err Locus 4: Memory Related

DOS Extended Error Suggested Action

201 Err Action 0: Retry
202 Err Action 1: Delayed Retry
203 Err Action 2: User Reentry Prompt Coming
204 Err Action 3: Abort after Cleanup
205 Err Action 4: Immediate Abort
206 Err Action 5: Ignore
207 Err Action 6: Retry After User Intervention

208 Operating System Asked for Shutdown
Dos Extended Error Action 3 or 4. Usually after too many errors.

209 Global Position Wanted %ld, but got %ld
VariForm could not get to a needed position in the file. If the position wanted is less than zero or very, very large, this may indicate a corrupt index.

210 Total Local Error Count Exceeds Maximum Limit
If too many of any 1 error type occur, VariForm will attempt to shut down normally, to prevent index corruption.

211 Read wanted %d bytes, but got %d
VariForm could not read enough bytes to fulfill a request. This indicates either a corrupted index, or a damaged database file. Indexes can be rebuilt (See first page), but databases must be packed.

212 Write wanted %d bytes, but got %d
See error 211.

Appendix 2: The Error System

Error Messages

DOS Standard Errors

213 Std %d (dos %d) : Error 0
214 Std %d (dos %d) : Invalid Function Number
215 Std %d (dos %d) : No such file or directory
216 Std %d (dos %d) : Path not found
217 Std %d (dos %d) : Too many open files
218 Std %d (dos %d) : Permission denied
219 Std %d (dos %d) : Bad file number
220 Std %d (dos %d) : Memory arena trashed
221 Std %d (dos %d) : Not enough memory
222 Std %d (dos %d) : Invalid memory block address
223 Std %d (dos %d) : Invalid Environment
224 Std %d (dos %d) : Invalid format
225 Std %d (dos %d) : Invalid access code
226 Std %d (dos %d) : Invalid data
227 Std %d (dos %d) : Reserved (14)
228 Std %d (dos %d) : No Such device
229 Std %d (dos %d) : Attempted to remove current directory
230 Std %d (dos %d) : Not same device
231 Std %d (dos %d) : No more files
232 Std %d (dos %d) : Invalid argument
233 Std %d (dos %d) : Arg list too big
234 Std %d (dos %d) : Exec format error
235 Std %d (dos %d) : Cross-device link
236 Std %d (dos %d) : Reserved (23)
237 Std %d (dos %d) : Reserved (24)
238 Std %d (dos %d) : Reserved (25)
239 Std %d (dos %d) : Reserved (26)
240 Std %d (dos %d) : Reserved (27)
241 Std %d (dos %d) : Reserved (28)
242 Std %d (dos %d) : Reserved (29)
243 Std %d (dos %d) : Reserved (30)
244 Std %d (dos %d) : Reserved (31)
245 Std %d (dos %d) : Reserved (32)
246 Std %d (dos %d) : Math argument
247 Std %d (dos %d) : Result too large
248 Std %d (dos %d) : File already exists

249 No %s Function Keys

This is part of the system that displays function keys. The text should be "Global" (Display-wide keys), or "Local" (Field-specific keys).

250 %s KeySet is %s

This is part of the system that displays function keys. The first text will be as in message 249. The second text will contain the Function Key Group Name, as it appears in the database of Function Keys.

Appendix 2: The Error System

Error Messages

251 %sF%d => %s

This is part of the system that displays function keys. The first text will be blank, "Shift", "Control", or "Alt". The number will be the function key number, and the final text is the CommandLine that the Function Key Performs if pressed.

252 No Such Function Key Available at this time

The user pressed a function key that is not part of the current Local or Global Function key sets.

260 No such menu display as %s

The VariForm Menu System could not find the display named in the error message. Either the name was misspelled, or the Display is not a DisplayType 5 (menu). Check the Display Database.

270 No entries in internal procedure list

Either the individual procedure-lines are inaccessible (wrong name, out of memory, damaged Procedure database), or all of the lines have been commented out. Check Procedures database.

271 No such procedure as '%s'

There are no records in the Procedures database containing the displayed procedure name.

272 Unmatched terminator (}) at line %ld level %d

All curly braces must be paired within a procedure. Every "if() {" must have a "}" to close it off. Check the procedures database.

273 No active lines in '%s' procedure

Either the individual procedure-lines are inaccessible (wrong name, out of memory, damaged Procedure database), or all of the lines have been commented out. Check Procedures database.

274 Procedure: Invalid action %d

A variable is touching a left-parenthesis. The Procedure System interprets "Textstuff()", as a procedure called "TextStuff", with parameters inside the parenthesis. In general, unless calling a procedure, there should always be a space to the left of a left parenthesis. The only exceptions are at the beginning of a line, immediately following a math or logical operator, or immediately following another left parenthesis.

280 Kill Report given non-loaded report to kill

Internal: a report was improperly loaded, or the user Flushed Memory while the report was running.

281 Kill Report found RS stacking error

Internal: a report was improperly loaded, or a variable assignment within the subreport is causing memory corruption. Try simplifying the report.

282 Page/Block break

During a report to screen, this pauses the report where the primary report's "page" ends. During other reports, this indicates the user has pressed <Escape>, probably several times.

283 Cannot load %s report

Secondary error: either the named report does not exist, or for other reasons, the report could not load into memory.

284 Could not open print file (%s)

The report can display, but cannot be extracted to the output file. This may be bad entry on Generic Report systems, or an invalid filename in the User Parameters / Printer Information field.

290 Fast Stop: No Record Test Cluster Information

The search system could not find valid field information in any type 6 SearchField records associated with the current search.

291 Search: Out of memory

292 Index problem preventing search from advancing

LoopSearch: This warning means the search has entered an infinite loop. Either damaged indexes or an improper use of the LoopSearch command has cause the search to find the same record twice in a row. If the repetitive find is desirable, the user, search, or procedure should be using MultiSearch. LoopSearch is only for actions that delete records, or edit and move them along the search index.

293 Cannot access Search Database

See error 10. Instead of blocking a display from loading, this will prevent a Search from loading.

294 Cannot access child of Search (%s)

The search system could not open the database named in the Search record's "Child Database" field.

295 Cannot access SearchField Database

See error 10. Instead of blocking a display from loading, this will prevent a Search from loading.

296 No such search exists as '%s'

A Lookup, Search, DoSearch, LoopSearch, or MultiSearch was given a name that does not exist in the Search Database.

Appendix 2: The Error System

Error Messages

300 Out of system space

Low memory condition: not enough RAM to load the next Virtual Field.

301 Out of Variable Space

Not enough space in the pre-allocated block for the next Virtual Field. Increase the "Variable Space" field in the User Settings record and re-start VariForm.

310 No detail information for this '%s'

Any display system: there is a Display Record, but no DisplayField Records matching the named Display group.

311 Help for '%s' not found

There is no help record by the displayed name. If the user called "Help(something)", there is no help group "something". If the user pressed the SystemHelp() key, the Field or Display Element's help, whichever is non-blank, is the help that is called. If both elements' help is blank, VariForm will try to find help on "Data Entry", "Browser", or "Menu System", based on which type of window the user was in at the time.

312 At most recent help

User is pressing "Next Help" key, and has reached the most recently requested help.

313 At oldest stored help

User is pressing "Prev Help" key, and has reached the oldest requested help in memory. VariForm stores 25 help calls in memory for user convenience.

314 At earliest used help

This is the same as error message 313, except that all Help historical memory hasn't been consumed yet.

320 FindNext index damage in %s=>%s

An Index has been corrupted, and should be rebuilt/packed.

321 FindPrev index damage in %s=>%s

An Index has been corrupted, and should be rebuilt/packed.

322 Error adding key to index file

An Index has been corrupted, and should be rebuilt/packed.

323 Cannot make new index file (%s)

Typically displayed after dos errors. Quit VariForm, correct the DOS problem (logon to the network, or run CHKDSK /f), and restart VariForm to see if the problem disappears.

324 Index Reopen failure (%s)
This error occurs on networks with loose cables, networks when the disk-host reboots, or standalone systems with bad File Allocation Tables. Right after an index file was rebuilt, it was closed, and cannot be reopened; this means a device has changed. This error should be repaired, and then the database that gave this error should be Packed.

325 Out of memory... Indexes reduced
Low memory condition: VariForm will keep less of the indexes in memory so that the system can still be accessed. This message can be eliminated by changing Index Buffering Parameters in the Database of Databases: Reduce buffer count (minimum is 2), Reduce buffer size (this will require rebuilding the indexes, minimum size is 128 bytes, or MaxIndex Length + 25, whichever is greater), and turn "Off" unused AutoLoad Databases.

326 Indexes have not been loaded
Severe low-memory condition: VariForm dropped all index references to try and access the database. Do not edit a database if this message appears! See Appendix 7 (page) 'Making the Most of Memory".

327 Index pulled from %s. CLOSE THIS DATABASE!
Severe low-memory condition: VariForm dropped all index references to try and access the database. Do not edit a database if this message appears! See Appendix 7 (page) 'Making the Most of Memory".

330 Cannot open '%s' datafile
VariForm could not open the main datafile for a database. This is most often caused by FILES and File_Handles being set too low, but can also result from an invalid or duplicate Data filename in the Database of Database Record.

340 Out of memory. Needed %d
Low-memory condition. Try bringing up the Free Memory window to see how much memory is available.

350 Kill Var: Nonexistant Variable
A procedure has a typo in a KillVar() statement. Either comment or delete the line, or correct the spelling in the Procedures Database. Note: KillVar() or Remove() should only be used on Variables created with the Assign() statement. Variables created with the LoadVirtuals() statement should only be removed by the KillVirtuals() statement.

351 '%s' can only be removed at the level where is was declared

A sub-procedure tried to remove a variable assigned by a parent procedure. This can also appear when duplicate Variable names are used.

352 Kill Table: Nonexistant Table

If this displays during a Flush Memory call, Quit VariForm Immediately. This is a 100% positive sign that memory is corrupted. Try flushing at various points in VariForm to try and recreate the error. Most likely causes: infinite recursion (a procedure or search calling itself), or overloading Virtual Fields.

353 Action #%d has no group route reference

Beta-test copies of VariForm only: an action has been listed in the program, but not implemented in the overlays. Check to make sure that Core.exe, WCore.exe, and WCore.ovl all contain the same date stamp: times should be within a 5-minute period.

Math Errors

To prevent a generally simple error from blossoming out of control, VariForm traps math errors, displays the problem, and returns a '0.0' to the system call that generated the error.

354 Math Error 1: Invalid Domain - no calculation possible: %s

Multiplication or Power functions were given bad values.

355 Math Error 2: Singularity - parameter is 0: %s

Log, root, division were given 0 values.

356 Math Error 3: Overflow - parameter too large to calculate: %s

64 bit math allows 15 bit precision : +/- 8 * 10e +/-32.
Number was greater than +/- 8 * 10e +32. (Nine-tillions):
abs(n) > 800,000,000,000,000,000,000,000,000,000,000

357 Math Error 4: Underflow - parameter too small to calculate: %s

64 bit math allows 15 bit precision : +/- 8 * 10e +/-32.
Number was less than +/- 1 * 10e -32. (Nine-tillionths):
abs(n) < 0.000,000,000,000,000,000,000,000,000,000,001

358 Math Error 5: Total Loss of significance - parameter exceeds precision

This will appear when parameters exceed the square root of the precision (see 356, 357 above).

359 Math Error 6: Partial Loss of significance - parameter approaches prec

This will appear when parameters exceed the cube root of the precision (see 356, 357 above).

Appendix 2: The Error System

Error Messages

370 Can't Compare Entries of Type %d

When VariForm Procedures perform comparison operations, VariForm looks at the 'Type' of the first (left-hand) variable, to determine how to perform the comparison. A rule of thumb: place explicit variables or the right-hand side of a comparison, and this error should never occur.

371 '%s' is a System Level Function and can not be called directly

Do not call the named function from the CommandLine Interface.

372 The '%s' function is not available in %s mode

Certain functions are mode-specific, such as Redraw(), which is for Data-Entry Screens, or ChooseIndex(), for Browsers.

373 Action %d %s: R=%d G=%d

Beta Testers on new functions: This is the error display for mode violations that will display both Reqired and Got Currently access mode values. (Sensitive mode 2, Beta Copies Only).

374 Field Names, such as '%s' must be declared before use

A procedure or command line attempted to assign a value with an equal sign to a nonexistant Field or to an implicit constant:

"4 = >ChildField" is an example of this, where:

A: 2 equal signs were needed for logical expressions
B: The implicit constant should be on the right

375 Cannot reconcile VarType %d

Either the Field Type in the Virtual Database, the Assign() Statement, or the Database of Fields has an invalid number, outside the range of 0-8, or there is some memory-corruption occurring.

376 Cannot Add VarType %d elements

Addition is only valid for : ASCIIZ strings, Numerics, Dates and Times. Other fieldtypes, such as BCD's, require the advanced mathpak.

377 Cannot Subtract VarType %d elements

Subtraction is only valid for Numerics and Dates. Other fieldtypes, such as BCD's, require the advanced mathpak.

378 Cannot Load '%s' menu

Secondary error: for previously stated reasons, the named menu could not be loaded. This is usually system memory, unloadable display (bad colors, no elements), or non-existant display (display does not exist, or is not DisplayType 5).

379 No such field as %s in child db (%s)

The Search and Display System both use this error when a field name in a MetaField line is not a member of the Child Database, nor one of the currently loaded Virtual Fields.

380 Search '%s' is infinite

See error #292.

381 MapSearch had insufficient data to run

MapSearch either could not load the secondary (Parent/receiver) database, or could not load the actual Search Record.

382 Consolidate 0: Action %s not installed

A procedure or CommandLine call referenced an action not found in the overlay libraries. See error #353.

390 Index was NULL

A search or browser attempted to use an index that was not loaded, or is no longer part of the database. Generally, a low-memory condition set up this problem.

391 %s: %ld total %ld erased %ld active

Message that displays for Info() requests if no records are damaged.

392 %s: %ld total %ld erased %ld damaged %ld active

Message that displays for Info() requests if records are damaged. This should instruct the standalone user to Pack the database immediately. The network user should retry in 30 seconds. If there are still damaged records, have all other users log off and pack the database.

393 '%s' PickList has no Search Structure

When a user selects an item from a PickList, VariForm needs a Search Structure with a Type 10/11 command to tell it what to do with the selected item. Check the PickList's display record and make sure there is a valid Search in the SearchName Field.

394 '%s' BrowseList has no Search Structure

When a user selects an item from a BrowseList, VariForm needs a Search Structure with a Type 10/11 command to tell it what to do with the selected item. Check the BrowseList's display record and make sure there is a valid Search in the SearchName Field.

395 Cannot Choose indexes on a Limited (sub) Browser

Because PickLists, BrowseLists and Sub-Browsers use Search Structures, changing the index of the search could be at best a slowdown. If the user desparately needs to have more than one

index available, delete the Index Name from the associated Search Record.

396 Cannot Load '%s' requestor

Procedures and Primary Reports can call requestors, which act exactly like Data Entry Screens, except that they operate on Virtual Fields. Either the named requestor does not exist, or it is not a DisplayType 1 record. Check the Display Record Database.

397 Cannot Load '%s' browser

If this message appears as a first error, then either the named Browser Display Record does not exist, or it is not a DisplayType 1 or Type 3 record. Check the Display Record Database. If this message appears secondarily, ignore it.

398 Consolidate 1: Action %s not installed

A procedure or CommandLine call referenced an action not found in the overlay libraries. See error #353.

400 MemWindow must be 0=off, 1=on or 2=toggle

The MemWindow() command requires a parameter of either 0, 1, or 2, which close the window, open/reopen the window, or change the window's state (open if closed, and vice versa).

401 No KillSafe Variable

There must be a Virtual Field in the "Main" Group that is an Integer Type, called KillSafe. This variable determines if the Browsers will prompt the user "Delete This Record?" before executing a KillRec() command. VariForm default is 1.

402 Kill function DOES NOT REQUIRE confirmation

This displays when KillSafe is toggled off (set to 0).

403 Kill function REQUIRES confirmation

This displays when KillSafe is toggled on (set to 1).

404 Consolidate 2: Action %d not installed

A procedure or CommandLine call referenced an action not found in the overlay libraries. See error #353.

410 No DateFormat Variable

There must be a Virtual Field in the "Main" Group that is an Integer Type, called DateFormat. This variable determines the format in which Date Fields are stored and Displayed in the Virtual System. VariForm default is 560: Year\Month\Day, sorted order, Padded to maximum size with zeros (4\2\2).

Appendix 2: The Error System

Error Messages

411 Cannot use format %d as a date

Only date and character fields can be interpreted as dates. If this happens when Virtual Fields are loading, and one of them is a date type, remove the initializer, or change the date separators to back-slashes (\).

412 Could not erase original target file (%s)

Move command: copy was made, but source file was not erased.

413 File Copy: '%s' does not exist

Copy command: source file does not exist.

414 Consolidate 3: Action %s not installed

A procedure or CommandLine call referenced an action not found in the overlay libraries. See error #353.

Appendix 3: Pitfalls & Common Problems

This Appendix is a supplement to Appendix 2: the Error System. Look there first: in addition to error messages, there are explanations of errors, and some suggested remedial actions to take in response. This appendix is intended for when the Error System either doesn't display a problem, or when the remedial actions in Appendix 2 didn't solve the problem.

VariForm will Not Start

Enough memory - Excluding AutoLoad requirements, VariForm must have 400,000 bytes available at runtime: Largest Executable should be at least 400,000, according to CHKDSK or MEM commands at the DOS level.

System directory ok - minimum files for VariForm to run: CORE.EXE and CORE.OVL or WCORE.EXE, or VARIFORM.EXE. Data files: USERDATA.DAT, DATABASE.DAT, DBFIELD.DAT, DBINDEX, COLORS.DAT, VFIELDS.DAT, DESTRUCT.DAT, DEFIELD.DAT, and something.SCR.

VariForm Keeps Shutting Down

QUIT variable - there should be no variable or database field named 'quit', since this is the name of an internal function that 'shuts down' VariForm.

Not Enough memory to display messages - if Error number 340 keeps appearing without further explanation, then there is not enough memory for any other function.

Memory window flickers, but the numbers don't change

OverLoading - Some databases, especially larger ones, are better off if they are loaded when VariForm is started. Change the 'autoload' bit in the database of database characteristics field for this record to 'Yes'. quit and restart VariForm.

Huge Indexes - See Appendix 1 (page) on resizing indexes.

Indexes Keep getting Damaged

Share/File limitations - the number of accessible files, or the amount of memory to reserve with the Share command may need to be increased. Make the appropriate changes in autoexec.bat or config.sys, and reboot the system.

Disk Damage - run CHKDSK /F. If an index file is named, delete it; if a database file is named, pack the database from VariForm.

Appendix 3: Pitfalls & Common Problems

Records Disappeared during a Pack

Other users online are accessing the database. Have them log off, drop to dos, delete the database.dat and .0* files, and rename the database.bak to database.dat. Re-accessing the database from VariForm will force the index to automatically regenerate.

Out of disk space. Make more disk space, then follow previous suggestion.

Structure Change : check Database of Fields for changes or index damage.

Changed Encryption Status/Password without rebuild/retranslate. Return encryption status and Key to previous settings in the database of databases, make of copy of the database record with the new settings, remembering to change the file mask, as well, and transfer records from the 'old', original database to the new one. Rename the databases after the record transfer is complete. If the database is a system or autoload database, users will have to quit and re-start VariForm for the changes to take affect. they should also make sure that they are accessing the newest copyof the database of databases, fields, and indexes.

Files won't Copy

Other users online are accessing the primary file to be copied: have them log off.

System crash - reboot and retry.

Bad FAT - critical DOS problem, try a CHKDSK /f command, or a disk utilities / repair program.

Procedure won't Stop Running

Infinite loops
Searching a bad index

Can't access a Data Entry Field

Uneditable field
Security Violation

Report Hangs in a sub-Report

Singular sub-report on a singular main report

Appendix 4: Miscellaneous Tables

Ascii Codes

All characters, whether typable or communications related, are represented by a set of numbers, known as ASCII characters. The American Standard Code for Information Interchange represents all characters of the English language. This 'symbol set' has been augmented by IBM, to provide the 'basic PC character set', which covers all 256 combinations of 1 byte.

Each non-alphabetic, non-numeric ASCII code has a name, which is included in this list. The 'non typable' characters can be sent to VariForm through either a CHAR() command in the Command system, or by holding down the <Alt> key and pressing the decimal value of the ASCII code on the numeric keypad on any data entry field.

Code	Char.	Name or Meaning	Code	Char.	Name or Meaning
0	NUL	End of line	32	SPACE	Spacebar Character
1	☺ SOH	Start/Header	33	!	Exclamation Point
2	☻ STX	Start/Text	34	"	Quotation Mark
3	♥ ETX	End/Text	35	#	Number/Pound Sign
4	♦ EOT	End/Transmission	36	$	Dollar Sign
5	♣ ENQ	Enquiry	37	%	Percent Sign
6	♠ ACK	Acknowledge	38	&	Ampersand
7	● BEL	Bell: tone	39	'	Apostrophe
8	◘ BS	Backspace	40	(	Left Parenthesis
9	○ HT	Horizontal Tab	41	)	Right Parenthesis
10	◙ LF	Line Feed	42	*	Asterisk
11	♂ VT	Vertical Tab	43	+	Plus Sign
12	♀ FF	Form Feed	44	,	Comma
13	♪ CR	Carriage Return	45	-	Minus Sign/Hyphen
14	♫ SO	Shift Out	46	.	Period
15	☼ SI	Shift In	47	/	Slash
16	► DLE	Data Link Escape	48	0	
17	◄ DC1	Data Ctl 1:XON	49	1	
18	↕ DC2	Data Ctl 2	50	2	
19	‼ DC3	Data Ctl 3:XOFF	51	3	
20	¶ DC4	Data Ctl 4	52	4	
21	§ NAK	Negative ACK	53	5	
22	▬ SYN	Synchronous Idle	54	6	
23	↨ ETB	End/Trans. Block	55	7	
24	↑ CAN	Cancel	56	8	
25	↓ EM	End of Medium	57	9	
26	→ SUB	Substitute	58	:	Colon
27	← ESC	Escape [begin]	59	;	Semicolon
28	∟ FS	File Separator	60	<	Left Angle Bracket
29	↔ GS	Group Separator	61	=	Equal Sign

30	▲RS	Record Separator	62	>	Right Angle Brack.
31	▼US	Unit Separator	63	?	Question Mark

(Table Continued on the next page)

Appendix 4: Miscellaneous Tables

ASCII Codes

Code	Char.	Name or Meaning	Code	Char.	Name or Meaning
64	@	At sign	96	`	Grave Accent
65	A		97	a	
66	B		98	b	
67	C		99	c	
68	D		100	d	
69	E		101	e	
70	F		102	f	
71	G		103	g	
72	H		104	h	
73	I		105	i	
74	J		106	j	
75	K		107	k	
76	L		108	l	
77	M		109	m	
78	N		110	n	
79	O		111	o	
80	P		112	p	
81	Q		113	q	
82	R		114	r	
83	S		115	s	
84	T		116	t	
85	U		117	u	
86	V		118	v	
87	W		119	w	
88	X		120	x	
89	Y		121	y	
90	Z		122	z	
91	[	Lft. Sq. Brack.	123	{	Lft. Curly Brace
92	\	Backslash	124	|	Vertical Line/Cat.
93	]	Rt. Sq. Bracket	125	}	Rt. Curly Brace
94	^	Circumflex	126	~	Tilde
95	_	Underscore	127	⌂ DEL	Delete / No Op.

(Table Continued on the next page)

Appendix 4: Miscellaneous Tables

ASCII Codes

Code	Char.	Name or Meaning	Code	Char.	Name or Meaning
128	Ç		160	á	
129	ü		161	í	
130	é		162	ó	
131	â		163	ú	
132	ä IND	Index	164	ñ	
133	à NEL	Next line	165	Ñ	
134	å SSA	Start SelectArea	166	ª	
135	ç ESA	End Select Area	167	º	
136	ê HTS	Horiz. Tab Set	168	¿	
137	ë HTJ	H.T. w/justif.	169	⌐	
138	è VTS	Vert. Tab Stop	170	¬	
139	ï PLD	Partial Line Dn	171	½	
140	î PLU	Partial Line Up	172	¼	
141	ì RI	Reverse Index	173	¡	
142	Ä SS2	Single Shift 2	174	«	
143	Å SS3	Single Shift 3	175	»	
144	É DCS	Device Ctl Str.	176	▒	
145	æ PU1	Private Use 1	177	▓	
146	Æ PU2	Private Use 2	178	▓	
147	ô STS	SetTerminalState	179	┤	
148	ö CCH	Cancel Character	180	┤	
149	ò MW	Message Writing	181	╡	
150	û SPA	StartProtectArea	182	╢	
151	ù EPA	End Protect Area	183	╖	
152	ij		184	╕	
153	Ö		185	╣	
154	Ü		186	╢	
155	¢ CSI	Ctrl SequenceInt	187	╗	
156	£ ST	String Terminate	188	╝	
157	¥ OSC	Oper.System Cmd.	189	╜	
158	₧ PM	Privacy Message	190	╛	
159	ƒ APC	App.Prog. Cmd.	191	┐	

(Table Continued on the next page)

Appendix 4: Miscellaneous Tables

ASCII Codes

Code	Char.	Name or Meaning	Code	Char.	Name or Meaning
192	└		224	α	
193	┴		225	ß	
194	┬		226	Γ	
195	├		227	π	
196	─		228	Σ	
197	┼		229	σ	
198	╞		230	µ	
199	╟		231	τ	
200	╚		232	Φ	
201	╔		233	Θ	
202	╩		234	Ω	
203	╦		235	δ	
204	╠		236	∞	
205	═		237	φ	
206	╬		238	ε	
207	╧		239	∩	
208	╨		240	≡	
209	╤		241	±	
210	╥		242	≥	
211	╙		243	≤	
212	╘		244	⌠	
213	╒		245	⌡	
214	╓		246	÷	
215	╫		247	≈	
216	╪		248	°	
217	┘		249	∙	
218	┌		250	·	
219	█		251	√	
220	▄		252	ⁿ	
221	▌		253	²	
222	▐		254	■	
223	▀		255		

(Table Continued on the next page)

Appendix 4: Miscellaneous Tables

Bit Fields

Many places in this book, "Bit Fields", or "Fields of Bit Flags" are mentioned. When presented with a series of 'Yes/No' conditions, it is often more efficient to store the results in one field than in several. Bit fields manipulation can be performed on integer fields, Long Ints, and even ASCIIZ character strings. Integers and Long Ints are the most common, however.

Example: 6 'Yes/No' questions could consume 6 integers, at 2 bytes each, or 6 1-byte character strings, at 1 byte each, or be combined into 1 2-byte bit field. The savings in data and record space can be enormous.

Every byte consists of 8 bits, so a 2-byte Integer field can act as 16 separate bit fields, if the Data Entry System is set up properly to do so. The 16 bits are numbered 0-15 (inclusive), from least to most significant. If there are no subfields set up, bit fields can still be changed by a user with a little basic math: every bit field is a power of 2.

Bit Field Value Table

Bit	Numeric	Bit	Numeric	Bit	Numeric	Bit	Numeric Value
0	1	8	256	16	65 536	24	16 777 216
1	2	9	512	17	131 072	25	33 554 432
2	4	10	1 024	18	262 144	26	67 108 864
3	8	11	2 048	19	524 288	27	134 217 728
4	16	12	4 096	20	1 048 576	28	268 435 456
5	32	13	8 192	21	2 097 152	29	536 870 912
6	64	14	16 384	22	4 194 304	30	1 073 741 824
7	128	15	32 768	23	8 388 608	31	2 147 483 648

So, if a user needed bits 5, 3, and 1 "On" or "Yes" in an Integer Field that did not have Yes/No Bit Field entries, the value 42 (32 + 8 + 2) would suffice.

Appendix 4: Miscellaneous Tables

Number Bases

For those who did not have 'the new math' in school, or who need a quick refresher, this section is dedicated to number bases. Conventional counting is done in the decimal, or base 10 system, where 1,234 means $1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0$.

Computers prefer to count in powers of 2, instead of powers of 10. This is due to the fact that computers can only count 'on' and 'off' circuits, instead of having 10 fingers to count with. The reason for the base systems of numbering in computing is to determine a number without always having to convert back and forth to base 10.

There are three bases that are most commonly used in computer environments: base 2, or binary; base 8, or octal, and base 16, also called hexadecimal. The rules are simple: a base is a numbering system with the base value (2,8,10, or 16) as the power. For example, the number 23 in base 10 would be:

17 in base 16 (1 x 16 + 7)
27 in base 8 (2 x 8 + 7)
10111 in base 2 (16 + 4 + 2 + 1)

Binary seems to make sense to most people: it is the numbering system for Boolean Algebra, Logic Systems, and represents all combinations of on and off in a given set of wires. To put this in perspective, personal computers consist of 'bytes', which are 8 'bits' long: each byte can be completely represented by an 8-digit binary number.

The Octal system became popular very early in the computer age, when many computer processors were working with 3-bit systems. 3 bits, called an 'octet', represents all values from 0-7, inclusive, so base 8 became the numbering system of choice.

With Intel, Motorola, and the advent of the 8 and 16 bit processors, it made sense to start counting four bits at a time. Since 4 bits represents 16 possible values, hexadecimal, or base 16 came into use. Technically, an 8-bit byte consists of 2 4-bit 'nybbles' (pronounced nibbles).

A pleasant side-effect of using base 16 is that it can be easily converted to and from binary easily. Since bytes and words are multiples of 8-bits, they are represented by multiples of 2 hexadecimal digits.

Appendix 4: Miscellaneous Tables

Number Bases

The following table illustrates the values of the different numbering systems:

10	16	8	2	10	16	8	2	10	16	8	2
0	0	0	0	33	21	41	100001	66	42	102	1000010
1	1	1	1	34	22	42	100010	67	43	103	1000011
2	2	2	10	35	23	43	100011	68	44	104	1000100
3	3	3	11	36	24	44	100100	69	45	105	1000101
4	4	4	100	37	25	45	100101	70	46	106	1000110
5	5	5	101	38	26	46	100110	71	47	107	1000111
6	6	6	110	39	27	47	100111	72	48	110	1001000
7	7	7	111	40	28	50	101000	73	49	111	1001001
8	8	10	1000	41	29	51	101001	74	4A	112	1001010
9	9	11	1001	42	2A	52	101010	75	4B	113	1001011
10	A	12	1010	43	2B	53	101011	76	4C	114	1001100
11	B	13	1011	44	2C	54	101100	77	4D	115	1001101
12	C	14	1100	45	2D	55	101101	78	4E	116	1001110
13	D	15	1101	46	2E	56	101110	79	4F	117	1001111
14	E	16	1110	47	2F	57	101111	80	50	120	1010000
15	F	17	1111	48	30	60	110000	81	51	121	1010001
16	10	20	10000	49	31	61	110001	82	52	122	1010010
17	11	21	10001	50	32	62	110010	83	53	123	1010011
18	12	22	10010	51	33	63	110011	84	54	124	1010100
19	13	23	10011	52	34	64	110100	85	55	125	1010101
20	14	24	10100	53	35	65	110101	86	56	126	1010110
21	15	25	10101	54	36	66	110110	87	57	127	1010111
22	16	26	10110	55	37	67	110111	88	58	130	1011000
23	17	27	10111	56	38	70	111000	89	59	131	1011001
24	18	30	11000	57	39	71	111001	90	5A	132	1011010
25	19	31	11001	58	3A	72	111010	91	5B	133	1011011
26	1A	32	11010	59	3B	73	111011	92	5C	134	1011100
27	1B	33	11011	60	3C	74	111100	93	5D	135	1011101
28	1C	34	11100	61	3D	75	111101	94	5E	136	1011110
29	1D	35	11101	62	3E	76	111110	95	5F	137	1011111
30	1E	36	11110	63	3F	77	111111	96	60	140	1100000
31	1F	37	11111	64	40	100	1000000	97	61	141	1100001
32	20	40	100000	65	41	101	1000001	98	62	142	1100010

Appendix 4: Miscellaneous Tables

Color Codes

There are look-up fields that allow the VariForm user to select up to 128 different color combinations within the Color Scheme Database, but the basic principle of the color code should still be understood.

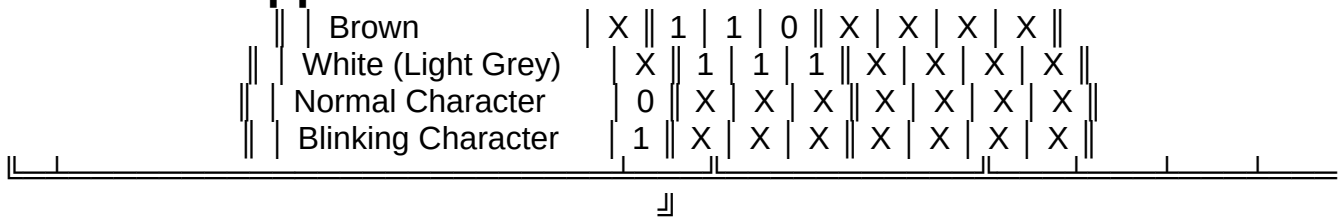
Color codes are based on a compound RGBI scheme. In English, that means that the color code is actually comprised of two codes (foreground and background), and those codes are comprised of four codes (Red, Green, Blue, Intensity).

The BackGround Intensity bit is used for blinking characters, instead of an intense background setting. Observe the following table:

Color Implication	Bit Settings for color screen							
	Blink	R	G	B	I	R	G	B
Foreground Color	7	6	5	4	3	2	1	0
Black	X	X	X	X	0	0	0	0
Blue	X	X	X	X	0	0	0	1
Green	X	X	X	X	0	0	1	0
Cyan	X	X	X	X	0	0	1	1
Red	X	X	X	X	0	1	0	0
Magenta	X	X	X	X	0	1	0	1
Brown	X	X	X	X	0	1	1	0
Light Grey	X	X	X	X	0	1	1	1
Dark Grey (Br. Black)	X	X	X	X	1	0	0	0
Bright Blue	X	X	X	X	1	0	0	1
Bright Green	X	X	X	X	1	0	1	0
Bright Cyan	X	X	X	X	1	0	1	1
Bright Red	X	X	X	X	1	1	0	0
Bright Magenta	X	X	X	X	1	1	0	1
Yellow (Bright Brown)	X	X	X	X	1	1	1	0
White (Bright Lt. Gr.)	X	X	X	X	1	1	1	1

Background Color	7	6	5	4	3	2	1	0
Black	X	0	0	0	X	X	X	X
Blue	X	0	0	1	X	X	X	X
Green	X	0	1	0	X	X	X	X
Cyan	X	0	1	1	X	X	X	X
Red	X	1	0	0	X	X	X	X
Magenta	X	1	0	1	X	X	X	X

Appendix 4: Miscellaneous Tables



X: don't care / not applicable
0: Clear / off
1: Set / on

Appendix 4: Miscellaneous Tables

Monochrome Codes

There are look-up fields that allow the VariForm user to select up to 128 different color combinations within the Color Scheme Database, but the basic principle of the color code should still be understood. This is especially true when dealing with non-color monitors. Several of the color codes which would be visible on a color screen will become invisible, or reverse-video, and the pattern for conversion is not obvious.

The only common element between color and monochrome code settings is the 'blinking' bit, (bit 7). Observe the following table:

Bit Settings for mono screen								
Display Attribute	7	6	5	4	3	2	1	0
Non-Display (Invisible)	X	0	0	0	X	0	0	0
Underline	X	0	0	0	X	0	0	1
White-On-Black (normal)	X	0	0	0	X	1	1	1
Black-On-White (reverse)	X	1	1	1	X	0	0	0
Normal Intensity	X	X	X	X	0	X	X	X
Bright Intensity	X	X	X	X	1	X	X	X
Normal Characters	0	X	X	X	X	X	X	X
Blinking Characters	1	X	X	X	X	X	X	X

X: don't care / not applicable
0: Clear / off
1: Set / on

Appendix 4: Miscellaneous Tables

EBCEDIC Codes

Many mainframes and other large IBM equipment used an alternative, 7-bit method for identifying typable and communications characters. This system is called EBCEDIC, and has been expanded to an 8-bit notation on most systems. VariForm uses the conjunction of this table with the ASCII table in this Appendix to generate ASCII <=> EBCEDIC translation features for EBCEDIC databases.

Code	Char.	Name or Meaning	Code	Char.	Name or Meaning
0	NUL		32	DS	
1	SOH		33	SOS	
2	STX		34	FS	
3	ETX		35	LCUR	
4	PF		36	BYP	
5	HT		37	LF	
6	LC		38	ETB	
7	DEL		39	ESC	
8	GE		40	LDR	
9	RLF		41	CDR	
10	SMM		42	SM	
11	VT		43	CU2	
12	FF		44	MDR	
13	CR		45	ENQ	
14	SO		46	ACK	
15	SI		47	BEL	
16	DLE		48	LPER	
17	DC1		49	LNER	
18	DC2		50	SYN	
19	TM		51	LCFR	
20	RES		52	PN	
21	NL		53	RS	
22	BS		54	UC	
23	IL		55	EOT	
24	CAN		56	LFR	
25	EM		57	CFR	
26	CC		58	AFR	
27	CU1		59	CU3	
28	IFS		60	DC4	
29	IGS		61	NAK	
30	IRS		62	AUR	
31	IUS		63	SUB	

Appendix 4: Miscellaneous Tables

(Table Continued on the next page)

Appendix 4: Miscellaneous Tables

EBCDIC Codes

Code	Char.	Name or Meaning	Code	Char.	Name or Meaning
64	STH	Space	96	- STD	
65	LA		97	/	
66	STC		98		
67	IC		99		
68	EX		100		
69	BAL		101		
70	BCT		102		
71	BC		103	MXD	
72	LH		104	LD	
73	CH		105	CD	
74	¢ AH		106	AD	
75	. SH		107	, SD	
76	< MH		108	% MD	
77	(		109	- DD	
78	+ CVD		110	> AW	
79	| CVB		111	? SW	
80	& ST		112	STE	
81			113		
82			114		
83			115		
84	N		116		
85	CL		117		
86	0		118		
87	X		119		
88	L		120	LE	
89	C		121	CE	
90	! A		122	: AE	
91	$ S		123	# SE	
92	* M		124	@ ME	
93	) D		125	' DE	
94	; AL		126	= AU	
95	¬ SL		127	" SU	

(Table Continued on the next page)

Appendix 4: Miscellaneous Tables

EBCDIC Codes

Code	Char.	Name or Meaning	Code	Char.	Name or Meaning
128		SSM	160		
129	a		161	~	
130	b	LPSW	162	s	
131	c	diag	163	t	
132	d	WRD	164	u	
133	e	RDO	165	v	
134	f	RXH	166	w	
135	g	RXLF	167	x	
136	h	SRL	168	y	
137	i	SLL	169	z	
138		SRA	170		
139		SLA	171		
140		SRDL	172		
141		SLDL	173		
142		SRDA	174		
143		SLDA	175		
144		STM	176		
145	j	TM	177		
146	k	MVS	178		
147	l	TS	179		
148	m	NI	180		
149	n	CLI	181		
150	o	0I	182		
151	p	YJ	183		
152	q	LM	184		
153	r		185		
154			186		
155			187		
156		S10	188		
157		T10	189		
158		H10	190		
159		TCH	191		

(Table Continued on the next page)

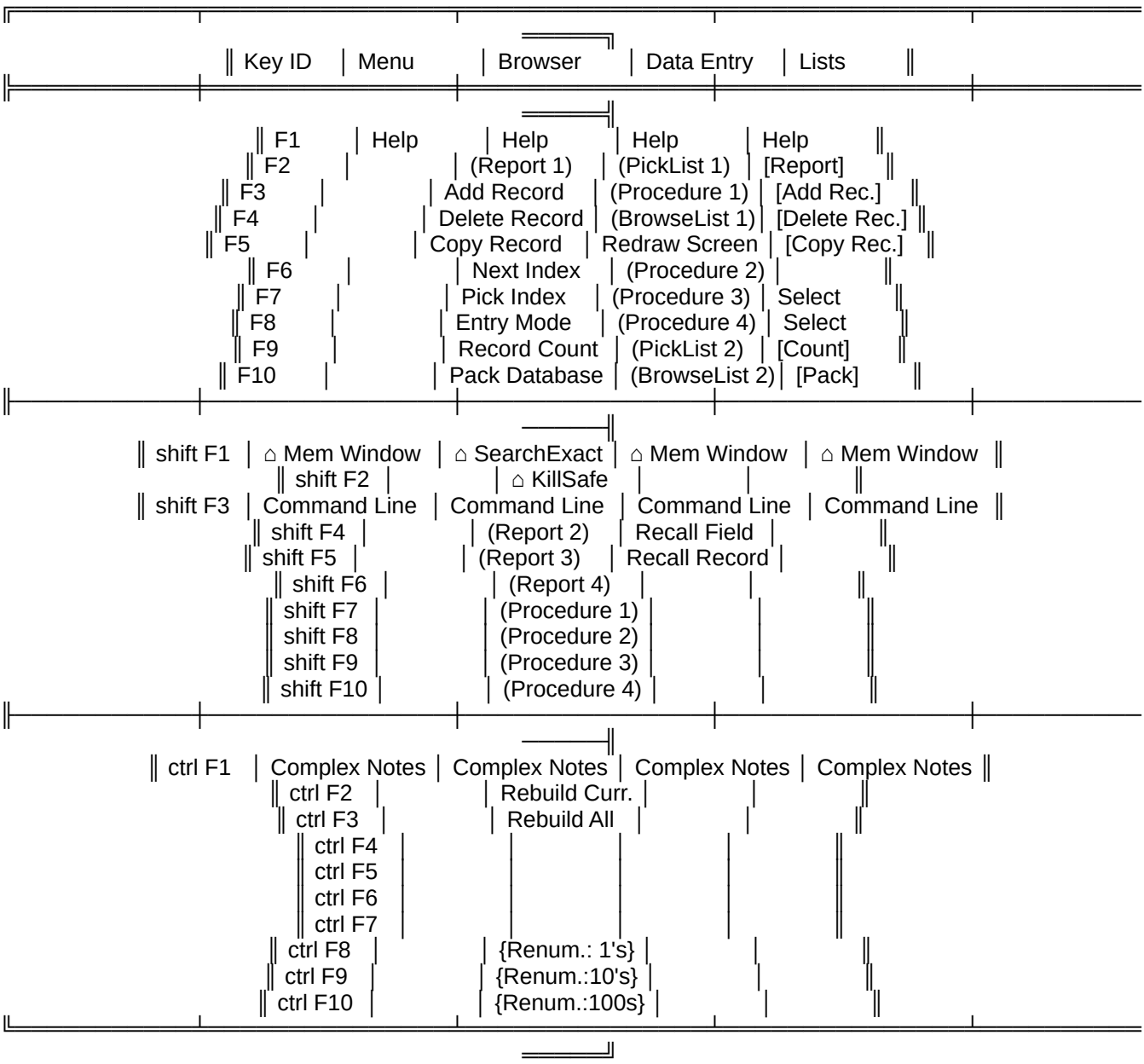
Appendix 4: Miscellaneous Tables

EBCDIC Codes

Code	Char.	Name or Meaning	Code	Char.	Name or Meaning
192	{		224		
193	A		225		
194	B		226	S	
195	C		227	T	
196	D		228	U	
197	E		229	V	
198	F		230	W	
199	G		231	X	
200	H		232	Y	
201	I		233	Z	
202			234		
203			235		
204			236		
205			237		
206			238		
207			239		
208	}		240	0	
209	J	MVN	241	1	MVO
210	K	MVC	242	2	PACK
211	L	MVZ	243	3	UNPE
212	M	NC	244	4	
213	N	CLC	245	5	
214	O	OC	246	6	
215	P	XC	247	7	
216	Q		248	8	ZAP
217	R		249	9	CP
218			250		AP
219			251		SP
220		TR	252		MP
221		TRT	253		DP
222		ED	254		
223		EDWK	255		EO

Appendix 4: Miscellaneous Tables

VariForm System Function Keys



Key ID	Menu	Browser	Data Entry	Lists
F1	Help	Help	Help	Help
F2		(Report 1)	(PickList 1)	[Report]
F3		Add Record	(Procedure 1)	[Add Rec.]
F4		Delete Record	(BrowseList 1)	[Delete Rec.]
F5		Copy Record	Redraw Screen	[Copy Rec.]
F6		Next Index	(Procedure 2)	
F7		Pick Index	(Procedure 3)	Select
F8		Entry Mode	(Procedure 4)	Select
F9		Record Count	(PickList 2)	[Count]
F10		Pack Database	(BrowseList 2)	[Pack]
shift F1	⌂ Mem Window	⌂ SearchExact	⌂ Mem Window	⌂ Mem Window
shift F2		⌂ KillSafe		
shift F3	Command Line	Command Line	Command Line	Command Line
shift F4		(Report 2)	Recall Field	
shift F5		(Report 3)	Recall Record	
shift F6		(Report 4)		
shift F7		(Procedure 1)		
shift F8		(Procedure 2)		
shift F9		(Procedure 3)		
shift F10		(Procedure 4)		
ctrl F1	Complex Notes	Complex Notes	Complex Notes	Complex Notes
ctrl F2		Rebuild Curr.		
ctrl F3		Rebuild All		
ctrl F4				
ctrl F5				
ctrl F6				
ctrl F7				
ctrl F8		{Renum.: 1's}		
ctrl F9		{Renum.:10's}		
ctrl F10		{Renum.:100s}		

The delta mark (⌂) refers to 'toggling' an in-memory variable: turning it off if it is on, and vice-versa.

The parentheses in the Browser and Data Entry Screen columns refer to recommended placement of functions.

The curly braces in the Browser column refers to certain system browsers only: Renumber only runs on databases that contain an {Order / SubOrder} field set in the first index.

The square brackets in the Lists column refer to BrowseList-exclusive functions, and will not work with PickLists.

Appendix 4: Miscellaneous Tables

Keyboard Scan Codes (US)

VariForm has a built-in converter that provides the ASCII character equivalents for all characters when typed in, and ASCII <=> EBCEDIC converters built into the record read/write routines. Some TSR device drivers, however, may still need the scan codes stuffed into memory by VariForm to simulate typing, instead of the normal "Stuff()" routine.

There are actually 2 scan codes for each key on the keyboard: a "make" code, which means the key has been pressed, and a "break" code, which means the key has been released.

Technical: In most cases, the break code is the make code with bit 8 set.

Main Keyboard Scan Codes

Key	Make Code	Break Code	Key	Make Code	Break Code
A	1E	9E	2@	03	83
B	30	B0	3#	04	84
C	2E	AE	4$	05	85
D	20	A0	5%	06	86
E	12	92	6^	07	87
F	21	A1	7&	08	88
G	22	A2	8*	09	89
H	23	A3	9(	0A	8A
I	17	97	-_	0C	8C
J	24	A4	=+	0D	8D
K	25	A5	[{	1A	9A
L	26	A6	]}	1B	9B
M	32	B2	;:	27	A7
N	31	B1	'"	28	A8
O	18	98	,<	33	B3
P	19	99	.>		
Q	10	90	/?	35	B5
R	13	93	LfSh	2A	AA
S	1F	9F	LfCtl	1D	9D
T	14	94	LfAlt	38	D8
U	16	96	RShft	36	B6
V	2F	AF	RtCtl	E0 1D	E0 9D
W	11	91	RtAlt	E0 38	E0 B8
X	2D	AD	CapLk	3A	BA
Y	15	95	BakSp	0E	8E
Z	2C	AC	Tab	0F	8F
0)	0B	8B	Space	39	B9
1!	02	82	Enter	1C	9C

Appendix 4: Miscellaneous Tables

US Keyboard Scan Codes

Numeric Keypad Scan Codes

Key	Make Code	Break Code
ScrLk	46	C6
NumLk	E0 35	E0 B5
/	45	C5
*	37	B7
-	4A	CA
+	4E	CE
Enter	E0 1C	E0 9C
.		
0	52	D2
1	4F	CF
2	50	D0
3	51	D1
4	4B	CB
5	4C	CC
6	4D	CD
7	47	C7
8	48	C8
9	49	C9

Dedicated Cursor Keys

Key	Make Code	Break Code
PrtScr	E0 2A E0 37	E0 B7 E0 AA
Ins.	E0 52	E0 D2
Del.	E0 53	E0 D3
Home	E0 47	E0 C7
End	E0 4F	E0 CF
PgUp	E0 49	E0 C9
PgDn	E0 51	E0 D1
UpAr	E0 48	E0 C8
DnAr	E0 50	E0 D0
LfAr	E0 4B	E0 CB
RtAr	E0 4D	E0 CD
Pause	E11DE19DC5	No Break

Appendix 4: Miscellaneous Tables

US Keyboard Scan Codes

Function Key Scan Codes

Key	Make Code	Break Code
<F1>	3B	BB
<F2>	3C	BC
<F3>	3D	BD
<F4>	3E	BE
<F5>	3F	BF
<F6>	40	C0
<F7>	41	C1
<F8>	42	C2
<F9>	43	C3
<F10>	44	C4
<F11>	57	D7
<F12>	58	D8
<Esc>	01	81

Appendix 5: Formulas and Explanations

Basic Geometry

Circle

Circumference = 2 * π * radius
Area = π * radius^2

Ellipse

Area = π * max. radius * min. radius

Triangle

Area = ½ height * base

Square

Area = side length^2

Rectangle

Area = Height * Width

Cube

Surface Area = 6 * side length^2
Volume = side length^3

Rectangular Prism (Rectanguloid)

Surface Area = (2 * base * width) + (2 * height * depth) + (2 * width * depth)
Volume = (base + width) * height

Sphere

Surface Area = 4 * π * radius^2
Volume = (4 / 3) * π * radius^3

Cylinder

Surface Area = (2 * π * radius * height) + (2 * π * radius^2)
Volume = π * radius^2 * height

Cone

Surface Area = (π * radius * (radius^2 + height^2)^{½}) + (π * radius^2)
Volume = (π * radius^2 * height) / 3

Appendix 5: Formulas and Explanations

Trigonometry

What the heck is pi?

Pi (Pronounced like a 'pie' for eating) is a ratio that makes dealing with with curves, arcs and angles easier to manage. Technically, pi (mathematical symbol π) is the ratio of enclosed areas between a circle and the smallest possible square that completely encloses the circle. The simple ratio is 22/7, but pi is always a mathematically 'irrational' number: it is an apparently infinite length, non-repeating fraction. The two most commonly accepted value sets for pi are:

3.1415927, which is part of a statistical approximation
3.1428, which is part of a 'simple' ratio, used before statistics.

VariForm uses the former, statistical value for all trigonometry calculations, as well as for degree-to-radian conversions. Its accuracy is maintained for 15 decimal places.

Degrees, Radians, Gradients: what's the difference?

No matter what the application, trigonometry reduces to performing calculations based on angular measurements. An angular measurement is a number that describes the fraction of a circle that the angle would cover. The difference between degrees, radians and gradients is the method of measuring a circle.

Degree measurements break the unit circle into 360 units, or degrees, of equal size (the size of an angle is referred to as 'arc').

Gradient measurements break the unit circle into 400 units, called gradients.

Radian measurements use the number called 'pi', or 3.14... to divide the unit circle: the full circle is "2 pi" radians, or 6.28... radians.

There are no accuracy problems in converting these measurements. VariForm provides the degree-to-radian conversions, and below is a table for the rest of the conversions if necessary.

From↓ To→	Degrees	Radians	Gradients
Degrees	x 1	x π/180	/ 0.9
Radians	x 180/π	x 1	x 200/π
Gradients	x 0.9	x π/200	x 1

So, to get radians from degrees, multiply the number of degrees by 'pi', and divide the product by 180. The result will be a radian measure, accurate to the number of decimal places used

for pi.

Appendix 5: Formulas and Explanations

Trigonometry

Trigonometry calculations are often represented by something called a 'unit triangle', which is a triangle that shows the angle's scope within a unit circle. A unit triangle is a "right triangle", which means one angle is 90 degrees or π/2 radians; the side of the triangle across from this 'right angle' is called the 'hypotenuse'. The other two sides of the triangle are named according to which of the two angles is being observed: the side that touches the angle (in addition to the hypotenuse) is called the adjacent side, and the side that does not touch the angle is called the 'opposite' side.

The basic trigonometric functions refer to ratios of various combinations of the sides of a unit triangle. These ratios allow engineers to calculate simple loads, stresses, and 'shadow angles', without having to measure every possible distance 'out in the field'.

The sine is the ratio of the opposite side over the hypotenuse.	o / h
The cosine is the ratio of the adjacent side over the hypotenuse.	a / h
The tangent is the raio of the opposite side over the adjacent side.	o / a
The cotangent/arc tangent is the adjacent over the oppositeside.	a / o
The secant is the ratio of the hypotenuse overthe adjacent side.	h / a
The cosecant is the ratio of the hypotenuse over the opposite side.	h / o

Other trigonometry related formulas:

Pythagorean Relations:

$\text{Sin}(\Theta)^2 + \text{Cos}(\Theta)^2 = 1$
$1 + \text{Tan}(\Theta)^2 = \text{Sec}(\Theta)^2$
$1 + \text{Cot}(\Theta)^2 = \text{Csc}(\Theta)^2$
$a^2 + o^2 = h^2$

Product Relations:

$\text{Sin}(\Theta) = \text{Tan}(\Theta) * \text{Cos}(\Theta)$ $\quad \text{Cot}(\Theta) = \text{Cos}(\Theta) * \text{Csc}(\Theta)$
$\text{Cos}(\Theta) = \text{Cot}(\Theta) * \text{Sin}(\Theta)$ $\quad \text{Sec}(\Theta) = \text{Csc}(\Theta) * \text{Tan}(\Theta)$
$\text{Tan}(\Theta) = \text{Sin}(\Theta) * \text{Sec}(\Theta)$ $\quad \text{Csc}(\Theta) = \text{Sec}(\Theta) * \text{Cot}(\Theta)$

Function Product Relations:

$\text{Sin}(\Theta) * \text{Sin}(ß) = (.5 * \text{Cos}(\Theta - ß)) - (.5 * \text{Cos}(\Theta + ß))$
$\text{Cos}(\Theta) * \text{Cos}(ß) = (.5 * \text{Cos}(\Theta - ß)) + (.5 * \text{Cos}(\Theta + ß))$
$\text{Sin}(\Theta) * \text{Cos}(ß) = (.5 * \text{Sin}(\Theta + ß)) + (.5 * \text{Sin}(\Theta - ß))$
$\text{Cos}(\Theta) * \text{Sin}(ß) = (.5 * \text{Sin}(\Theta + ß)) - (.5 * \text{Sin}(\Theta - ß))$

Double Angle Relations:

$\text{Sin}(2 * \Theta) = 2 * \text{Sin}(\Theta) * \text{Cos}(\Theta) = 2 * \text{Tan}(\Theta) / (1 + \text{Tan}(\Theta)^2)$
$\text{Cos}(2 * \Theta) = \text{Cos}(\Theta)^2 - \text{Sin}(\Theta)^2 = (2 * \text{Cos}(\Theta)^2) - 1 = 1 - (2 * \text{Sin}(\Theta)^2)$
$\text{Tan}(2 * \Theta) = (2 * \text{Tan}(\Theta)) / (1 - \text{Tan}(\Theta)^2)$
$\text{Cot}(2 * \Theta) = (\text{Cot}(\Theta)^2 - 1) / (2 * \text{Cot}(\Theta))$

Appendix 5: Formulas and Explanations

"English" Weights and Measures

The 'English Measure' or Averdupois system of measurements is still the most common unit of measure in the United States of America. The disadvantage to this system of measurements is that there are multiple 'minimum units' for standards of length, area, weight, volume, and temparature.

(Liquid) Volume

27.34 Grains	1 Dram
16 Drams	1 Ounce (Averdupois)
1.0971 Oz. (av)	1 Ounce (Troy)
3 teaspoons	1 tablespoon
8 tablespoons	1 cup
1 cup	8 fluid ounces
2 cups	1 pint
2 pints	1 quart
4 quarts	1 gallon
1 gallon	231 cubic inches
1 barrel	42 gallons

Length

12 inches	1 foot
3 feet	1 yard
6 feet	1 fathom
5,280 feet	1 mile
1 mile	0.86897624 nautical miles

Area

43560 Square Feet	1 Acre

Weight

12 or 16 ounces	1 pound (12 ounces is for precious metals)
26 pounds	1 stone
5 stones	1 hundredweight
2000 pounds	1 'short' ton
1 Short Ton	0.89286 Long Ton

Solid Volume

1 Board Foot	144 Cubic Inches
1 Cord	128 cubic feet
43560 Cubic Feet	1 Acre foot

Rod, bushel, acre, hectare, pennyweight, dram, furlong, peck, bushel

Appendix 5: Formulas and Explanations

Water freezes at	32 degrees Farenheit
"Normal" Human	98.6 degrees Farenheit
Water boils at	212 degrees Farenheit
Pig Steel melts at	2000 degrees Farenheit

Appendix 5: Formulas and Explanations

The Metric System

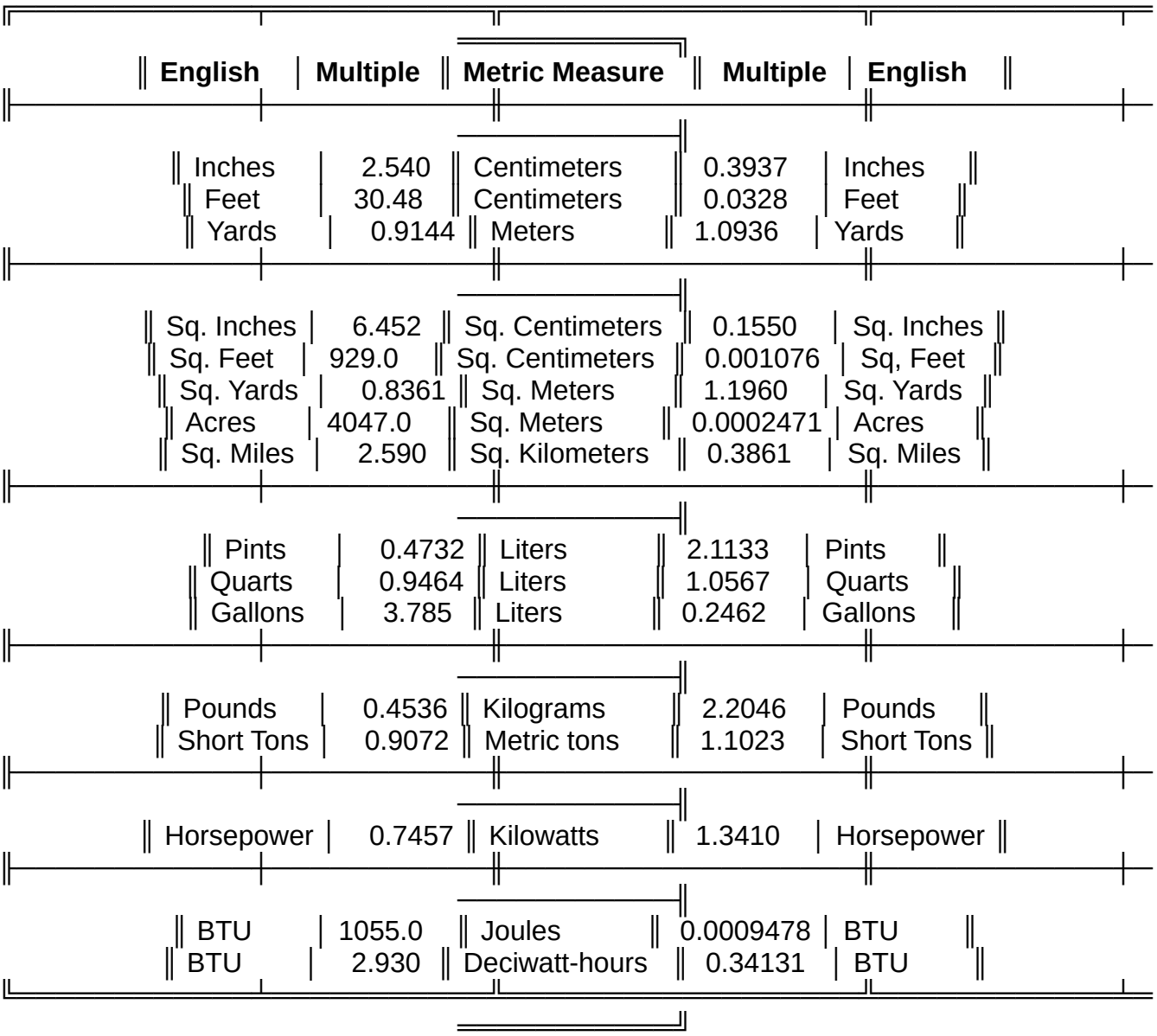
In most countries outside of the United States, the standard units of measure are based on the 'metric system'. The metric system uses powers of ten for all unit divisions, so only the prefixes must be memorized: there is one standard weight (gram), one standard length (meter), area and volume are multiples of the length unit and on unit of thermal measure (called Centigrade or Celsius).

Pico	1 trillionth	.000000000001
Nano	1 billionth	.000000001
Micro	1 millionth	.000001
Milli	1 thousandth	.001
Centi	1 hundredth	.01
Deci	1 tenth	.1
Deca/Deka	Ten	10
Hecto/Hecta	Hundred	100
Kilo	Thousand	1,000
Mega	Million	1,000,000
Giga	Billion	1,000,000,000
Tera	Trillion	1,000,000,000,000

Water freezes at	0 degrees Centigrade
"Normal" Human	37.6 degrees Centigrade
Water boils at	100 degrees Centigrade
Pig Steel melts at	1093 degrees Centigrade

Appendix 5: Formulas and Explanations

English - Metric Conversions



English	Multiple	Metric Measure	Multiple	English
Inches	2.540	Centimeters	0.3937	Inches
Feet	30.48	Centimeters	0.0328	Feet
Yards	0.9144	Meters	1.0936	Yards
Sq. Inches	6.452	Sq. Centimeters	0.1550	Sq. Inches
Sq. Feet	929.0	Sq. Centimeters	0.001076	Sq, Feet
Sq. Yards	0.8361	Sq. Meters	1.1960	Sq. Yards
Acres	4047.0	Sq. Meters	0.0002471	Acres
Sq. Miles	2.590	Sq. Kilometers	0.3861	Sq. Miles
Pints	0.4732	Liters	2.1133	Pints
Quarts	0.9464	Liters	1.0567	Quarts
Gallons	3.785	Liters	0.2462	Gallons
Pounds	0.4536	Kilograms	2.2046	Pounds
Short Tons	0.9072	Metric tons	1.1023	Short Tons
Horsepower	0.7457	Kilowatts	1.3410	Horsepower
BTU	1055.0	Joules	0.0009478	BTU
BTU	2.930	Deciwatt-hours	0.34131	BTU

BTU: Abbreviation for British Thermal Unit, which is about 0.252 Kilocalories, or the thermal energy necessary to raise 1 pound of water from 39.2 to 40.2 degrees Farenheit at standard temparature and pressure.

Farenheit: subtract 32, then multiply by 5 / 9 for centigrade.

Centigrade: multiply by 9 / 5, then add 32 for Farenheit.

To use this table: multiply or divide units to get an english unit in the left column, or a metric unit in the center column. Multiply by the number **to the immediate right** of the current units, and the product will be the units listed to the right of that numeric column.

Appendix 6: Boolean Logic

Overview

All computer systems, as well as most philosophical systems, are based on some rudimentary form of logic. In terms of the 'modern' computer system, such as a personal computer, the logic employed is binary, representing the 'on' and 'off' states of an electrical circuit. Binary logic was defined by a series of mathematicians that go as far back as Aristotle and the Greek Empire. The primary, defining works that systemized logic into algebraic terms was a professor of mathematics in Queen's college of Cork, Ireland, by the name of George Boole. This is why logical expressions and operators are often called 'Boolean' operations, or operators that perform 'Boolean Algebra'.

Boole's two most relevant works for computers were a pamphlet entitiled *Mathematical Analysis of Logic*, written in 1847, and a book called *An Investigation of the Laws of Thought, on Which Are Founded the Mathematical Theories of Logic and Probabilities*, published in 1854. Through these works and others, Boole created a system of symbols and analysis that allowed for in-depth mathematical analysis of logical 'propositions' and 'statements'. This mathematics has been developed, and continues to evolve today. An understanding of the fundamentals of Boolean logic will assist any person who needs to write search queries, edit data entry screens, or develop application code.

The meaning of zero

Zero means 'false', 'not-true', or 'off'. When VariForm interprets a Search Type 1 (condition) expression, a record that returns a value of zero is <u>not</u> selected; when it interprets a Search Type 6 (stop condition) expression, a record that returns a value of zero does <u>not</u> stop the search.

The meaning of non-zero

In VariForm, any non-zero value is interpreted as 'not-false', or 'true'. This is a slight variation from Boolean Algebra, where there are only ones and zeros: in VariForm, there is False: all else is true. This is only slightly different from 'fuzzy logic', where there are varying degrees of truth. VariForm Notation, in fact, lends itself to hybrid combinations of fuzzy and boolean logic, where the procedures can be programmed to interpret fuzzy logic response values, and the Search system maintains a rigid enforcement of Boolean restrictions.

Appendix 6: Boolean Logic

Basic Functions

The primary definition of Boolean Logic, as it applies to VariForm, states that all expressions will evaluate to either a true or false overall meaning. In VariForm terms, a logical expression will consolidate the meanings of one or several statements to a singular true or false value, which will allow the search or condition to operate, or not, based on that final value at a particular point in time or data.

Positive versus Negative Operators

In VariForm, there are two basic forms of operators: positive operators, which require a condition to be true to evaluate to true, and negative operators, where a true condition evaluates to false. In VariForm, all Negative operators start with the letter 'N'.

Positive Operators

There are three basic operators: And, Inclusive Or, and Exclusive Or. These represent the logical operations of combination, option, and singularity. These operators will be addressed in order.

And: for an 'and' statement to be true, **both** conditions supplied must be true, or non-zero. So (true and true) is true, while (true and false), (false and true), and (flas and false) are all false.

Inclusive Or: for an 'or' statement to be true, **at least one** of the conditions supplied must be true or non-zero.

Exclusive Or: for an 'xor' (pronounced ex-or) to be true, **only one** of the conditions supplied must be true or non-zero.

To simplify the explanation of these measnings, scientists and logicians have developed a set of 'truth tables', which is a table that defines what result or output comes from a particular expression.

Input 1	Input 2	1 And 2	1 Or 2	1 XOr 2
False	False	False	False	False
True	False	False	True	True
False	True	False	True	True
True	True	True	True	False

Appendix 6: Boolean Logic

Basic Functions

Negative Operators

Negative operators are basically the opposite of positive operators: they output false when a positive operator would be true, and output true when a positive operator would output false. They employ the same three basic operators: Nand, Inclusive NOr, and Exclusive NOr. these represent the logical operations of combination, option, and singularity. These operators will be addressed in order.

Nand: for a 'Nand' statement to be false, **both** conditions supplied must be true, or non-zero. So (true and true) is false, while (true and false), (false and true), and (flas and false) are all true.

Inclusive NOr: for a 'Nor' statement to be false, **at least one** of the conditions supplied must be true or non-zero.

Exclusive Or: for an 'Nxor' (pronounced en-ex-or) to be false, **only one** of the conditions supplied must be true or non-zero.

Input 1	Input 2	Nand	NOr	NXor
False	False	True	True	True
True	False	True	False	False
False	True	True	False	False
True	True	False	False	True

The inverter: Not()

The only unary (single-input) operator in VariForm's Boolean Logic rocessing is the 'Not' operator, which inverts the value of the input. If the input is true, the output is false; if the input if false, the output is true. Technically, all 'negative operators' are simply positive operators with an inverter on the output.

a Nand b = not(a and b)
a NOr b = not(a or b)
a NXor b = not(a xor b)

Appendix 6: Boolean Logic

Commutative Properties

All logical operations, except the inverter, are commutative if the operations are the same. This means that the order of evaluation does not affect the outcome of the operation, if the values do not depend on each other, and are being operated on in the same fashion. So, while the expression parser works from right-to-left, and the human eye operates from left-to-right, the outcomes should be the same.

(A and B) is the same as (B and A)
(A or B) is the same as (B or A)
(A xor B) is the same as (B xor A)

Associative Properties

Parentheses allow the operator to override the standard order of operation, and impose a specific order of evaluation to an expression. A associative operation is one where the parenthesis can be moved without affecting the outcome of the operation. In logic, Logical operations are associative until the operation type changes.

(A and B and C) is the same as ((A and B) and C), as well as (A and (B and C)):
All three conditions must be true for a true result

(A or B or C) is the same as ((A or B) or C), as well as (A or (B or C)):
Any of the three conditions will produce a true result

((A XOr B) XOr C) is the same as (A Xor (B Xor C)):
Exactly one of the three conditions must be true for a positive result

(A and (B or C)) is not the same as ((A and B) or C)):
In the first case, A must be true, and one of B or C
In the second case, A and B must both be true, or C

Appendix 6: Boolean Logic

Application to VariForm Expressions

In a search, Logical expressions are used in Type 1 (condition), Type 6 (stop condition), and Type 7 (reverse-order stop condition) records. In procedures, expressions are used inside If, ElseIf, IfCall, and While statements, where a true or false evaluation determines which procedure line will be executed next.

Type 1 Records, if evaluated to 'true', cause the search to stop and select the current record being searched.

Type 6 Records, if evaluated to 'true', cause the search to stop and indicate that it has reached the end of all possible records.

Type 7 Records, if evaluated to 'true', cause a reverse-search to stop and indicate that the searched has reached the end (beginning) of all possible records.

If() statements, if evaluated to 'true', cause the lines enclosed in the related curly braces ({ and }) to be executed. The same lines are skipped if the expression evaluates to false.

ElseIf() statements, if the previous 'If' statement evaluated to 'false', and the contents evaluate to 'true', cause the lines enclosed in the curly braces to be executed.

Else statements, if the previous 'If' or 'ElseIf' statement evaluated to 'false', cause the lines enclosed in the curly braces to be executed.

While statements, if evaluated to true, cause the lines enclosed in the curly braces to be executed repeatedly, until the 'While' evaluation at the beginning or end of the 'loop' becomes false.

Appendix 7: Making the Most of Memory

Computers have two basic types of memory: temporary and permanent, also called RAM and Disk Storage. "Optimizing Memory" refers to making better use of RAM. There are only three ways to optimize disk storage: use a Data Compression Module, Shrink the Indexes (see Appendix 1 on page), or delete records, Pack the Databases, and delete the *.bak files.

To make the best use of RAM, the user or system designer should understand what RAM is, and how the various types of RAM are used before getting into the details of 'tweaking' RAM for the best fit for an application.

Section 1: What is RAM?

RAM is an acronym that stands for "Random Access Memory". On PC's, RAM refers to chip-based memory, with typical sizes ranging between 512 KiloBytes to 16 MegaBytes. Many people get confused about RAM because there are several categories of RAM, each of which is used and accessed differently.

After purchasing a computer with '4 MegaBytes of RAM', a user plugs in the computer, turns it on, sees a number count to 4096 Kb, and everything seems allright. After booting, however, a MEM, or CHKDSK command will probably say that there is less than 600K of 'conventional memory' available, and the 'largest executable' will probably be less than that. How can both statements be true? Please refer to the diagram on the next page while reading the next section.

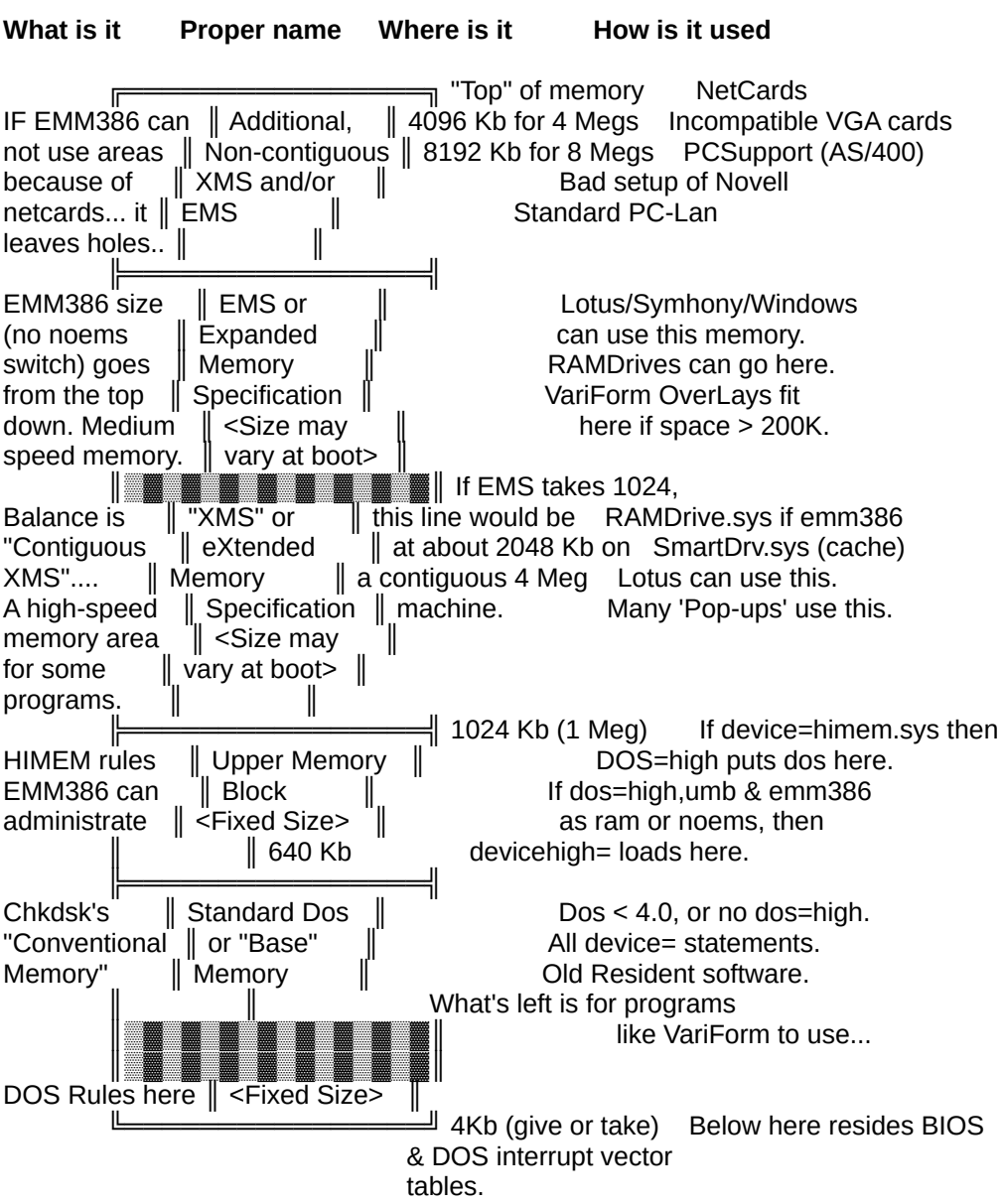
DOS can only directly access the first MegaByte of RAM, and can only use the first 640 KiloBytes without outside assistance. This is because DOS was designed with an inherent limitation of 640 KiloBytes. Accessing the other memory requires a few more names than just RAM, and a few other programs as well.

Memory above the first 640 KiloBytes is called "Upper Memory", or "Extended Memory", and refers to memory that requires a program called a 'device driver' that shows DOS how to access it. The area between the 640 KiloByte boundary and the end of the first MegaByte is called the Upper Memory Block; all memory above the 1 MegaByte line is called Extended Memory.

Extended Memory has its uses, and is available if DOS loads an upper memory manager, such as HIMEM.SYS during Boot-Up. Check the Config.sys file: if there is a line that says "DEVICE=HIMEM.SYS", then there is a memory manager in use.

Extended Memory can also be used in a more standard fashion, thanks to some standards developed by a committee formed by Lotus (spreadsheet maker), Intel (PC Chip Maker), and MicroSoft (MS-DOS designer). These standards are called the LIM-spec, and refer to Expanded Memory Specifications, or EMS. LIM-EMS allows programs to use upper memory in a standard way on every computer. To see if EMS is loaded, check the config.sys file: if there is a line that says "DEVICE=EMM.SYS", or EMM386, or 386MAX, or QEMM, then there is an EMS memory manager in use.

Appendix 7: Making the Most out of Memory



Appendix 7: Making the Most out of Memory

Section 2: Making Room

VariForm needs a minimum of 400 KiloBytes available to run the system default interfaces. For custom interfaces, or screens or reports with many lookups or layers, the memory requirement is often higher, around 480 - 540 KiloBytes. These numbers should appear in the CHKDSK statement next to "Conventional Memory Available", or next to similar words in the DOS 5+ MEM command.

More Conventional Memory

PC and XT users have only 1 set of options: remove residents or reduce resident consumption. The following section is primarily aimed at them. AT, 386, 486, and Pentium users should skip to the section on extended memory, referring to this section as a last resort.

Remove Residents: Programs such as a Mouse Driver, a 'pop-up' programs such as screen savers and SideKick can be removed to make more space. Either consult the appropriate manuals on these programs to remove them, or comment them out of the config.sys and autoexec.bat files and reboot.

> **Commenting out residents:** use a text editor, such as edlin or edit, and put the word "REM " (note the space) in front of the lines that load resident programs, such as DEVICE= and INSTALL= statements in config.sys, and resident programs loaded by autoexec.bat.
>
> Do not comment out XMS or EMS residents, such as HIMEM.SYS, or EMM386.exe. They will be useful later.

Reducing resident consumption: consult the resident program's manual; many TSR (Terminate and Stay Resident) programs have a 'size' parameter that can be reduced.

> **DOS has residents, too:** several parameters in the config.sys file can usually be changed safely to provide more memory. While VariForm requires the FILES setting to be high (at least 150), many programs reccommend setting other parameters artificially high. If any of the following lines are in config.sys, make sure that the file's numbers are no higher than the numbers here:
>
> BUFFERS = 20 to 30 for dBase and WordPerfect, 15 otherwise
> STACKS = 16,256 for most networks
> STACKS = 9,256 or 8,128 for most standalone computers
> FCB or FCBS = 16 or less unless there is pre-1986 software running

PC and XT users should skip the next section, since it refers to memory managers that are only available on 286 or better machines. This is also true for owners of DOS versions below 4.0.

Appendix 7: Making the Most out of Memory

Section 2: Making Room

UMB's, EMS instead of Conventional

This section is for owners of AT/286 or better, with at least 1 MegaByte of RAM, and who are using DOS 5.0 or higher. For DOS 6.0 and above, most of this section will be performed by running MEMMAKER during of after DOS 6 installation, if **all residents** are located in config.sys and autoexec.bat.

"MEM /C" will provide a classification list of all tsr's currently in memory. If there are residents in conventional memory, one of two things is likely: either the software was configured in low memory (accident), or it will only work in low memory.

The config.sys file should contain at least these lines:

- DEVICE=HIMEM.SYS
- DEVICE=EMM386.EXE (this may be followed by a size, and RAM or NOEMS)
- DOS=HIGH,UMB

After these statements:

- "DEVICE=" statements should all be "DEVICEHIGH=" statements.
- "INSTALL=" statements should be "INSTALLHIGH=".
- Other statements calling programs (not .bat files) should be preceded by a "LOADHIGH " or "LH " statement on the same line.

FastCache Users: if you have a choice between FastCache and SmartDrive, use SmartDrive: it is faster, more accurate, and far less memory-intensive than FastCache.

SmartDrive Users: if you are on a 386 or better, DEVICEHIGH=SMARTDRV.SYS will detect and use XMS, if it is available. Without sufficient XMS, SmartDrive defaults to Conventional memory, so use the /E or /X switch to force it into appropriate memory locations. That way, if it won't fit, it will provide an error message during boot-up.

DOS 5.0 Users: avoid the "SHELL=" and the "Command /c" statements in batch files, such as autoexec.bat. DOS 5, 6, and 6.2 all provide better return mechanisms that do not consume such large quantities of Conventional memory.

PCSupport Users: in addition to making changes on the PC, making changes to the PCSupport configuration to use Extended DOS Setup will save 20-45 KiloBytes of Conventional Memory, but will cause a (one-time) massive file folder update to your terminal at next-login time.

LanTastic Users: if the computer is acting as a 'host', and is offering devices to the network, reducing the network task buffer size, the printer buffer size, and the number of tasks and users in Host Startup Settings can also dramatically increase available Conventional Memory.

If you are on a network, or use PCSupport (AS/400 terminal) drivers, there are separate versions of the programs and devices to make the best possible use of the XMS and EMS available, which will also free up conventional memory. You must reboot for these changes to take effect.

Appendix 7: Making the Most out of Memory

Section 3: Where does it all go?

When a computer reports 400 KiloBytes of conventional memory free, and 200 KiloBytes of EMS free, VariForm can run (Core). When a computer reports 500 KiloBytes of conventional memory free, VariForm can run without EMS overlays (WCore).

After the system has loaded and initialized, the Memory Window will display significantly less memory, based on user settings:

- Core consumes 120 Kb of RAM; WCore consumes 200 Kb
- System data (system autoloads) consumes 60 Kb of RAM by default
- Virtual Field Space is an additional memory cost (minimum = 4K)
- Internal Cache Memory is an additional memory cost (minimum = 0 (off))

Most medium-sized applications will run in under 100Kb free. To 'shrink' an application's RAM consumption, consider the following 'Rules of Thumb':

- **Screens: Bigger is (often) Better**: One large screen uses less RAM than 2 small screens. The exceptions to this are when a sub-screen is seldom-used, or when there are more than 2 sub-screens to the main screen.

- **Indexes: Check Price/Performance**: On multi-user VariForms, no database should have more than 2 buffer blocks: OnNetwork defeats the advanced index caching. On single user systems, try near multiples: there are cases (see Appendix 1 on page) where 4 blocks of 1024 bytes are better and as fast or faster than 3 blocks of 2,048 bytes. For general editing, more blocks of smaller sizes are usually better performers for the memory consumption.

- **Strength in Solitude**: Mandatory relationships force all related sub-databases to open when their 'parent' database opens, costing time and RAM. It is very easy to become a 'relational hog', connecting 30 databases together, when there are actually only 4 relationships and 26 lookups.

  Strength in Solitude applies especially to many-to-one and many-to-many relationships, which should never be mandatory relationships. Nested fields can provide relational-appearing data entry and reports. Why load the searches and sub-sub-sub databases when it is not necessary?

- **Keep Children Small**: sub-databases should almost always be significantly smaller than their respective parent, in terms of record size and number of indexes. If this is not the case, the database structure layouts may need to be redefined.

- **Consolidate information**: Put similar lookups in the same database, as exemplified by the Charts and Tables databases. Use the Charts and Tables databases instead of creating a new single-integer or <18 character database.

Appendix 7: Making the Most out of Memory

Section 3: Where does it all go?

◦ **Cue lines add up**: VariForm most put all the cue lines for a Data Entry Screen in memory at one time, so keep them brief. Do not put cue lines in Display Field records that are parts of reports: they will take up memory and serve no purpose whatsoever.

◦ **Recycling Pays**: Combine lookups and other search structures: one search that loads four variables is smaller and faster than four searches that load one variable each.

Appendix 8: Acronyms and Abbreviations

Ascii - American Standard Code for Information Interchange: a series of numbers that represent all characters used in both typing and communications.

Baud - Bits per second : serial communication speed measurement.

BCD - Binary Coded Decimal : a method of representing numbers to the computer in base 10, to eliminate all possibility of a round-off error in basic floating-point work.

BIOS - Basic Input / Output System : this is the program stored on ROM and/or CMOS chips that tells the computer how to communicate with other hardware devices, such as the keyboard, printer port, hard and floppy drives. BIOS refers to the entire system of ROM, CMOS, and peripheral boards' permanent memory settings.

BTU - British Thermal Unit : a measurement of thermal energy or heat content.

CD-ROM - Compact Disc - Read Only Memory. A removable mass-storage device that can only be read from: useful for large programs or encyclopedic-size databases; commonly used in 'multimedia' applications.

CLI - Command Line Interface : the single-command entry mode of VariForm.

CLP - CommandLine Processor : the engine that interprets and acts on individal commands and procedure statements sent in from Function Keys, Procedure Calls, Searches, or the CLI.

CMOS - Complementary Metal Oxide Semiconductor : An engineering term referring to a very-low power consumption design for microchips. Common Usage: a usually programmable, semi-permanent part of some computer's memory that contains settings about the computer's hardware settings, capabilities and intended uses. CMOS contains things like pre-boot passwords, size and type of hard drive(s), and special uses of RAM memory. When CMOS technology is used within a CPU, the letter 'C' will be inserted into the chip's description, as in 80C86 (instead of 8086).

CPU - Central Processing Unit : the 'brain' or decision-making chip(s) inside the computer.

CRC - Cyclical Redundancy Check : a complex mathematical form of error detection used by most hard disk drives, and generally available on VariForm databases. Each record in a database can be followed by a CRC value for the record; if the record does not match its trailing CRC, it is listed as 'damaged'.

DB - Database

DBMS - Database Management System

DES - Data Encryption Standard : A standard encryption method defined by the National Institute of Standards and Technology for encoding files or records. The method is considered extremely difficulty to crack, and is considered acceptable (if DES certified by NIST) for use in "Sensitive" (level C2 or lower) information.

DMA - Direct Memory Access : a system, usually separate from the central processor unit, that

allows for the rapid transfer of RAM information.

EDI - Electronic Data Interchange. A standardized method of communication business-related information directly between computers, or through a VAN.

E-Mail - Electronic Mail. This term refers to information in the form of 'notes', or 'letters' that can be sent between computers through a LAN, WAN, or alternative communication system.

EMM - Expanded Memory Manager : a program or device driver that tells MS-DOS and programs how to access EMS memory. EMM programs include EMM386.EXE, 386Max, and QEMM.

EMS - Expanded Memory System : with a device driver, EMS allows programs running under DOS in real mode to access memory above the 1-Megabyte Limit. LIM-EMS version 4.0 combined the original EMS with the EEMS enhancements.

EEMS - Enhanced Expanded Memory System : an early elaboration of EMS.

FPU - Floating Point Unit : a co-processor (a secondary CPU) that specifically performs mathematical operations with high-precision accuracy, at very high speeds.

GB - Gigabyte : 1K Megabytes: about 1 billion bytes: (10^9 bytes); precisely 1024^3 bytes.

IDX - Index.

IEEE - Institute for Electrical and Electronic Engineers. An engineering trade association that establishes and maintains several hardware-based standards for communications and computer systems.

KB - Kilobyte : also abbreviated K: 1024 bytes

KBaud - Kilobaud : Thousands of bits per second.

kbd - Keyboard

LAN - Local Area Network. A method for 'local' computers to share services and resources.

LIM - Lotus, Intel, Microsoft : refers to a committee formed to create generic specifications for expanded and extended memory management.

MB - Megabyte : about 1 million bytes: (10^6 bytes); precisely 1024^2 bytes.

Mb / MBaud - MegaBaud : 1 million bytes per second. Serial communications speed.

Modem - Modulator/Demodulator. A device that allows computers to communicate through telephone systems by converting (modulating) digital serial signals into tones, and demodulating the incoming tones back into digital signals for the computer.

ODBS - Open Database System : an emerging 'standard' for active interaction between different types of databases. Any program that implements this standard can read data files from any other program that uses this standard.

ODI - Open Data-link Interface. Novell's peer-to-peer program and data connection standards.

OOP - Object Oriented Programming. A system of programming where sections of programs can be attached to internal data structures. Theoretically, the attachment scheme allows for software that manages the objects to be reusable. This term refers to extensions to Pascal, Modula-2, ADA, and an extension of the C programming language called C++.

POST - Power On Self Test: part of the 'booting' process, this is a diagnostic routine called when the computer is either turned on or reset. POST checks for available RAM, peripheral boards, floppy drives and hardware ports and connections.

RAM - Random Access Memory; also called 'chip memory', RAM refers to memory that the computer uses for temporary storage and execution of programs.

RDBMS - Relational Database Management System: a program that allows multiple databases to be 'related' or connected together in specific ways.

ROM - Read Only Memory. This is a part of the computer that stores permanent system programs, such as POST, Video board communications, and other basic hardware based information.

SQL - Structured Query Language : a standard for programming-like interface connections between databases. While many databases boast of 'SQL compatibility', very few systems can actually interoperate: most are simply very similar approximations of the SQL standard.

TB - TeraByte: approximately 1 Trillion bytes: about 10^{12} bytes; precisely 1024^4 bytes.

TSR - Terminate and Stay Resident. This identifies programs that 'load' into memory after the computer has booted, such as mouse programs, and computerized 'scratch pads'.

UMB - Upper Memory Block. This refers to the section of RAM between 640 KB and 1024 Kb (1 Meg). On 80286 or better computers using at least DOS 5.0, some TSR programs and device drivers can be loaded into this area to conserve 'conventional memory'.

VAN - Value Added Network. A third-party electronic mailboxing system for EDI. A Van will receive mail from one customer, forward it to the intended recipients' electronic mailboxes, and deliver it when requested to do so by the recipient.

WAN - Wide Area Network. A network of machines that is larger in geographic scope than one building, usually describing networks with terminals separated more than 5 geographic miles apart.

WORM - Write Once Read Many. Refers to CD-style storage devices which can be written to one time; often used in larger-scale backup systems for permanent archival records.

X.12 - ANSII standards definitions for Electronic Data Interchange.

XMS - Extended Memory System. This refers to all physical memory above 1 Megabyte (1024 Kb)

Appendix 9: Definitions

Averdupois - The 'English' system of weights, including ounces, pounds, short and long tons.

Baud - Bits per second : serial communication speed measurement.

Binary - The Base 2 numbering system, where each digit represents on bit of information (0 or 1).

Bit - The smallest unit of computer knowledge: yes or no, true or false, on or off, a bit is a single unit of information.

Boolean Algebra - A system of mathematics used to prove logical statements and theories, primarily dependant on the Binary, or base 2 numbering system.

Booting - The start-up process a computer undergoes when turned on or reset. Generally, booting consists of several processes: POST, initializers, and loading of the operating system.

Boot-Up - See 'Booting'.

British Thermal Unit - 0.252 Kilocalories, or the thermal energy necessary to raise 1 pound of water from 39.2 to 40.2 degrees Farenheit at standard temparature and pressure.

Browser - A window that allows a user to examine all or part of a database in a vertical format: each record is represented on one or more lines. Browsers may be used to edit information, perform additions, deletions, or other functions assigned as keys to the browser, and may also load a data entry window for individual record editing.

BrowseList - A hybrid function, combining the field-filling capabilities of a PickList with the record-editing options of a Browser. The default BrowseList key on data entry fields is <F4>.

Byte - The smallest basic unit of computer information that can be stored. For PC's, a byte consists of 8 'bits'. 8-bit bytes contain numeric values between 0 and 255, inclusive.

Cache - A small, fast storage space. On CPU's, a cache is a small amount of very-high-speed memory dedicated to the processor, which 'pre-fetches' software from regular RAM, before the processor needs the instructions, reducing the amount of time wasted waiting for information from Chip Memory. For hard-drives, a cache is a small amount of RAM, which either 'pre-fetches' (read-ahead) or 'remembers' (write-storage) information on the disk, so that the program doesn't waste as much time waiting for the hard drive.

Command - A statement tht tells the computer what to do at a given point. In DOS, each command is on given on a separate line, separated by a carriage return (code 13, or the <Enter> key).

Command Line Interface - In VariForm, the Command Line Interface is a pair of Windows that allow the user to send individual commands to the CommandLine Processor. One window in the interface displays the values of all currently loaded Virtual Fields, and the other provides a single data-entry field: saving the latter screen sends the command to be processed. Escaping or saving a blank line returns the user to whatever mode or screen was used to call the CLI.

CommandLine Processor - In VariForm, the CommandLine Processor is the 'engine' of the procedural system. This interprets and acts on individual commands and procedure statements sent in from Function Keys, Procedure Calls, Searches, or the CLI.

Coprocessor - A chip or expansion board that assists the Central Processor Unit. Coprocessors generally perform specialized functions, such as video graphics or floating point math, and do so at far greater speeds than the CPU could perform them. An example is the set of math coprocessors for the PC family of CPU's:

8086 (PC-1, PC/XT) uses an 8087
80286 (AT) uses an 80287
80386 uses an 80287 or an 80387
80486SX uses an 80387-2 or an 80487
80486DX has one built into the chip as a separate circuit

Cursor - A small block showing where the next character that is typed will appear. In the VariForm Data Entry System, the size of the cursor also indicates typing mode: a large cursor indicates "Insert" mode, while a small cursor indicates "OverWrite" mode.

Cursor Bar - On VariForm Browsers and Menus, a row or group of rows that are colored differently from the surrounding rows. The cursor bar indicates which record will be edited, or which selection will be chosen if the user presses <Enter>. Cursor bars can be moved from selection to selection by using the <arrow> keys, or the <page> keys.

Database - A file or files of information on a computer. Databases are organizations comprised of 'records'.

Database Management System - A program that allows access to and modification of databases.

Database Relations - Connections between physically separated, individual databases, that allow them to act together. This is done be creating 'relationships' that attach one or more records in one database to one or more records in another database. The four types of database relationships are: One to One, One to Many, Many to One, and Many to Many.

Data Entry - The act of supplying information to a database by manually typing the information into a window that is acting as a 'data entry screen'.

Device Driver - A small program that loads and becomes part of the operating system. Device drivers either provide access to hardware elements that DOS could not otherwise access (such as networking devices or Mice), or enhance DOS's functionality (ram drives, EMM programs, disk caches). Most device drivers are loaded at boot-up time by a command such as DEVICE= or DEVICEHIGH= in the Config.Sys file, or INSTALL= or INSTALLHIGH= in the Autoexec.Bat file.

Disk Drive - A mechanical device that provides permanent storage of information.

Display Field - In VariForm, an area that 'displays' or shows the contents of a field within a record from a database. A display field controls the size, format, and content of a field, as it appears on a data-entry screen, within a database browser, or on a report.

Display Structure - In VariForm, the 'master control structure' for a particular view of a record within a database. The display structure contains references for color schemes, global function keys (within the display), help topics, upper-line help references, and contains a 'group name' that refers to one or more display fields that will be used within the display structure.

Display System - A generic term refering to one or more Display Structures and their respective Display Fields.

English Measures - non-metric units of measure, including averdupois and troy weight measures, comprised of the inch, foot, yard, rod, acre, hectare, mile, ounce, pound, pint, quart, and gallon.

Extended Memory - Memory in the Expanded Memory area (above 1 MegaByte), which is managed and controlled by an 'EMS' manager, such as EMM386. Also refered to as EMS or LIM memory.

Expanded Memory - All physical memory above 1 Megabyte. This is the memory which is not accessible to a PC, or to a larger CPU running in Real Mode without a memory manager program.

Expression - A single line or command that can be sent to the CommandLine Processor. Alternatively, a logical 'statement', which may (in Search Records) cover several lines which eventually evaluate to a single true or false value.

Field - A fixed portion of each record within a database, containing information of a particular size and type. Also called a 'segment of data within a record'.

Flat File - A text file, or one where no binary (non-displayable) characters are present. Numbers are shown as digit characters, instead of the computer's native storage format.

Floating Point - A number capable of containing a fraction. In VariForm, all 'floating point' values are 32-bit IEEE double-precision numbers, which are generally accurate to 15 digits.

Floppy Disk - A piece of flexible material on removable media. Allows for permanent storage of information on a computer, as well as transfer of information between computers by exchanging floppy disks. Sizes of floppy disks are 2", 3½", 5¼", 8", and disks come in single, double or high, and quadruple density. Storage capacities range from 120 Kb to 4 Mb, with the most common storage range being between 360 Kb and 1.44 Mb.

GigaByte - 1024^3 bytes, or approximately 1 billion bytes.

Grow an Index - The act of resizing an index to use more RAM, by either increasing the number of index buffers (single user mode), or by increasing the size of the index buffer records (single or multiuser mode).

Hard Disk - A non-removable mass-storage device, capable of permanently recording information for a computer or network. This is the most common form of permanent storage on Personal Computers, with storage capacities ranging from 10Mb to over 1Tb.

Hayes - A standard communications protocol implemented by many Modems, the Hayes standards are comprised of a series of "AT" commands which direct the modem on how to configure, how to dial out or receive a call, and when to break the connection.

Hex or Hexadecimal - The base 16 numbering system, where each digit represents 4 bits, or a nybble of information.

Index - A file used to organize or 'sort' records within a database. Indexes allow database records to be viewed in orders other than the order in which they were entered. This is done by keeping an ordered list of key fields and the corresponding physical record numbers in a separate file.

Integer - A whole number. On computers, an integer is a number that will fit in a word of space. On PC's, an integer may have values of 0 - 65,535, or -32,768 - 32,767.

Key Field - A field or assembly of fields that are used within an index. Also, a field used in the definition of a relationship between two databases.

KiloByte - 1024 bytes.

Long Integer - On computers, a long integer is a whole number that will fit in two words of space. On PC's a long integer may have values of 0 - 4 billion, or -2 bil. - +2bil.

Logical Expression - An expression or formula that evaluates to true or false, such as "A == 4", which is only true if a variable 'A' exists which is exactly equal to 4.

Lotus Intel Microsoft Specifications - A series of standards for expanded memory management that made memory above 1 Mb available to most programs through a standard set of memory management commands. Most LIM-EMS programs provide services at the 'version 3.0' or 'version 4.0' level, which is what VariForm requires to run in "Overlay Mode".

Many to Many - A complex, 'grouping' relationship, where one or more records on a database are related to one or more records in a parallel, or 'brother' database. While this relationship is the hardest to exemplify, it is the most useful for complex, temporary associations or records. Most examples of many to many relationships are involved in complex set theory, which is beyond the scope of this manual. Because this is a complex relationship, deleting one of the brother's records will usually leave the other brother records in both databases unchanged.

Many to One - A complex, 'backward' relationship where one or more records in a 'parent' database are related to a common 'ancestral' or 'child' record. This is essentially the reverse view of a One-to-Many database. Because this is a complex relationship, deleting one of the parent records will usually leave the other parent records and the child record unchanged.

MegaByte - 1024^2 bytes, or approximately 1 million bytes.

Metric System - An alternative (non-'English') system of measures based on grams, meters, liters, and degrees centigrade.

Modem - Modulator/Demodulator. A device that allows computers to communicate through telephone systems by converting (modulating) digital serial signals into tones, and demodulating the incoming tones back into digital signals for the computer.

Nested Field - A field on a Data Entry Screen which displays 'Press Enter' when the field is highlighted / made current. In Data Entry, pressing <Enter> on this field will access another data entry or browser system, for subscreens, and access to related databases. In Reporting,

nested fields call sub-reports.

Nybble or Nibble - 4 bits of information. The amount of information that can be represented in 1 hexadecimal digit.

Octal - The base 8 numbering system, where each digit represents 3 bits of information.

One to Many - A direct relationship between two databases, where a single record in on database, called the 'parent' database, can have one or more corresponding records in another database, called the 'child' database. An example of this relationship would be a database of customers having individual sales records as a child database: each customer may have one or more sales directly related to it. If this is a Mandatory or Automatic relationship, then deleting one record will automatically cause the matching record to be deleted.

Appendix 9: Definitions

One to One - A direct relationship between two databases, where a record in one database does (or can) have a single corresponding record in another database. An example of this relationship would be a database of names related to the associated addresses for those names. One to One databases can (and often should) be 'merged' into a single, comprehensive database. If this is a Mandatory or Automatic relationship, then deleting one record will automatically cause the matching record to be deleted.

Parallel Port - A 25 or 36 pin connector that can connect a computer to a printer. Usually, these ports are either unidirectional (send only), or mostly send-only (1 or 2 readable lines).

Port - Another name for the sockets and connectors that connect a computer to other devices, such as keyboard, mouse, printer, modem, et cetera. Also refers to specific locations in memory that can affect or report upon a computer's hardware status.

Procedural - When referring to a programming language, 'procedural' refers to the availability of flow-control commands, and their ability to call other procedures by a name. Non-procedural languages require the use of line numbers and create the potential for 'spaghetti code'.

Procedure - A set of one or more command lines, grouped together by a common 'procedure name'.

Protected Mode - for 80286 (AT) and larger CPU's, Protected Mode is the mode of operation that provides full access to memory: systems can access 16 MegaBytes or more of 'Real Memory', and a billion bytes of 'Virtual Memory'. The disadvantage to protected mode is that programs written for real mode operation will not run in protected mode. OS/2, for example, runs in Protected Mode.

Real Mode - for 80286 (AT) and larger CPU's, Real Mode is the mode of operation that provides compatibility with the original 8086 processor. MS-Dos, for example, runs in 'real mode', so that software written for the older computers would work on the newer equipment. The disadvantage to real mode is that memory above 1 Megabyte requires additional memory-management programs, such as XMS and EMS managers.

Reboot - To reset or restart a computer; often used when changing 'startup settings', such as network locations or memory usage. Typically, Rebooting is performed from the DOS prompt by pressing <Ctrl-Alt-Delete>.

Record - A single block of information within a database. Records may consist of one or many fields, and are most commonly 'fixed size', or uniform-length sections of the database file.

Relational Database - A database that has 'relationships', or connections to other databases. These connections are usually identified by matching values in 'Key Fields' between the two databases.

Resident - Loaded in memory, such as a TSR program, or a mouse driver.

Restart - In VariForm, restarting means quitting to Dos and running VariForm. Most commonly used when changing startup parameters, such as user settings, "Main" group virtual fields, or

AutoLoaded databases.

Reset - Rebooting the computer by using the [Reset] button on the computer.

Scan Codes - The internal numeric codes that are used to tell the computer when a key on the keyboard is depressed (Make Code) or released (Break Code).

Screen - A complete unit of display information, consisting of all display fields or commands loadable under one displayfield group. A screen may be bigger than its allocated window, such as in 'scrolling data entry windows', or smaller than the window, as is the case with browsers in data entry mode.

Serial Port - A 9 or 25 pin connector that can attach a computer to a mouse, modem, or other serial device. While serial ports are generally slower than parallel ports, they are more commonly bidirectional, and more commonly used.

Shadow Memory - RAM memory that is used to keep a copy of BIOS in faster-access memory. Often an option on 32+ bit systems (80386, 80486, Pentium), Shadow memory can increase system performance by allowing the BIOS code to reside in conventional or expanded memory, where it can be accessed more quickly than through the BIOS chips themselves.

Shrink an Index - The act of resizing an index to use less RAM, by either decreasing the number of index buffers (single user mode), or by decreasing the size of the index buffer records (single or multiuser mode).

Statement - A single command in a search or procedure that causes VariForm to perform an action, such as running another procedure, running a search, or putting a value in a Field.

TeraByte - 1024^4 bytes, or approximately 1 Trillion bytes.

Terminate and Stay Resident - This identifies programs that 'load' into memory after the computer has booted, such as mouse programs, and computerized 'scratch pads'.

Upper Memory Block - A block of 'upper memory', which is the area between 640 Kb and 1024 Mb. This area of memory can be used by some device drivers and TSR programs instead of conventional memory.

Vapor or Vaporware - Nonexistant. Used when referring to a product that has been announced before it can be delivered.

Window - A section of the computer's display which is dedicated to one type of display or function, such as browsing or data entry.

Word - On any computer, the standard minimum multi-byte configuration for whole numbers. On PC systems, a word is 16-bits, or 2 bytes long.

XBase - A standard for both flat database files and a database programming language. Used in dBase, Clipper, Foxpro, and some other database programs.

Recommended Reading

For more VariForm Information

Advanced VariForm Programming Styles and Structures: The philosophy of compiled data, (1994) by Richard F. Parker, from Parker Software. [Comparative illustrations on sophisticated user interfaces, needs determination, complex database relationships]

Hardware References

The Winn Rosch Hardware Bible, (1989) by Winn L. Rosch, from Brady Publishing and Simon and Schuster, Inc. ISBN 0-13-160969-3 [History of the P.C. design, peripheral hardware systems and architectures]

Software/Programming References

Ansi Fortran 77: An Introduction to Structured Software Design, (1983) by William C. Brown Company Publishers, College Division. ISBN 0-697-08167-2 [Basic Programming Concepts and Introduction to Algorithms]

Digital Logic and State Machine Design, (1984) by David J. Comer, from CBS College Publishing. ISBN 0-03-063731-7. [Basic and Advanced concepts of Digital and Boolean Logic Systems]

File Formats for Popular PC Software, (1986) by Jeff Walden, from John Wiley and Sons, Inc. ISBN 0-471-83671-0. [Detailed information on Lotus, Symphony, dBase, DIF, and VisiCalc data file formats, for those who want to write their own data-conversion utilities in VariForm]

Additional References

The Great International Math on Keys Book, (1976) by the Staff of the Texas Instruments Learning Center. ISBN 0-89512-002-X. [Information on practical applications of basic math principles]

Acronyms & Definitions: A PC user's survival guide, (1994) by Richard F. Parker, from Parker Software. [An extensive list of jargons, abbreviations, and 'computerese']

Other VariForm Modules

Why Start from Scratch?

Telecommunications

Async Script Extensions: People reach out through modems and telecommunicate. This module provides a sophisticated terminal emulation system, as well as extensions to the CommandLine system to provide both manual and "batch" automated communications. Several file transfer protocols supported, including XModem, YModem, ZModem, Kermit, UUCP, and others. Available in late May, 1994.

Multiport Async Script Extensions: You can create a complex, customizable "Electronic Bulletin Board System", to help keep buyers and suppliers apprised of current information. Includes support for 4 - 32 lines, plus the console, conferencing, usage statistics, electronic mail, and unlimited menuing and file transfer areas. Available in July, 1994.

Graphic Systems

VariForm Graph Pack: Draw sophisticated graphs and charts using source information from VariForm databases, or flat files from other programs through the graph pack. Support for Hercules, CGA, EGA, VGA, SVGA monitors, with several dithering and antialiasing schemes. Create computerized 'slide shows' for multimedia business presentations, or print out sophisticated grayscale or multicolor reports. Scheduled for release in January, 1995.

Windows / OS/2 User Interfaces: Access 32-bit processing power from our support for either MicroSoft Windows or OS/2. This enhances VariForm's speed, and memory access. Other features in the GUI interface include mouse support, movable icon menus, sliding 'browser bars', along with all of the features in the Graph Pack (above). Scheduled for release in May, 1995.

Sophisticated Mathematics

VariForm Spreadsheet: VariForm Spreadsheets provide unlimited 'rows' and 'columns' by providing all spreadsheet services within a virtually structured database. 'Imports' and 'Exports' are simple, because the spreadsheet data is stored in VariForm database records. Direct import and export with other commercial spreadsheets makes the transition to VariForm spreadsheets quick and easy. Why get a spreadsheet that acts like a database, when you can get a spreadsheet that is part of a database system? Available in July, 1994.

Statistical Math Pack: Statisticians and Business Analysts often need more functions than are available in a common spreadsheet. VariForm Statistics package provides several powerful extensions for both the Database System and the Spreadsheet, including high-precision (128 bit: $10^{\pm 100}$ and 30 decimal places) calculator, Standard and Programmable Deviations, Populations, Complex Mean Calculations, RMS evaluators, and derivative generators. Available in July, 1994.

Complex Math Pack: This system is for the serious engineer who needs provable calculations in a timely fashion. Features include Arrayed variables, Finite Element Analysis, Derivatives, Matrix Determinants (beyond 7th degree), Integral Approximators, and much more. Available

in August, 1994. (Includes Spreadsheet and Statistics).

Other VariForm Modules

Why start from scratch?

Business Information Systems

Point Of Sale & Inventory System (POSIS): Dramatically reduce retail sales management workloads. POSIS provides a full range of features, from basic cash-register work to year-end inventory accounting. Built-in analytical procedures provide up-to-the-minute figures for cash, cashflow, stock hold time, reorder advisory analysis, sales trends, and sales projections. POSIS also provides a system for developing and managing personnel schedules, eliminating the rescheduling work that keeps the manager off the sales floor. Currently available.

Accounting System: Cash or Accrual, this 9-module system can reconcile bank statements, generate and maintain an unlimited ledger system, and consolidate all information into a variety of consolidation and review reports. Modules include Cash and Master Ledger, Income and Receivables, Expensing and Payables, Journal/Ledger Builder, Billing and Dunning System, Tax and Liability Calculators, Payroll Management, Corporate Periodic Reports, and Business Analysis Tools. Available in September, 1994.

Bar Code Reader/Writer Systems: Bar Code Systems help eliminate repetitive data entry for Inventory, Point of Sale, or Shipping/Receiving database systems. Bar codes also improve reporting accuracy, since the scanner is actually doing the data entry. Our Bar Code Reader/Writer module interacts with most commercially available bar-code scanners and 'wands', as well as being able to generate bar code labels in 12 standard formats on most dot-matrix and laser printers. The inputs can be used as automated data entry into a transaction processing or point of sale style system. Available in September, 1994.

Other VariForm Modules

Why start from scratch?

For the Technically Oriented

Full DataLink Pipeline: scripts and pre-defined structure analyzers for lotus, symphony, dBase, Paradox, WordPerfect, and over 80 other file structures. File structure analysis tools to design interfaces to map information to, from, or between one or many databases. SQL Interface and Bidirectional XBase language translation routines are also part of this module. Available in August, 1994.

EDI Translation Interface: bring in EDI files to our generic variable-record-size format, where they can be analyzed, reported on, and exported to your in-house systems. Take in-house information and generate full and complete X.12, EDIFACT, or TDCC documents in just a few keystrokes after layouts. Includes sample templates and template database system. Available in July, 1994.

Data Compression/Archival Database Systems: Compress data to save on disk space, without losing access to information. A variety of choices in compression level to determine space/speed trade-off, or 'off-load' to removable-media-based data files. Available in July, 1994.

Compound Security Systems: Concerned that our standard encryption might not be enough? Our compound module adds DES, SuperCrypt, and TripleDES encryption schemes to make detranslation by unauthorized personnell virtually impossible. Optional security 'overloading' feature quadruples the number of available security access levels, as well. Available in June, 1994.

Screen Saver: Design your own picture, select from our list of 35 photos, or select a 'random play' to protect screens and users' faces when the computer is not in use. Available in June, 1994.

Protected Mode Extender: Don't want or like OS/2 or Windows? Wondering what to do with those extra megs of upper memory? Our Protected Mode Extender provides maximum database access to VariForm, or up to 610 K available to other programs loaded into a VariForm Shell. Take full advantage of your 386, 486, or Pentium. Scheduled for October, 1994.

Index

