Command Reference

ACIUS

4th DIMENSION™

Acius

4th Dimension™ Command Reference

4th Dimension by Laurent Ribardière

This manual was written by Dominique Hermsdorff, Will Mayall, Bruce Barrett, Bill Kling, and Samir Arora.

Cover design by Patrick Chédal C & C.

Contents

Figures and tables

Preface

About This Book

This section gives an overview of this reference, presents typographical conventions, and lists all syntactic metasymbols used in this volume.

Overview of the command reference

This book includes

- □ Part I, "Introduction to the Command Reference." The 4th Dimension command set is presented by category with a brief description of each command.
- □ Part II, "The Commands in Alphabetical Order." Each command appears with a syntax statement, a description, one or more examples, and references to other commands.
- □ Appendixes.

Aids to understanding

Look for these visual cues throughout this book:

❖ *Notes:* Text set off in this manner presents sidelights or interesting pieces of information.

Important

Text set off in this manner presents important information that you should keep in mind.

Warning

Warnings like this indicate potential problems or situations where you could lose data.

The command reference uses a special typeface for procedure listings, command names, filenames, and variable names. For example:

In this case, SAVE VARIABLE wrote MyVar to a document on disk named MemVar.

In syntax statements, italic is used for metasymbols. For example:

GO TO FIELD (*fieldname*)

Vocabulary

A built-in 4th Dimension *command* simply does a task, like sorting. Commands always appear in all capital letters. For example, SORT SELECTION. A built-in 4th Dimension *function,* on the other hand, does a task and returns a value. Functions always appear with an initial capital letter. For example, Records in file. The procedure editor groups 4th Dimension commands and functions together in a window under the term *Routines*. It groups control of flow and assignment operators as *Keywords* in another window.

When referring to user-written code, *routine* means any programming entity you might create. (Developer-created routines appear in italic type at the end of the list of 4th Dimension Routines in the procedure editor. Externally written and compiled routines appear in bold italic.) When referring specifically to a developer-created procedure or function, the book uses the term *procedure* or *function,* accordingly.

The term *numeric* refers to any data object on which you can perform arithmetic. Thus, *numeric* comprises the data types Real, Integer, and Long Integer.

Syntactic definitions

This manual uses a consistent set of metasymbols to express command arguments. This section gives you the symbols and syntactical expressions used throughout this guide. Metasymbols appear in italic throughout this book. See Tables P-1 and P-2.

Table P-1
Syntactic symbols

Symbol	Description	Example
*	Repeat preceding statement up to last required statement.	{*statement*}«{;*}»
{ }	Repeatable any number of times	{*statement*}
« »	Optional argument	**BEEP**«(*posintexpr*)»
|	One (and only one) of two options	*var* | *subfieldname*
...	Intervening code	**While...End while**

Note: Both the asterisk (*) and the vertical bar (|) also appear as arguments and operators.

Table P-2
Syntactic metasymbols

Metasymbol	Description	Example
boolexpr	Boolean expression	**If** (*boolexpr*) . . . **End if**
buttonvar	Button variable	**BUTTON TEXT** (*buttonvar*;*strexpr*)
date	4th Dimension date	**Day of** (*date*)
docname	Desktop document name	**DELETE DOCUMENT** (*docname*)
expr	An expression of any type	**SEARCH BY INDEX** « (*fieldname*{= | ±}*expr*«{;*}») »
fieldname	Name for field	**CREATE LINKED RECORD** (*fieldname*)
filename	Name for file	**ADD RECORD**« (*filename*) »
intexpr	Integer expression	**TRUNC** (*numexpr*;*intexpr*)
numexpr	Numeric expression	**Arctan** (*numexpr*)
numvar	Numeric variable	**GET HIGHLIGHTED TEXT** (*var* | *fieldname*;*numvar1*; *numvar2*)
picturexpr	Picture expression	*picturexpr* + *numexpr*
posintexpr	Positive integer	**BEEP**« (*posintexpr*) »
strexpr	String expression	**Ascii** (*strexpr*)
strvar	String variable	**GRAPH** (*var*;*posintexpr*;*strvarX*;*numvarY*)
statement	Logical line of code	**APPLY TO SELECTION** («*filename*;» *statement*)
subfieldname	Name for field in a subfile	**Squares sum** (*subfieldname*)
subfilename	Name for a subfile	**End subselection** (*subfilename*)
var	Any variable	**Undefined** (*var*)

Part I

Introduction to the Command Reference

This part breaks down 4th Dimension commands by category. Some commands appear in multiple categories.

The command categories are

- □ selection commands
- □ working with records
- □ working with subrecords
- □ working with links
- □ working with sets
- □ data entry
- □ data output
- □ user interface
- □ communicating with the outside world
- □ multi-user commands
- □ standard functions
- □ programming

Selection commands

These commands work on a selection of records.

Selecting a file

In a multiple-file database, you can specify which file's records you want to work with and determine the number of records in the file.

- □ DEFAULT FILE declares a file as the default file.
- □ Records in file returns the number of records in a file.

Manipulating a selection

These commands modify the data in a selection, the number of records in the selection, or the order in which the records appear.

- □ APPLY TO SELECTION applies a statement to each record in the current selection.
- □ DELETE SELECTION deletes all records in the current selection.
- □ MODIFY SELECTION displays the current selection, using the default output layout. Users can select one or more records for modification.
- □ Records in selection returns the number of records in the current selection of a file.
- □ SEARCH searches through all records in the file for records matching selection criteria.
- □ SEARCH BY INDEX searches through all records in the file for records matching selection criteria for an indexed field(s).
- □ SEARCH SELECTION searches through the current selection for records matching selection criteria.
- □ SORT SELECTION sorts the current selection of the file into ascending or descending order. You can sort on multiple fields with one SORT command.
- □ SORT BY INDEX sorts all records in the file into ascending or descending order. Only one sort field is permitted, and it must be indexed.

Working with records

The commands and functions in this section deal directly with records: how to select and manipulate them.

Selecting records

To create a selection, you must select records from all records in the file or from a subset of them. ALL RECORDS, ONE RECORD SELECT, and all three search commands automatically load the first record and make it the current record of the current selection.

- ☐ ALL RECORDS makes the current selection all records in the file.
- ☐ Before selection returns TRUE when you try to move the record pointer before the first record of the current selection.
- ☐ End selection returns TRUE when you try to move the record pointer after the last record of the current selection.
- ☐ FIRST RECORD moves the record pointer to the first record in the current selection, loads the record, and makes it the current record.
- ☐ LAST RECORD moves the record pointer to the last record in the current selection, loads the record, and makes it the current record.
- ☐ NEXT RECORD moves the record pointer to the next record in the current file, loads the record, and makes it the current record.
- ☐ ONE RECORD SELECT makes the current record (usually a record popped from the stack with POP RECORD) the current selection.
- ☐ POP RECORD takes the top record off the file's record stack and makes it the current record, although not necessarily of the current selection. ONE RECORD SELECT makes this record the current selection.
- ☐ PUSH RECORD pushes the current record onto the file's record stack.
- ☐ PREVIOUS RECORD moves the record pointer to the previous record in the current selection, loads the record, and makes it the current record.
- ☐ SEARCH searches through all records in the file for records matching selection criteria.
- ☐ SEARCH BY INDEX searches through all records in the file for records matching selection criteria for an indexed field(s).
- ☐ SEARCH SELECTION searches through the current selection for records matching selection criteria.

Manipulating records

These commands deal with adding, modifying, and deleting an individual record.

- ☐ ADD RECORD adds a blank record to the current file, presents the default input layout to the user, and saves the record to the current selection as the current record when the user validates the record.
- ☐ CREATE RECORD adds a blank record to the file, loads it into memory and makes it the current record of the current selection. You must issue a SAVE RECORD command to save the record.
- ☐ DELETE RECORD deletes the current record from its file.
- ☐ MODIFY RECORD presents the default input layout and displays the values in the current record. The user can modify these values and save the record to the current selection as the current record by clicking an Accept button.
- ☐ SAVE RECORD saves a record usually created with CREATE RECORD.

Working with subrecords

The commands and functions in this section deal directly with subrecords: how to select and manipulate them.

Manipulating a subselection

These commands modify the data in a subselection, the number of subrecords in the subselection, or the order in which the subrecords appear.

- ☐ APPLY TO SUBSELECTION applies a statement to each subrecord in the current subselection.
- ☐ Records in subselection returns the number of subrecords in the current subselection of the current parent record.
- ☐ SEARCH SUBRECORDS searches only through the current subselection of the current record for subrecords matching search criteria.
- ☐ SORT SUBSELECTION sorts all subrecords in the current subselection of the current record into ascending or descending order. You can sort on multiple fields with one SORT SUBSELECTION command.

Selecting subrecords

To create a subselection, you must select subrecords from all subrecords belonging to the current record of the the file or from a subset of them. A subrecord selection cannot include records from more than one parent record.

- ALL SUBRECORDS puts all subrecords belonging to the current record into the current subselection. It automatically loads the first subrecord and makes it the current subrecord of the current subselection.
- Before subselection returns TRUE when you try to move the subrecord pointer before the first subrecord of the current subselection of the current parent record.
- End subselection returns TRUE when you try to move the subrecord pointer after the last subrecord of the current subselection of the current parent record.
- FIRST SUBRECORD moves the subrecord pointer to the first subrecord in the current subselection and makes it the current subrecord.
- LAST SUBRECORD moves the subrecord pointer to the last subrecord in the current subselection and makes it the current subrecord.
- NEXT SUBRECORD moves the subrecord pointer to the next subrecord in the current subselection and makes it the current subrecord.
- PREVIOUS SUBRECORD moves the subrecord pointer to the previous subrecord in the current subselection and makes it the current subrecord.

Manipulating subrecords

These commands deal with adding, modifying, and deleting an individual subrecord. Changes to the file or subfile are only recorded if you save the parent record.

- ADD SUBRECORD creates a blank subrecord in memory and makes this subrecord the current subrecord. It then presents the specified input layout.
- CREATE SUBRECORD creates a blank subrecord for the current subfile and makes it the current subrecord of the current subselection.
- DELETE SUBRECORD deletes the current subrecord from its subfile, leaving the current subselection empty.
- MODIFY SUBRECORD presents a specified input layout and displays the values in the current subrecord, so the user can modify these values.

Working with links

The commands and functions in this section work with links created between fields. When using a link, you can read data from a linked field and write data to it.

- □ ACTIVATE LINK searches a linked file for a value matching its field argument and creates a link to the linked file. It does this without affecting the current selection or current record of the linked file. It does not load a record.
- □ CREATE LINKED RECORD creates a new linked record in the linked file.
- □ LOAD LINKED RECORD finds the specified linked record, loads it into memory, and makes it the current record for the linked file.
- □ LOAD OLD LINKED RECORD finds the specified record based on the old link, loads it into memory, and makes it the current record for the linked file.
- □ Old returns the value of a field before it was modified.
- □ SAVE LINKED RECORD saves the linked record pointed to by its field name argument.
- □ SAVE OLD LINKED RECORD saves the record pointed to by its field name argument, using the old link.

Working with sets

A set is an object in memory containing one bit for each record in the file the set belongs to. 4th Dimension has a set named UserSet that contains the most recent user selections made by clicking on records displayed by DISPLAY SELECTION or MODIFY SELECTION.

- □ ADD TO SET puts the current record in the set.
- □ CLEAR SET deletes a set from RAM, freeing the memory it used.
- □ CREATE SET creates a set comprising all records in the current selection.
- □ CREATE EMPTY SET creates an empty set.
- □ DIFFERENCE compares *set1* and *set2* and places the unique elements in *set1* in *set3,* which it creates.
- □ INTERSECTION compares *set1* and *set2* and places only their common elements in *set3,* which it creates.
- □ LOAD SET reads a set from disk.
- □ Records in set returns the number of records in a specified set.
- □ SAVE SET saves a set to disk.
- □ UNION creates *set3,* containing any element in *set1* or in *set2* or in both.
- □ USE SET creates a current selection containing all records in the set.

Data entry

The commands in this section refer to handling data entry from the keyboard to an input layout or a dialog layout and not through telecommunications.

❖ *Input layouts and dialog layouts:* Input layouts take input through fields only, but can display variables. Dialog layouts take input through variables only, but can display fields. Thus, commands referring to fields only can only be applied to input layouts, not to dialogs.

- □ After returns TRUE when the user clicks an Accept button or presses the Enter key in an input layout.
- □ Before returns TRUE before a layout is displayed.
- □ DIALOG displays the specified layout for taking data through variables.
- □ During returns TRUE when the user does an activity such as modify a field and move to another field.
- □ GET HIGHLIGHTED TEXT gets the positions of the first and last character of the user selected text.
- □ GO TO FIELD places the insertion point in a specified field.
- □ HIGHLIGHT TEXT either places the insertion point or highlights text.
- □ INPUT LAYOUT sets the default input layout for a file.
- □ Modified returns TRUE when a user modifies the specified field.
- □ Old returns the value a field held before it was modified.
- □ REDRAW forces a screen update during input.
- □ REJECT on a field prevents the user from leaving the field or, in a second syntax, rejects the entire record.

Data output

The commands in this section refer to outputting data to the screen or to a printer and not through telecommunications.

- **Before** returns TRUE for each record, before printing the current record.
- **DISPLAY RECORD** displays the current record in the current input layout only as long as another event does not repaint the screen.
- **DISPLAY SELECTION** displays all records in the current selection, using the default output layout.
- **During** returns TRUE for each record, before printing the current record.
- **FONT** specifies a font for displaying a variable in a layout.
- **FONT SIZE** specifies a font size for a variable in a layout.
- **FONT STYLE** specifies a font style for a variable in a layout.
- **FORM FEED** issues a form feed character, pushing the paper to the next top of form position.
- **GRAPH** graphs data from subfiles and/or variables to a graph area in a layout.
- **GRAPH FILE** graphs data from selected fields to the graph window.
- **In break** returns TRUE when a sort level changes.
- **In footer** returns TRUE at the end of each page.
- **In header** returns TRUE at the beginning of each page.
- **Level** returns the current break level.
- **MODIFY SELECTION** displays the current selection, using the default output layout. Users can select one or more records for modification.
- **OUTPUT LAYOUT** specifies the default output layout.
- **PRINT LABEL** prints the current selection as labels, using the default output layout and issues a form feed after printing the last record.
- **PRINT LAYOUT** prints a specified layout, including the values of its fields and/or variables. Used for printing one document composed of multiple layouts. It issues no form feed.
- **PRINT SELECTION** prints all records in the current selection, using the default output layout and issues a form feed after printing the last record on each page.
- **PRINT SETTINGS** displays the two standard print dialog boxes on the screen.
- **REPORT** prints a specified report or brings up the Quick report dialog for the user to specify the elements of the report.
- **Subtotal** returns the sum of items in a field for the current break level only.

User interface

The section presents commands for managing the user interface. Its categories include

- □ menus
- □ layouts and layout variables
- □ dialogs and messages
- □ windows
- □ miscellaneous

Menus

These commands manipulate menu bars, menus, and menu items.

- □ CHECK ITEM checks or unchecks a specific menu item.
- □ DISABLE ITEM disables a specific menu item.
- □ ENABLE ITEM enables a disabled menu item.
- □ MENU BAR puts up the specified menu bar.
- □ Menu selected returns the menu and menu item that a user selected.

Layouts and layout variables

These commands are important for managing layouts and layout variables.

- □ BUTTON TEXT specifies text for a layout button or check box variable.
- □ DIALOG displays a layout for data entry through variables.
- □ DISABLE BUTTON disables a layout button or check box variable.
- □ ENABLE BUTTON enables a disabled layout button or check box variable.
- □ FONT specifies a font for displaying a variable either in a layout or a report.
- □ FONT SIZE specifies a font size for a variable in a layout.
- □ FONT STYLE specifies a font style for a variable in a layout.
- □ GRAPH draws a graph in a graph layout variable.
- □ INPUT LAYOUT specifies which layout to present for adding or modifying records or importing files.
- □ INVERT BACKGROUND inverts the background for a displayed variable.
- □ OUTPUT LAYOUT specifies the default output layout.
- □ REDRAW updates the display after changes to a variable, a scrollable array, or an included layout.

Dialogs

Use these commands to communicate with the user.

- □ ALERT presents a simple dialog box with just one button: OK.
- □ CONFIRM presents a dialog box asking the user to accept or cancel a course of action by clicking OK or Cancel.
- □ DIALOG displays a layout for data entry through variables.
- □ MESSAGE presents a message box with no buttons.
- □ REQUEST presents a dialog box asking to enter data. It includes OK and Cancel buttons.

Windows

These commands create, manage, and close a custom window.

- □ CLOSE WINDOW closes the custom window.
- □ ERASE WINDOW clears the contents of the custom window and homes the cursor.
- □ GO TO XY positions the cursor at a specified point in the current custom window.
- □ OPEN WINDOW opens a custom window of prescribed dimensions on the screen.
- □ Screen height returns the height in pixels of the screen.
- □ Screen width returns the width in pixels of the screen.
- □ SET WINDOW TITLE sets the window title for the current window.

Miscellaneous

These commands also help with the work of managing a clean interface.

- □ BEEP generates a tone, the duration of which you can specify.
- □ Current password returns the user's password.
- □ MESSAGES OFF suppresses 4th Dimension's progress graphics.
- □ MESSAGES ON turns on 4th Dimension's progress graphics.
- □ ON ERR CALL installs a procedure for trapping and responding to errors.
- □ ON EVENT CALL installs a procedure for responding to key presses and mouse clicks.

Communications with the outside world

The first part of this section covers the six commands for direct importing and exporting data in DIF, SYLK, and text (ASCII) documents. In other circumstances, as when subfiles are involved, you may have to use a different set of commands. The "Communicating With Documents and Ports" section covers these.

Importing and exporting a selection

With these commands, you can import and export data in three different formats: DIF, SYLK, and text (ASCII). Export commands use the default output layout, and import commands use the default input layout.

- □ EXPORT DIF writes the current selection to a DIF file.
- □ EXPORT SYLK writes the current selection to a SYLK file.
- □ EXPORT TEXT writes the current selection to a text file.
- □ IMPORT DIF reads the specified DIF file and makes it the current selection.
- □ IMPORT SYLK reads the specified SYLK file and makes it the current selection.
- □ IMPORT TEXT reads the specified text file and makes it the current selection.

Communicating with documents and ports

With these commands, you can create, write to, and read record-oriented and other documents. You can also exchange 4th Dimension records and variables between your database and a document or another Macintosh™ computer running 4th Dimension.

- □ DELETE DOCUMENT deletes a document (including those created by other applications) from disk.
- □ LOAD VARIABLE loads a variable into memory from a document on disk.
- □ ON SERIAL PORT CALL installs the procedure to be called if serial port activity occurs.
- □ RECEIVE BUFFER assigns whatever is in a serial port buffer to a variable.
- □ RECEIVE PACKET assigns delimited ASCII strings from a serial port or document.
- □ RECEIVE RECORD reads a 4th Dimension record from a serial port or document.
- □ RECEIVE VARIABLE reads a 4th Dimension variable from a serial port or document.
- □ SAVE VARIABLE saves a variable to a document on disk.
- □ SEND PACKET writes ASCII strings sequentially to a document or a serial port.

- SEND RECORD sends a 4th Dimension record to a serial port or a document.
- SEND VARIABLE sends a 4th Dimension variable to a serial port or a document.
- SET CHANNEL creates, opens, or closes a document. It can also be used to set up a serial port.
- USE ASCII MAP loads an ASCII map into memory. The map transposes characters.

Multi-user commands

This section lists commands for programming multi-user systems.

- CLEAR SEMAPHORE clears a specified flag from the network.
- LOAD RECORD loads the current record into memory to see if it is locked.
- Locked returns TRUE or FALSE depending on the read-only and read-write state of a record.
- READ ONLY assigns a default read-only state to each record in the file.
- READ WRITE assigns a default read-write state to each record in the file.
- Semaphore returns TRUE if the network already has flag with the same name as given in its argument. It returns FALSE and creates a flag if the name doesn't exist.
- UNLOAD RECORD removes a record from memory for the current user.

Standard functions

The section presents standard programming functions in the following categories:

- arithmetic and statistical functions
- numeric functions
- transcendental functions
- string functions
- date functions
- time functions

Arithmetic and statistical functions

These functions are available for records at print time and for subrecords at print time and display time. All act on a given field or subfield in the current selection or subselection of the current record.

□ Average returns the arithmetic mean.

□ Max returns the maximum value.

□ Min returns the minimum value.

□ Squares sum returns a sum of squares figure.

□ Std deviation returns a standard deviation figure.

□ Subtotal returns the total for a particular break level.

□ Sum returns the total.

□ Variance returns a variance figure.

Numeric functions

These functions return numeric values.

□ Abs converts its argument to an absolute (unsigned, positive) value.

□ Dec returns the decimal portion of its argument.

□ Int returns the integer portion of its argument.

□ Num returns the numeric value of its argument, a numeric string.

□ Random returns an integer between 0 and 32,767.

□ Round returns its argument rounded by a specified number of places.

□ Trunc returns its argument truncated by a specified number of places.

Transcendental functions

These are the standard trigonometric and logarithmic functions.

□ Arctan returns the arctangent of its argument.

□ Cos returns the cosine of its argument in radians.

□ Exp returns the exponential of its argument.

□ Log returns the natural (Naperian) logarithm of its argument.

□ Sin returns the sine of its argument in radians.

□ Tan returns the tangent of its argument in radians.

String functions

These functions handle strings.

- □ Ascii returns the ASCII code for the first character in its string argument.
- □ Char returns the character specified by its numeric argument, an ASCII code.
- □ GET HIGHLIGHTED TEXT gets the positions of the first and last character of the user selected text.
- □ HIGHLIGHT TEXT either places the insertion point or highlights text.
- □ Length returns the number of characters in a string.
- □ Lowercase returns all characters in its string argument in lowercase.
- □ Num returns the numeric value of its argument, a numeric string.
- □ Position returns the position of the first occurence of one string within another string.
- □ String converts a numeric value to its numeric string counterpart.
- □ Substring returns a specified portion of its string argument.
- □ Uppercase returns all characters in its string argument in uppercase.

Date functions

These functions work with dates.

- □ Current date returns the date kept by the Macintosh clock in 4th Dimension date format.
- □ Date returns a date in 4th Dimension date format from a date expressed as a string.
- □ Day number returns the weekday number from a date.
- □ Day of returns a day of the month number from a date.
- □ Month of returns a month number from a date.
- □ Year of returns a year number from a date.

Time functions

These functions deal with the time of day.

- □ Current time returns the time kept by the Macintosh clock as the number of seconds since midnight.
- □ Time returns the number of seconds since midnight from time expressed as a string in HH:MM:SS format.
- □ Time string returns a time string in HH:MM:SS format from a number indicating the number of seconds since midnight.

Programming

The section presents commands and functions for applications programming. Its categories include

- □ execution flow
- □ managing variables
- □ system variables
- □ Boolean functions
- □ managing the interpreter

See Chapter 2 of *4th Dimension Programmer's Reference* for a discussion of operators.

Execution flow

These commands control the flow of execution.

- □ Case of...Else...End case is the case statement.
- □ If...Else...End if is the branching statement.
- □ While...End while is the loop statement.

Managing variables

These commands manage variables.

- □ CLEAR VARIABLE clears one or more variables from memory.
- □ LOAD VARIABLE loads a variable into memory from a document on disk.
- □ SAVE VARIABLE saves a variable to a document on disk.
- □ Undefined returns TRUE when a variable has not been created (assigned).

System variables

These are 4th Dimension's own variables. See also Appendix E, "System Variables and the System Set."

- □ Document contains the name of the last document that the application read from or wrote to. Read only.
- □ Error contains a 4th Dimension error code, when an error occurs. Read only.

- ☐ FldDelimit contains the ASCII code for the current field delimiter.
- ☐ KeyCode contains the ASCII code for the most recent key press. (Available only to a procedure installed by ON EVENT CALL.) Read only.
- ☐ Modifiers contains a code for the most recent key press of a modifier key (Shift, Option, Command, Caps Lock). (Available only to a procedure installed by ON EVENT CALL.) Read only.
- ☐ MouseDown contains 1 when the user clicks the mouse. (Available only to a procedure installed by ON EVENT CALL.) Read only.
- ☐ OK reflects the status of many 4th Dimension operations.
- ☐ RecDelimit contains the ASCII code for the current record delimiter.

Boolean functions

Use these functions for writing Boolean expressions.

- ☐ False returns FALSE.
- ☐ Not returns the negation of a Boolean expression.
- ☐ Num when applied to a Boolean expression returns 1 for TRUE and 0 for FALSE.
- ☐ True returns TRUE.

Managing the interpreter

These commands work with the interpreter for debugging and executing strings as commands.

- ☐ EXECUTE executes strings that evaluate as valid 4th Dimension commands, functions, and/or arguments.
- ☐ TRACE turns on the Trace facility for debugging.
- ☐ NO TRACE turns off the Trace facility.

Reading interrupts

These commands let you read and act on interrupts.

- ☐ ABORT stops program execution and returns to the menu.
- ☐ ON ERR CALL installs a procedure for trapping and responding to errors.
- ☐ ON EVENT CALL installs a procedure for responding to key presses and mouse clicks.
- ☐ ON SERIAL PORT CALL installs a procedure for responding to serial port events.

Part II

The Commands in Alphabetical Order

ABORT

Syntax

ABORT

Description

ABORT must be executed from within a procedure installed by ON ERR CALL. If the procedure producing the error was called from a menu, ABORT returns execution to the calling menu. If the error occurs during a layout procedure, ABORT stops execution of the procedure. If you are in the Before or During phase of the execution cycle, you stay in the layout. However, if you are in the After phase, ABORT stops execution of the procedure and you leave the layout. ABORT has no effect on the OK variable; it simply stops execution of the layout procedure. The record, however, will be saved if the user clicks an Accept button. Likewise, if a dialog procedure is active, ABORT would have no effect on the values of variables.

Developers use ABORT to protect users from things like file I/O errors. To decide how and where to place ABORT, you can read the 4th Dimension error number through the system variable ERROR. See Appendix H, "Error Messages."

The ABORT command is equivalent to clicking the Abort button in the Error window or in the Trace window.

Example

```
  `Oops; called by ON ERR CALL Error handling routine
CONFIRM("Error present! Do you want to stop?")
If (OK=1)
  ABORT `End procedure, return to menus
End if
```

Reference

ON ERR CALL.

Abs

Syntax Abs (*numexpr*)

Description Abs returns the absolute (unsigned, positive) value of *numexpr*. The example assigns 10 to vVector.

Example

```
vVector:=Abs(-10)
```

References Dec, Int, Round, Trunc.

ACTIVATE LINK

Syntax ACTIVATE LINK (*fieldname1«;fieldname2»*)

Description ACTIVATE LINK searches the linked file for a record whose field matches the value of *fieldname1,* where *fieldname1* is a linking field. If found, ACTIVATE LINK stores the pointer to the record found (the link) with the linking record. If not found, it stores a null pointer.

The optional *fieldname2* must be a field in the linked file. If you specify the second argument, *fieldname2,* 4th Dimension displays a two-column list of records that match the value in the linking field. The left column displays linked field values and the right column displays *fieldname2* values. If the user selects a record from the list, that record pointer is stored as the link. If there is only one match, the link is to that match, and the list does not appear.

ACTIVATE LINK always forces a link, whether the linking field has been modified or not. Unlike LOAD LINKED RECORD, ACTIVATE LINK doesn't change the current selection or current record of the linked file or load the linked record. Use ACTIVATE LINK when you don't want to modify the current selection of the linked file. A typical instance of this is when you work with recursive links or any situation in which you don't want to disturb the current selection of the linked file. This is not a commonly used command. For other discussions on links, see *4th Dimension User's Guide* and *4th Dimension Programmer's Reference.*

Example

```
ACTIVATE LINK([Geneology]Father)
```

References CREATE LINKED RECORD, LOAD LINKED RECORD, LOAD OLD LINKED RECORD.

ADD RECORD

Syntax

ADD RECORD «(*)|(*filename*«;*»)»

Description

ADD RECORD lets the user create, add, and save a new record to the database. ADD RECORD creates a blank record in memory for *filename,* marks the record as the current record, and displays the current input layout. If you do not specify *filename,* the default filename applies.

4th Dimension starts the execution cycle (described in *4th Dimension Programmer's Reference*) by executing the input layout procedure. Clicking an Accept button saves the new record and leaves OK set to 1. Cancelling sets OK to 0. If you have more than one Accept button, you must test each individually to see which button the user clicked.

❖ *Note:* If the user presses Enter, 4th Dimension sets OK to 1, without setting button variables to 1.

Even when canceled, the record remains in memory. Issuing the SAVE RECORD command before changing the current record pointer will save the record anyway. After ADD RECORD executes, the current selection becomes the record created by ADD RECORD.

4th Dimension displays the layout in the window with scroll bars and a grow box on the window. Specifying the optional asterisk argument causes the layout to appear without scroll bars and a grow box. ADD RECORD can't take input through variables. To add a record through variables instead of fields, choose CREATE RECORD, DIALOG, and SAVE RECORD.

Example

```
bOK:=1
DEFAULT FILE([CUSTOMERS])
INPUT LAYOUT("AddRecs")
While(bOK=1)
   ADD RECORD
End while
```

References

CREATE RECORD, DELETE RECORD, MODIFY RECORD.

ADD SUBRECORD

Syntax

ADD SUBRECORD (*subfilename*;*strexpr*«; *»)

Description

ADD SUBRECORD lets the user create and add a new subrecord to the parent record. ADD SUBRECORD creates a blank subrecord in memory for *subfilename* to the current record of the file. It makes this blank subrecord the current subrecord and displays the subfile input layout named by *strexpr*. 4th Dimension executes the input layout procedure as described in *4th Dimension Programmer's Reference.*

Clicking an Accept button keeps the new subrecord in memory. Clicking a Don't Accept button deletes the subrecord from memory. To actually save the new subrecord, you must use SAVE RECORD, because a subrecord is only saved with the record to which it belongs.

ADD SUBRECORD can't take input through variables. To add a subrecord through a form containing variables instead of subfields, choose DIALOG, CREATE SUBRECORD, and SAVE RECORD.

If you have a subfile within a subfile, you must add the lower level subrecord to a specific higher level subrecord. Therefore, be sure to select the desired higher level subrecord before adding the lower level subrecord. If you don't, your new subrecord entry could be added to any of the higher level subrecords. If neither a current record nor a higher level subrecord exists, ADD SUBRECORD has no effect.

4th Dimension displays the layout in the window.with scroll bars and a grow box on the window. Specifying the optional asterisk argument causes the layout to appear without scroll bars and a grow box.

Example

```
ADD SUBRECORD([Employees]Addresses;"InputSubs")
```

References

ADD RECORD, CREATE SUBRECORD, DELETE SUBRECORD, DIALOG, MODIFY RECORD, MODIFY SUBRECORD, SAVE RECORD.

ADD TO SET

Syntax

ADD TO SET («*filename*;»*strexpr*)

Description

ADD TO SET adds the current record from *filename* to the set named by *strexpr*. Use ADD TO SET to put individual records into an existing set. Combined with USE SET and DELETE SELECTION, ADD TO SET comes in handy for placing duplicates into a set and deleting them.

The set must already exist. Otherwise, you'll get an error message. If you don't specify *filename*, ADD TO SET uses the default file. If a current record does not exist, ADD TO SET has no effect.

The example searches for duplicates in a mailing list, adds each duplicate to a set, and saves them in a set.

Example

```
  `Save the Duplicates
gOldName1:="zzzzzz"
gOldName2:="zzzzzz"
DEFAULT FILE([Addresses])
CREATE EMPTY SET("Duplicate Names")
ALL RECORDS
SORT SELECTION([Addresses]ZIP;>;[Addresses]Name2;>;[Addresses]Name1;>)
While (Not(End selection))
  If(([Addresses]Name2=gOldName2)&([Addresses]Name1=gOldName1))
    ADD TO SET("Duplicate Names")
  End if
  gOldName1:=[Addresses]Name1  `Prepare for next test
  gOldName2:=[Addresses]Name2
  NEXT RECORD
End while
SAVE SET("Duplicate Names";"New Dupes Set")
CLEAR SET("Duplicate Names")
```

References

CLEAR SET, CREATE EMPTY SET, CREATE SET, Records in set, USE SET.

After

Syntax

After

Description

4th Dimension generates an After phase for each subrecord and then for the parent record when the user validates the record to which the subfiles belong. An After phase occurs when 4th Dimension sets After to TRUE and executes an input layout procedure. Clicking an Accept button or pressing the Enter key validates a record. Clicking Cancel or pressing Command-Period cancels a record. 4th Dimension does not generate an After phase for dialog and output layout procedures.

❖ *No After phase:* If the user doesn't modify data and does validate the record, there is no After phase.

For details on the part of After in the execution cycle, see the *4th Dimension Programmer's Reference*. Place After only in an input layout or file procedure.

❖ *Testing execution phases:* Typically, a developer will write If ...End if statements or a Case statement for each phase.

The example parses the month out of the SalesDate field and assigns it to the Month field. The Month field needn't appear in the layout. In this case, the programmer has created a Month field for sorting and doing breaks on monthly totals. The assigning of the month takes place in After, for two reasons. First, the user has finished all data entry. Second, the user doesn't need to see the month number or its assignment.

Example

```
If(After)
  Month:=Month of(SalesDate)
End if
```

References

ADD RECORD, Before, During, INPUT LAYOUT, MODIFY RECORD.

ALERT

Syntax ALERT (*strexpr*)

Description ALERT puts up an alert box that displays *strexpr*. The box includes an OK button. The message area can accommodate as many as 255 characters, depending on the widths of characters. ALERT is good for passing information to users and for providing debugging information like variable values. This is the "Note" type Alert box.

Example

```
ALERT("You have entered "+String(vCount)+" records.")
```

References CONFIRM, DIALOG, MESSAGE, Request.

ALL RECORDS

Syntax ALL RECORDS «(*filename*)»

Description ALL RECORDS makes the current selection of *filename* all records in the file and loads the first record as the current record. This action cancels any previous sort order. If you don't specify *filename,* ALL RECORDS uses the default file.

When a database is first opened, the current selection for each file is empty. Use ALL RECORDS to bring all records into the current selection.

Example

```
  `Forward: Display each record
DEFAULT FILE([Addresses])
ALL RECORDS
While (Not(End selection))
    DISPLAY RECORD
    NEXT RECORD
End while
```

References ALL SUBRECORDS, FIRST RECORD, LAST RECORD.

ALL SUBRECORDS

Syntax

ALL SUBRECORDS (*subfilename*)

Description

ALL SUBRECORDS selects all subrecords for the current record. It does not select subrecords for all records in the current selection. The same principle applies when working with levels of subfiles. If neither a current record nor a higher level subrecord exists, ALL SUBRECORDS has no effect.

In the example, Stats is the *filename* of the parent file and Sales is the *subfilename*.

Example

```
ALL SUBRECORDS([Stats]Sales)
```

References

FIRST SUBRECORD, LAST SUBRECORD, NEXT SUBRECORD, PREVIOUS SUBRECORD, Records in subselection.

APPLY TO SELECTION

Syntax

APPLY TO SELECTION (*«filename;»statement*)

Description

APPLY TO SELECTION applies a one-line statement or a developer-written procedure to the current selection in a record-by-record fashion. When doing a data modification task, APPLY TO SELECTION automatically loads each record, does the required work, and saves each record if the record data was modified. Because of this record-by-record approach, APPLY TO SELECTION can take a long time when working with large selections. Besides doing summaries of data in the current selection, you can change field values. For example, making price changes, putting all names in uppercase, or adding a suffix to selected part numbers. MESSAGES OFF turns off the standard 4th Dimension progress thermometer for this command.

After APPLY TO SELECTION has been called, you can check whether the user has cancelled the operation by testing the OK variable. If the user has cancelled, OK returns 0. Otherwise, it returns 1. If you don't specify *filename,* APPLY TO SELECTION uses the default file. If the current selection is empty, APPLY TO SELECTION has no effect.

The example computes total sales for a company's Midwest region.

Example

```
Tot:=0
DEFAULT FILE([Sales])
SEARCH([Sales]Region="Midwest")
APPLY TO SELECTION(Tot:=Tot+[Sales]Amount)
ALERT("Midwest sales="+String(Tot;"$#,###,###.00"))
```

References

ALL RECORDS, SEARCH, SEARCH SELECTION.

APPLY TO SUBSELECTION

Syntax APPLY TO SUBSELECTION (*subfilename*;*statement*)

Description APPLY TO SUBSELECTION applies *statement,* a one-line statement, or a developer-written procedure to the current subrecord selection in *subfilename* of the current record. Besides doing summaries of data in the current selection, you can change subfield values. For example, making price changes, putting all names in uppercase, or adding a suffix to selected part numbers.

If you modify values in a subrecord, the subrecords will be saved only if you save the record that contains them. Because subrecords reside in memory, APPLY TO SUBSELECTION is a far quicker process than APPLY TO SELECTION. If the subselection is empty, APPLY TO SUBSELECTION has no effect.

Example

```
ALL SUBRECORDS([Invoice]Detail)
APPLY TO SUBSELECTION([Invoice]Detail; [Invoice]Detail'ItemTotal:=
    [Invoice]Detail'Price*[Invoice]Detail'Units Sold)
vInvTot:=Sum([Invoice]Detail'ItemTotal)
```

References ALL SUBRECORDS, APPLY TO SELECTION.

Arctan

Syntax Arctan (*numexpr*)

Description Arctan returns the arctangent in radians of *numexpr,* where *numexpr* is a tangent. (One degree equals 0.0174532925199432958 radians.) The example assigns 0.785397663397198З1 to vAtan.

Example

```
vAtan:=Arctan(.999999)
```

References Cos, Exp, Log, Sin, Tan.

Ascii

Syntax Ascii (*strexpr*)

Description Ascii returns the ASCII code of the first character in *strexpr*. The Char function is the counterpart of Ascii, returning a character for an ASCII code. To test for case in 4th Dimension, use Ascii. For example:

(**Ascii**("A")=**Ascii**("a"))

returns FALSE, whereas

("A"="a") returns TRUE.

The example assigns 65 to the variable GetAsc.

Example GetAsc:=**Ascii**("ABC")

Reference Char.

Average

The **Average** function computes the arithmetic mean of the specified field or subfield. **Average** has two syntaxes. The first applies to finding the average value of a particular field in the current selection. It works only at print time. The second applies to finding the average value of a particular subfield in a specified subfile. In both cases, the field type must be numeric.

Syntax 1

Average (*fieldname*)

Description 1

Average returns the average for *fieldname* in the current selection. This works only in In footer or In break at break level 0 (grand totals) in an output layout procedure when printing with PRINT SELECTION. To compute an average, 4th Dimension must work with each record in the selection. At print time, 4th Dimension brings each record into memory and builds the average figure.

Important

Average does not clear after a break. Rather, it calculates a record-by-record average for the entire selection. Average is meaningful only when printed in break level 0.

Example 1

```
If(In footer & End selection([Income]))
   vAverage:=Average([Income]Sale)
End if
```

References 1

In break, In footer, Level, Max, Min, PRINT SELECTION, Squares sum, Std deviation, Sum, Variance.

Syntax 2

Average (*subfieldname*)

Description 2

Average returns the average value of *subfieldname* in *subfilename* for the current subselection of the current record.

Example 2

```
vAvg:=Average([Invoice]Ordered'ItemTotal)
```

References 2

Max, Min, Squares sum, Std deviation, Sum, Variance.

BEEP

Syntax BEEP «(*posintexpr*)»

Description BEEP without an argument causes the computer to emit a tone for 0.5 second. To change the duration of the tone, include the optional numeric argument. The duration of the beep is *posintexpr* times 1/60. The example would cause a 1-second beep.

Example **BEEP**(60)

Reference None.

Before

Syntax

Before

Description

4th Dimension generates a Before phase when you initiate any activity that uses a layout (whether input, output, or dialog). A Before phase occurs when 4th Dimension sets Before to TRUE and executes a layout procedure. Once 4th Dimension executes these statements, it sets Before to FALSE. For details on Before in the execution cycle, see *4th Dimension Programmer's Reference.*

❖ *Testing execution phases:* Typically, a developer will write If...End if statements for each phase or a Case statement in which two or three of the phases are the cases.

For an input layout, if a record contains subrecords, 4th Dimension generates a Before phase for each subrecord, and then the Before phase for the parent record. A Before phase is generated for a subrecord multi-line layout when the user presses Command-Tab to add a new subrecord.

A Before phase is generated for a subfile Full Page layout when the user double-clicks in a subfile area. Typically, developers use the Before phase to do initialization for a record. In the example (an input layout routine), 4th Dimension assigns the current date to the Entry date field and then sets Before to FALSE.

When printing, 4th Dimension sets Before to TRUE for each record, before printing the record. If you need to select subrecords, make the selection in the Before phase of the parent record's output layout. 4th Dimension executes the Before phase for each subrecord. This is the opposite order of that performed for data entry.

When you display a selection with DISPLAY SELECTION or MODIFY SELECTION, 4th Dimension executes Before and During simultaneously.

Place Before only in a layout or file procedure.

Example

```
If(Before)
   If(Entry date = !00/00/0000!)
      Entry date:=Current date
   End if
End if
```

References

After, During, In break, In footer, In header.

Before selection

Syntax

Before selection «(*filename*)»

Description

Before selection returns TRUE after PREVIOUS RECORD moves the record pointer before the first record in the current selection of *filename*. If you don't specify *filename,* Before selection uses the default file. If the current selection is empty, Before selection returns TRUE.

If you want to print a unique message (including special strings, the time the report was run, and so on) in the first page's header, you can test Before selection in a header. Before selection returns TRUE when you print the first header, meaning that no record in the selection has been printed.

Example

```
 `Backward: Move backward through file
DEFAULT FILE([Addresses])
ALL RECORDS
LAST RECORD
While (Not(Before selection))
  DISPLAY RECORD
  PREVIOUS RECORD
End while
```

References

End selection, In header, PREVIOUS RECORD.

Before subselection

Syntax

Before subselection (*subfilename*)

Description

Before subselection returns TRUE when a PREVIOUS SUBRECORD command moves the current subrecord pointer before the first subrecord in the current subselection of *subfilename* for the current record. If the current subselection is empty, Before subselection returns TRUE. The example displays all subrecords in reverse order for each record.

Example

```
  `Demo before subselection, last subrecord, and previous subrecord
DEFAULT FILE([Stats]Name)
ALL RECORDS
While (Not(End selection))
  vReport:=""
  ALL SUBRECORDS([Stats]Sales)
  LAST SUBRECORD([Stats]Sales)
  While (Not(Before subselection([Stats]Sales)))
    vReport:=vReport+String([Stats]Sales'Bucks)+Char(13)
    PREVIOUS SUBRECORD([Stats]Sales)
  End while
  bContinue:=0
  While (bContinue=0)
    DIALOG("dStatList")
  End while
  NEXT RECORD
End while
```

References

End subselection, PREVIOUS SUBRECORD.

BUTTON TEXT

Syntax BUTTON TEXT (*buttonvar,strexpr*)

Description BUTTON TEXT displays *strexpr* as the text for the button specified by *buttonvar*. *buttonvar* is a button variable. "Button" includes check boxes and radio buttons. With commands like BUTTON TEXT, you can create generic dialogs and use the same dialog in different contexts, like searches on different files. The action of changing the button text is local to the use of the layout. Thus, if you change a button's name, it does not change it in the layout, but only during the current use of the layout. Be sure to make the button area large enough to accomodate any text you might assign to the button. Use BUTTON TEXT only in an input layout procedure.

❖ *Button and variable names:* Do not confuse the button variable name and the button title. They are two different things.

Example **BUTTON TEXT**(NextOne;"One More")

References DISABLE BUTTON, ENABLE BUTTON.

Case of...Else...End case

Syntax

Case of
{:(*boolexpr*)
{*statement(s)*}}«{*}»
«Else
{*statement(s)*}»
End case

Description

The Case of statement evaluates a series of Boolean expressions or cases one at a time. Case of executes statements belonging to the first and only the first true Boolean expressions it encounters, even if subsequent Boolean expressions are true.

When this execution ends, program execution continues with the statement following End case. You can include an Else clause before the concluding End case. Else acts as a default activity that only executes when all other Boolean expressions fail.

You can nest Case of statements within Case of statements, as long as the close of an inner Case of statement does not appear after the close of an outer Case of statement.

The example shows how you can use Case of to evaluate the execution cycle in an input layout procedure.

Example

```
Case of
   :(Before)
      If(Entry date = !00/00/0000!)
         Entry date:=Current date
      End if
   :(During)
      LOAD LINKED RECORD(CustNo;[Client]ID No)
      Name:=[Client]Client
      City:=[Client]City
   :(After)
      Month:=Month of(Entry Date)
End case
```

References

If...Else...End if, While...End while.

Char

Syntax

Char (*posintexpr*)

Description

Char returns an alphanumeric value for *posintexpr*, where *posintexpr* is a valid ASCII code (0 to 255). You can also find the value of the system variable KeyCode in an ON EVENT CALL routine with Char and assign symbols to menu items with the CHECK ITEM procedure.

The example assigns the carriage return character (ASCII 13) to the variable gCR and then uses the variable to place carriage returns at the end of each line in a mailing label. The example is a layout procedure for a mailing label layout. It concatenates all address information into one variable (vLabel), which is the only entity in the label layout.

Example

```
  `vLabel procedure for creating labels
If (Before)
  gCR:=Char(13)
  vLabel:=Name1+" "+Name2+gCR+Addr1+gCR
  If (Addr2#"")
    vLabel:=vLabel+Addr2+gCR
  End if
  vLabel:=vLabel+City+", "+St+" "+String(ZIP)
End if
```

Reference

Ascii.

CHECK ITEM

Syntax CHECK ITEM (*posintexpr1;posintexpr2;strexpr*)

Description CHECK ITEM places a character, *strexpr*—either a check mark (ASCII 18) or a space—by a menu command. *posintexpr1* is the menu title number and *posintexpr2* is the number of the item under the title. If you supply a null string or a space character for *strexpr,* it erases the check on the specified menu item.

CHECK ITEM is local to the current menu bar. For example, if you leave the current menu bar and go to another menu bar, and then return to the original bar, the check on the original bar will be gone.

❖ *Restoring the unchecked state:* Use MENU BAR to restore all items to their default (unchecked) state.

The Edit and Apple menus are built in and are not a part of the menu count. The File menu is menu title number one.

Examples

```
CHECK ITEM(2;1;Char(18))
CHECK ITEM(2;1;"")
```

References Char, DISABLE ITEM, ENABLE ITEM, MENU BAR, Menu selected.

CLEAR SEMAPHORE

Syntax CLEAR SEMAPHORE (*strexpr*)

Description CLEAR SEMAPHORE erases the semaphore flag *strexpr* for multi-user applications. See *4th Dimension Developer's Notes* for details on this and other multi-user commands.

CLEAR SET

Syntax

CLEAR SET (*strexpr*)

Description

CLEAR SET removes the set named by *strexpr* from memory and frees the memory taken by the set. CLEAR SET does not affect your files, selections, or records. If you want to re-use a set, save it to disk with the SAVE SET command before executing CLEAR SET. A good practice is to always clear sets when you are finished with them.

The at sign (@) may be used for clearing more than one set at a time. For example, **CLEAR SET** ("B@") will clear all sets whose names start with the letter B. Using **CLEAR SET** ("@") will clear all sets.

Example

```
  `Creates & saves a set of all Palo Alto customers
DEFAULT FILE([Addresses])
ALL RECORDS
SEARCH([Addresses]City="Palo Alto")
CREATE SET("Palo Alto")  `Makes current selection a set
MESSAGE("Now saving Palo Alto set.")
SAVE SET("Palo Alto";"Palo Alto Set")
CLEAR SET("Palo Alto")
```

References

ADD TO SET, CREATE EMPTY SET, CREATE SET, LOAD SET, Records in set, SAVE SET, USE SET.

CLEAR VARIABLE

Syntax

CLEAR VARIABLE (*strexpr*)

Description

CLEAR VARIABLE erases all memory variables that begin with *strexpr,* frees the memory space they occupy, and leaves them undefined. For example, an argument of "Co" would erase variables like Company and CodeNum.

Depending on the need at hand, CLEAR VARIABLE works well at the beginning and/or end of a procedure to free memory and to prevent problems that could arise from different routines using the same variable names with unexpected values. Use CLEAR VARIABLE for initialization and global cleanup.

Important

Executing CLEAR VARIABLE ("") clears all variables, including system variables. This leaves all variables undefined.

❖ *Local variables:* You can economize on memory by using local variables where appropriate. To signify a local variable, precede the variable name with a dollar sign ($). For example: $LoopCount. 4th Dimension automatically clears local variables when completing the routine in which the variables are defined and used. Local variables are accessible only to the procedure in which they are defined.

Example

```
 `Creates a document for variables if none exists
CLEAR VARIABLE("MyVar")
LOAD VARIABLE("MyDoc";MyVar)
If (Undefined(MyVar))
  CONFIRM("'MyDoc' doesn't exist. Create it?")
  If (OK=1)
    MyVar:=Request("Enter value of 'MyVar'")
    SAVE VARIABLE("MyDoc";MyVar)
  End if
Else
  ALERT("Value of 'MyVar' is "+MyVar)
End if
```

References

LOAD VARIABLE, RECEIVE VARIABLE, SAVE VARIABLE, SEND VARIABLE, Undefined.

CLOSE WINDOW

Syntax CLOSE WINDOW

Description CLOSE WINDOW closes the window created by an OPEN WINDOW command. If you want to switch from one window to another, you must close the current window before opening a new one. Otherwise, 4th Dimension will continue writing to the first window and not open the second window. (4th Dimension allows only one custom window open at a time.) CLOSE WINDOW has no effect if a custom window isn't open; it will not close the 4th Dimension window. The example writes ten items to a window through MESSAGE. When close window takes effect, you see further items written to a message box.

Example

```
 `Demo CLOSE WINDOW
gLine:=0
DEFAULT FILE([Catalog])
ALL RECORDS
OPEN WINDOW(5;40;250;300;0;"Window")
While (Not(End selection))
  gLine:=gLine+1
  If (gLine=10)
    CLOSE WINDOW
  End if
  MESSAGE([Catalog]Description+Char(13))
  NEXT RECORD
End while
```

References ERASE WINDOW, OPEN WINDOW.

CONFIRM

Syntax

CONFIRM (*strexpr*)

Description

CONFIRM creates a dialog box that displays *strexpr* as a prompt message. A CONFIRM box has two buttons: OK and Cancel. (ALERT creates a dialog box with just the OK button.) Also see Request. Clicking OK sets the system variable OK to 1; clicking Cancel sets OK to 0. The OK button is the default button. Confirm is a Caution type Alert box.

❖ *By the way:* If you need to have a user confirm or refuse more than one item of data, design a dialog box with variables and buttons, rather than displaying multiple Confirm boxes.

Example

```
 `Creates a document for variables if none exists
CLEAR VARIABLE("MyVar")
LOAD VARIABLE("MyDoc";MyVar)
If (Undefined(MyVar))
  CONFIRM(" 'MyDoc' doesn't exist. Create it?")
  If (OK=1)
    MyVar:=Request("Enter value of 'MyVar'")
    SAVE VARIABLE("MyDoc";MyVar)
  End if
End if
```

References

ALERT, DIALOG, MESSAGE, Request.

Cos

Syntax

Cos (*numexpr*)

Description

Cos returns the cosine of *numexpr*, where *numexpr* is expressed in radians. (One degree equals 0.01745329 radians.) The example returns the cosine of 45 degrees, 0.707106781186547524.

Example

```
vCos:=Cos(45*0.0174532925199432958)
```

References

Arctan, Exp, Log, Sin, Tan.

CREATE EMPTY SET

Syntax

CREATE EMPTY SET («*filename*;»*strexpr*)

Description

CREATE EMPTY SET creates an empty set for *filename* and gives the empty set the name *strexpr*. You can add to this empty set with ADD TO SET and create a current selection with USE SET. If you don't specify *filename,* CREATE EMPTY SET applies to the default file.

Example

```
  `Save duplicates
gOldName1:="zzzzzz"
gOldName2:="zzzzzz"
DEFAULT FILE([Addresses])
CREATE EMPTY SET ("Duplicate Names")
ALL RECORDS
SORT SELECTION([Addresses]ZIP;>;[Addresses]Name2;>;[Addresses]Name1;>)
While (Not(End selection))
  If (([Addresses]Name2=gOldName2)&([Addresses]Name1=gOldName1))
    ADD TO SET("Duplicate Names")
  End if
  gOldName1:=[Addresses]Name1   `Prepare for next test
  gOldName2:=[Addresses]Name2
  NEXT RECORD
End while
SAVE SET("Duplicate Names";"New Dupes Set")
CLEAR SET ("Duplicate Names")
```

References

ADD TO SET, CLEAR SET, CREATE SET, SAVE SET, USE SET.

CREATE LINKED RECORD

Syntax

CREATE LINKED RECORD (*fieldname*)

Description

CREATE LINKED RECORD has two effects. If no linked record exists (if a match is not found for the current value of *fieldname*), CREATE LINKED RECORD creates a blank linked record. If the linked record exists, CREATE LINKED RECORD acts like LOAD LINKED RECORD. That is, it loads the linked record into memory and stores the pointer to that record in the linking record. You must execute SAVE LINKED RECORD to save the record.

CREATE LINKED RECORD is activated the first time it's invoked for a record, and thereafter only when *fieldname* is modified. Therefore, if you place CREATE LINKED RECORD in the Before or After phase of the execution cycle, you must trick CREATE LINKED RECORD by performing a modification that really changes nothing (like PartNo:=PartNo).

Use CREATE LINKED RECORD instead of LOAD LINKED RECORD when you want the procedure, rather than the user, to supply data to the new linked record.

For other discussions on links, see *4th Dimension User's Guide* and *4th Dimension Programmer's Reference.*

Example

```
.
.
.
:(After)
     `Reinstate old Quantity to protect against false modification
   LOAD OLD LINKED RECORD(PartNo)
   [Catalog]OnHand:=[Catalog]OnHand+Old(Quantity)
   SAVE OLD LINKED RECORD(PartNo)
   PartNo:=PartNo
     `Subtract Quantity from Catalog Onhand & save new Onhand
   CREATE LINKED RECORD(PartNo)   `Create a link based on PartNo
   [Catalog]ItemNo:=PartNo   `Identify object of link
   [Catalog]OnHand:=[Catalog]OnHand-Quantity   `Subtract order qty from catalog
   SAVE LINKED RECORD(PartNo)
```

References

ACTIVATE LINK, LOAD LINKED RECORD, LOAD OLD LINKED RECORD, Old, SAVE LINKED RECORD, SAVE OLD LINKED RECORD.

CREATE RECORD

Syntax

CREATE RECORD «(*filename*)»

Description

CREATE RECORD creates a blank record for *filename,* in memory, and makes it the current record and the current selection (a one-record current selection). If you don't follow CREATE RECORD with SAVE RECORD, before changing the current record pointer, 4th Dimension ignores the created record and doesn't save it. If you don't specify *filename,* CREATE RECORD applies to the default file. If you want the user to add data through an input layout, use ADD RECORD.

Typically, you can use CREATE RECORD in four ways:

□ when getting input through a dialog layout, using variables

□ when moving data to another file

□ when reading data into a file from a desktop document

□ when importing data through a network or modem

Example

```
 `AddDate
DEFAULT FILE([I.Card])
bOK:=1
While (bOK=1)
   vDate:=""
   DIALOG("DateTest")
   CREATE RECORD
   [I.Card]Name:=vDate
   [I.Card]TestDate:=Date(vDate)
   If (bOK=1)`Test to make sure last record is not a blank
      SAVE RECORD
   End if
End while
```

References

ADD RECORD, DIALOG, MODIFY RECORD, SAVE RECORD.

CREATE SET

Syntax

CREATE SET («*filename*;»*strexpr*)

Description

CREATE SET creates a set named *strexpr* and places the current selection from *filename* in the set. *strexpr* may be from 1 to 80 characters long. Once you have created a set, you can save it, load it, and do set operations with other sets from the same file. If you don't specify *filename,* CREATE SET applies to the default file.

❖ *Sets and the current record:* When you create a set, the position of the currrent record is kept in the set. USE SET retrieves the position of this record and makes it the new current record. If you delete this record before you execute USE SET, 4th Dimension selects the first record in the set as the current record. Also, if you form a set that does not contain the position of the current record, USE SET selects the first record in the set as the current record.

Example

```
  `Creates & saves a set of all Palo Alto customers
DEFAULT FILE([Addresses])
SEARCH([Addresses]City="Palo Alto")
CREATE SET("Palo Alto")   `Makes current selection a set
MESSAGE("Now saving Palo Alto set.")
SAVE SET("Palo Alto";"Palo Alto Set")
CLEAR SET("Palo Alto")
```

References

ADD TO SET, CLEAR SET, CREATE EMPTY SET, LOAD SET, Records in set, SAVE SET, SEARCH, SEARCH BY INDEX, SEARCH SELECTION, USE SET.

CREATE SUBRECORD

Syntax

CREATE SUBRECORD (*subfilename*)

Description

CREATE SUBRECORD adds a new subrecord to *subfilename* for the current record and makes it the current subrecord. 4th Dimension saves the created subrecord only if you save the record that contains it with the SAVE RECORD command in a global procedure or if the user clicks an Accept button in an input layout. If you use CREATE SUBRECORD, and there is no current record, CREATE SUBRECORD has no effect. If you want the user to add data through an input layout, use ADD SUBRECORD.

You can create numerous subrecords without having to save each one individually. All changes to a subfile (additions, modifications, and deletions) are saved only when you save the parent record.

The example shows a portion of a procedure that retrieves records and subfiles from a text file.

Example

```
.
.
.
While(i<gStop)
  CREATE SUBRECORD([I.Card]Phone)
   [I.Card]Phone'PhNum:=§("gRec"+String(i))
   i:=i+1
   [I.Card]Phone'Comment:=§("gRec"+String(i))
   i:=i+1
End while
SAVE RECORD
```

References

ADD SUBRECORD, MODIFY SUBRECORD, SAVE RECORD.

Current date

Syntax Current date

Description Current date returns the current date from the Macintosh clock as a 4th Dimension date.

Example

```
If(Before)
  Entry date:=Current date
End if
```

References Date, Day number, Day of, Month of, Year of.

Current password

Syntax Current password

Description Current password returns the string the user typed to get into the database. The function returns a null string if no password exists.

Example

```
WhoIsIt:=Current password
```

Reference None.

Current time

Syntax

Current time

Description

Current time returns the current time from the Macintosh clock in seconds since midnight. Current time is always between 0 and 86,399 inclusive. Time string converts seconds into an hours, minutes, and seconds string. The example is a routine that pauses execution for approximately three seconds. This works well when trying to hold DISPLAY RECORD or MESSAGE output on the screen.

Example

```
  `Pause display for 3 seconds. Works near midnight.
vNow:=Current time
While(Abs(Current time-vNow)<3)
End while
```

References

Time, Time string.

Date

Syntax

Date (*strexpr*)

Description

Date returns a 4th Dimension date (for example: !04/15/90!) from a *strexpr* expressing the date. You can express date elements as one or two digits. The following characters are valid date separators: slash (/), space ( ), period (.), and hyphen (-).

strexpr must take the order mm-dd-yy. If you do not enter the date as a string (for example, you omit the enclosing double quotation marks), 4th Dimension will not generate an error, but your test or other activity may fail. Date only reads digits and delimiters; you cannot enter alphabetic characters like "Jan-1-1990."

If your year number contains only two digits, 4th Dimension automatically adds 1900 to it. If a year number contains more than two digits, 4th Dimension won't change it. If you need to enter a year between 1 and 100 A.D., you must fill with at least one leading zero. For example, 0033 or 033. If you must work with B.C. dates, use the Long integer type. Date numbers must be positive.

The example takes a date entered as a string from a Request, converts the string to a 4th Dimension date, and searches for records containing that date. Invoice Date is typed as a Date field.

Example

```
 `Search for date
ReqDate:=Request("Enter date"; String(Current date))
If(OK=1)
  DEFAULT FILE([Invoice History])
  SEARCH([Invoice History]Invoice Date=Date(ReqDate))
  DISPLAY SELECTION
End if
```

References

Current date, Day number, Day of, Month of, Year of.

Day number

Syntax Day number (*date*)

Description Day number returns a number representing the week day on which *date* falls (Sunday equals 1). The example returns the current day number.

Example

```
ALERT("It's Day number "+String(Day number(Current date)))
```

References Current date, Date, Day of, Month of, Year of.

Day of

Syntax Day of (*date*)

Description Day of returns the day of the month from *date,* where *date* is a valid 4th Dimension date. The example would assign 14 to the variable vDay.

Example

```
vDay:=Day of(!8/14/90!)
```

References Current date, Date, Day number, Month of, Year of.

Dec

Syntax

Dec (*numexpr*)

Description

Dec returns the decimal part of *numexpr*. The value returned is always positive or zero. The example converts decimal weights to pounds and ounces. The example would convert 8.5 pounds to 8 pounds 8 ounces.

Example

```
vLb:=Int(Weight)
vOz:=16*Dec(Weight)
```

References

Abs, Int, Round, Trunc.

DEFAULT FILE

Syntax

DEFAULT FILE (*filename*)

Description

DEFAULT FILE makes *filename* the default file. Once you execute this statement, you can omit optional filename arguments in commands where filename is an optional argument. In the example, the programmer was able to omit the filename from the INPUT LAYOUT and ADD RECORD commands, because DEFAULT FILE has already stated it.

Once you have declared a file to be the default file, it remains so, until you declare a different file as the default. This means you can specify other files in a procedure without giving up the default status of the named file. When working with statements in a global procedure that require field names, you must still include the filename prefix to the field name. For example,

```
vDate:=[Sales]SaleDate
```

If you were to omit the [Sales] filename prefix from this assignment, 4th Dimension would interpret SaleDate as a variable. The only time you can omit the filename prefix from a field name is in a layout procedure, when referring to a field name in the layout's file.

Example

```
OK:=1
DEFAULT FILE([CUSTOMERS])
INPUT LAYOUT("Add recs")
While(OK=1)
   ADD RECORD
End while
```

Reference

None.

DELETE DOCUMENT

Syntax

DELETE DOCUMENT (*docname*)

Description

DELETE DOCUMENT deletes the document (Macintosh file) named by *docname*. Deleting a document sets OK to 1. If DELETE DOCUMENT can't delete a document, OK returns a 0. DELETE DOCUMENT will not work on opened documents and non-existent documents.

For security purposes, you cannot give a null string argument. If you do this, 4th Dimension will not show the standard dialog for choosing a document and will display an error message.

❖ *Pathnames:* You can access documents in other folders by writing the pathname of the folder as a part of *docname*. See Appendix F, "4th Dimension and Macintosh Codes," for a brief discussion of pathnames.

Warning
DELETE DOCUMENT will delete 4th Dimension documents, documents created by other applications, and the applications as well. Deleting is permanent and cannot be undone.

Example

```
DELETE DOCUMENT("Customers.Text")
```

References

EXPORT DIF, EXPORT SYLK, EXPORT TEXT, SAVE SET, SAVE VARIABLE, SEND PACKET, SEND RECORD, SEND VARIABLE, SET CHANNEL.

DELETE RECORD

Syntax

DELETE RECORD «(*filename*)»

Description

DELETE RECORD deletes the current record of *filename*. After deletion, the current selection is empty. As a result, you can't use DELETE RECORD to scan through a selection. To delete a group of records, use DELETE SELECTION. If you don't specify *filename,* DELETE RECORD uses the default file. If there is no current record, DELETE RECORD has no effect.

Warning

A deletion is permanent and cannot be undone.

Example

```
DELETE RECORD([Customers])
```

Reference

DELETE SELECTION.

DELETE SELECTION

Syntax

DELETE SELECTION «(*filename*)»

Description

DELETE SELECTION deletes the current selection of records from *filename*. After deletion, the current selection is empty. If you don't specify *filename,* the command uses the default file. If the current selection is empty, DELETE SELECTION has no effect.

Warning

A deletion is permanent and cannot be undone.

Example

```
  `Delete 1990 Invoices
DEFAULT FILE([Invoice History])
ALL RECORDS
SEARCH(Year of([Invoice History]Invoice Date)=1990)
CONFIRM(String(Records in selection)+" are to be deleted. Continue?")
If (OK=1)
    DELETE SELECTION
   ALL RECORDS
End if
DISPLAY SELECTION
```

Reference

DELETE RECORD.

DELETE SUBRECORD

Syntax

DELETE SUBRECORD (*subfilename*)

Description

DELETE SUBRECORD marks the current subrecord for deletion. *subfilename* names the subfile of the current record to which the current subrecord belongs. After DELETE SUBRECORD marks the subrecord for deletion, the current subrecord selection is empty. As a result, you can't use DELETE SUBRECORD to scan through a subselection. 4th Dimension deletes marked subrecords only if you save the current record with SAVE RECORD or if the user clicks an accept button. This command has no effect if no current subrecord exists.

❖ *Deleting parent records:* Deleting a parent record automatically deletes all its subrecords.

Example

DELETE SUBRECORD([Addresses]PhoneNos)

References

DELETE RECORD, SAVE RECORD.

DIALOG

Syntax

DIALOG (*«filename;»strexpr*)

Description

DIALOG presents the dialog layout specified by *strexpr*. You can use DIALOG

- □ to get input through layout variables
- □ to present a group of related choices (like telecommunications settings)
- □ to get input for custom searches and sorts
- □ to create special applications like on-screen calculators

❖ *Windows:* You can use OPEN WINDOW to create windows that look like standard dialogs.

Dialog layouts can only take input through variables, not input fields. They can, however, display fields. If you omit buttons from dialog layout, 4th Dimension will automatically create OK and Cancel buttons. If you don't specify *filename,* DIALOG will use the default file.

4th Dimension executes the Before and During phases of dialog layout procedures (see Chapter 3 of *4th Dimension Programmer's Reference*). You can test the system OK variable to see if the user clicked an Accept button. Clicking an Accept button causes OK to return 1. Otherwise, it returns 0.

Example

```
   DateLook: Select invoices with dates since input date
DEFAULT FILE([Invoice])
OUTPUT LAYOUT("Output")
ALL RECORDS
vdate:=Current date   `vDate is date variable
DIALOG("GetDate")
If(bOK=1)
  SEARCH([Invoice]InvoiceDate>vDate)
  DISPLAY SELECTION
End if
```

References

ALERT, Confirm, INPUT LAYOUT, MESSAGE, Request.

DIFFERENCE

Syntax DIFFERENCE (*strexpr1*;*strexpr2*;*strexpr3*)

Description DIFFERENCE creates a set (named by *strexpr3*) by comparing set1 and set2, named respectively by *strexpr1* and *strexpr2*. DIFFERENCE puts the elements that are unique to set1 into set3. Set3 needn't exist prior to issuing this command. In fact, if it does exist, 4th Dimension will delete the old set and recreate it after recomputing the new set. Further, set3 can be either set1 or set2.

Example **DIFFERENCE**("Above9";"Below20";"Tens")

References CLEAR SET, INTERSECTION, LOAD SET, SAVE SET, UNION, USE SET.

DISABLE BUTTON

Syntax DISABLE BUTTON (*buttonvar*)

Description DISABLE BUTTON grays out (disables) the button specified by *buttonvar*. *buttonvar* is the button variable. Like BUTTON TEXT, DISABLE BUTTON is local to the layout and only works when the layout is on the screen.

Place DISABLE BUTTON only in a layout procedure. DISABLE BUTTON works with all 4th Dimension buttons and check boxes. Use DISABLE BUTTON when you want to prevent access to a button. For example, you would disable a button named Previous Record when the record pointer points to the first record of the current selection. Buttons include Accept buttons, Don't Accept buttons, Button buttons, Radio buttons, and Check boxes.

Example

```
If (Before)
  vWholename:=[RoloText]First+" "+[RoloText]Last
  vPNum:=[RoloText]Extension
  vRem:=[RoloText]Comment
  If (gRecNo=1)
    DISABLE BUTTON(bPrevious)
  Else
    ENABLE BUTTON(bPrevious)
  End if
  If (gRecNo=gTotRecs)
    DISABLE BUTTON(bNext)
  Else
    ENABLE BUTTON(bNext)
  End if
End if
```

References BUTTON TEXT, ENABLE BUTTON.

DISABLE ITEM

Syntax

DISABLE ITEM (*posintexpr1*;*posintexpr2*)

Description

DISABLE ITEM grays out (disables) the menu item described by its two arguments. *posintexpr1* is the number of the menu title that you create. Numbering excludes the Apple and Edit menus (File is menu number one). *posintexpr2* is the number of the item within the menu. Thus, the example would disable the fourth item in the second menu title. If *posintexpr2* is 0, the menu title and all its items are disabled. Like CHECK ITEM, DISABLE ITEM is local to the menu bar.

❖ *Restoring defaults:* You can use MENU BAR to restore the original menu bar, as you defined it in the Design environment thus restoring all menu and item defaults.

You enable a disabled menu item with ENABLE ITEM. If you plan to have a particular item disabled all or most of the time, it's easier to disable it in the menu editor. This command does the same thing as toggling off the Enable box in the menu editor.

Example

```
DISABLE ITEM(2;4)
```

References

CHECK ITEM, ENABLE ITEM, MENU BAR.

DISPLAY RECORD

Syntax

DISPLAY RECORD «(*filename*)»

Description

DISPLAY RECORD displays the current record using the current input layout only as long as another event does not change the display. If you created a window, 4th Dimension will display the record in it. Use DISPLAY RECORD to display records individually with input layout.

Normally, you execute DISPLAY RECORD in a global procedure. You can, however, place it in a layout procedure when the record you want to display belongs to a file other than the layout's file.

To display a record through an input layout other than the default input layout, specify the layout with INPUT LAYOUT before DISPLAY RECORD. If you name a different layout for display, you must specify the default input layout near the end of the routine. If you don't specify *filename,* the command uses the default file. The example includes a three-second timeout loop to hold the record on the screen.

Example

```
  `DisplayLoop: Displays records serially
DEFAULT FILE([Models])
ALL RECORDS
INPUT LAYOUT("dDisplay")
While (Not(End selection))
  DISPLAY RECORD
     `Pause display for 3 seconds. Works near midnight.
  vNow:=Current time
  While(Abs(Current time-vNow)<3)
  End while
  NEXT RECORD
End while
INPUT LAYOUT("InOut2")    `Restore std layout
```

References

DISPLAY SELECTION, INPUT LAYOUT, OPEN WINDOW.

DISPLAY SELECTION

Syntax

DISPLAY SELECTION «(*)|(*filename*«;*»)»

Description

DISPLAY SELECTION displays all records in the current selection of *filename* through the current output layout. If the selection contains only one record and you do not include the optional asterisk, the record appears in the input layout. If you include the asterisk, 4th Dimension will display a one record selection in the output layout.

To see a record in a multiple record selection in an input layout, the user may double-click on the desired record. If you have opened a window, 4th Dimension will display the selection in it. If you don't specify *filename,* the command uses the default file. DISPLAY SELECTION executes the Before and During phases simultaneously, once for each record.

❖ *DISPLAY or MODIFY:* DISPLAY SELECTION and MODIFY SELECTION work alike in all respects except one. That is, when the user double-clicks a record, both display the record in the current input layout, but only MODIFY SELECTION lets the user modify the contents of the record.

Normally, you execute DISPLAY SELECTION in a global procedure. You can, however, place it in a layout procedure when the selection you want to display belongs to a file other than the layout's file. If you do not include a button in your layout, 4th Dimension will supply its own. For output layouts, the button "Done" appears in the lower right corner of the window. For input layouts, the 4th Dimension button panel appears. When creating an output layout, place an Accept button in the footer area of the layout.

If the user clicks a button or selects a menu item, the layout procedure for the currently displayed layout will be activated. You can then test Menu selected or the button and perform an action such as a sort or search.

The user can scroll through the selection and click on a record to select it. Clicking a different record deselects the first record and selects the second record. To select a contiguous group of records, click the first record and Shift-click the last record. To select noncontiguous records, Command-click each desired record. To display any record in the current input layout, double-click on the record.

After DISPLAY SELECTION, you can find the records selected by the user by using the set named UserSet, that 4th Dimension automatically creates whenever the user makes a selection. If you want to save this set, you should first do a UNION operation between UserSet and an empty set of the same file, because at the next DISPLAY SELECTION, 4th Dimension first erases and then recalculates UserSet. Further, UserSet, as a system set, does not belong to any file.

❖ *Note:* DISPLAY SELECTION leaves the current record as "undefined." Use FIRST RECORD, LAST RECORD, or ALL RECORDS to reselect it.

Example

```
DEFAULT FILE([Accounts])
ALL RECORDS
DISPLAY SELECTION ([Accounts];*)
CREATE EMPTY SET("MyPicks")
UNION("UserSet";"MyPicks";"MyPicks")
USE SET("MyPicks")
PRINT SELECTION
CLEAR SET("MyPicks")
```

References DISPLAY RECORD, MODIFY SELECTION, OPEN WINDOW, USE SET.

During

Syntax During

Description 4th Dimension generates a During phase for an input layout or dialog layout procedure:

- □ when the user modifies a field and moves to another field
- □ when the user modifies a variable and moves to another variable
- □ when the user clicks a button, radio button, or a check box
- □ when the user clicks in an external area
- □ when the user selects from a custom menu, but not from the Apple or Edit menus
- □ when the user makes a selection from a scrollable area
- □ within a subrecord, only when the user enters data for that particular subrecord
- □ within a subrecord, when the subrecord must be redrawn
- □ when the user presses the Enter key

For printing, 4th Dimension generates Before and During phases for each record. When doing any calculation on a page-by-page basis (like summing a field), execute these calculations in the During phase. If you execute them in the Before phase, the calculations will include the first record on the next page, making your page figures incorrect.

When displaying with DISPLAY SELECTION or MODIFY SELECTION, 4th Dimension calls Before and During simultaneously for each record. In a subfile layout procedure, During goes TRUE when you modify a subfield, regardless of whether the layout is Full Page or Multi-line. For details on the part of During in the execution cycle, see *4th Dimension Programmer's Reference.*

❖ *Testing execution phases:* Typically, a developer will write If...End if statements for each phase or a Case statement in which two or three of the phases are the cases.

Example

```
If(During)
  LOAD LINKED RECORD(CustNo;[Client]ID No)
  Name:=[Client]Client
  City:=[Client]City
End if
```

References After, Before, In break, In footer, In header.

ENABLE BUTTON

Syntax ENABLE BUTTON (*buttonvar*)

Description ENABLE BUTTON reactivates a button specified by *buttonvar* and previously disabled by DISABLE BUTTON. *buttonvar* is the button variable name. Like BUTTON TEXT and DISABLE BUTTON, ENABLE BUTTON is local to the layout. Buttons include Accept buttons, Don't Accept buttons, Button buttons, Radio buttons, and Check boxes. Place ENABLE BUTTON only in a layout or dialog procedure.

Example

```
If (Before)
  vWholename:=[RoloText]First+" "+[RoloText]Last
  vPNum:=[RoloText]Extension
  vRem:=[RoloText]Comment
  If (gRecNo=1)
    DISABLE BUTTON(bPrevious)
  Else
    ENABLE BUTTON(bPrevious)
  End if
  If (gRecNo=gTotRecs)
    DISABLE BUTTON(bNext)
  Else
    ENABLE BUTTON(bNext)
  End if
End if
```

References BUTTON TEXT, DISABLE BUTTON.

ENABLE ITEM

Syntax

ENABLE ITEM (*posintexpr1;posintexpr2*)

Description

ENABLE ITEM enables a menu command disabled by the DISABLE ITEM procedure or disabled in the Design environment. *posintexpr1* is the number of the menu title and *posintexpr2* the number of the item within that bar. Numbering excludes the Apple and Edit menus (File is menu number one). The example would enable the fourth item in the second menu title. Like CHECK ITEM and DISABLE ITEM, ENABLE ITEM is local to the menu bar.

Use MENU BAR to restore the original menu bar as you defined it in the Design environment, thus restoring all menu and item defaults.

If you plan to have a particular item enabled all or most of the time, it's easier to enable it in the menu editor. This command does the same thing as checking the Enable box in the menu editor. If *posintexpr2* is zero, the title is enabled and the individual items reflect the state of their most recent enabling or disabling.

Example

```
ENABLE ITEM(2;4)
```

References

CHECK ITEM, DISABLE ITEM, MENU BAR, Menu selected.

End selection

Syntax

End selection «(*filename*)»

Description

End selection returns TRUE when the NEXT RECORD command moves the record pointer past the last record in the current selection of *filename*. If you don't specify *filename,* End selection uses the default file. If the current selection is empty, End selection returns TRUE.

Once End selection is TRUE, you can only move the record pointer back into the selection with either LAST RECORD or FIRST RECORD. PREVIOUS RECORD will not do the job.

During printing, you can test End selection in a footer. End selection returns TRUE when you print the last footer, meaning that all the records in the selection have been printed. This can come in handy when you want to print a special message at the end of a printout. The example moves the record pointer through the current selection, displaying each record as it goes. Moving past the last record sets End selection to TRUE, ending the loop.

Example

```
  `Forward: Display each record fast
DEFAULT FILE([Addresses])
ALL RECORDS
While (Not(End selection))
  DISPLAY RECORD
  NEXT RECORD
End while
```

References

Before selection, FIRST RECORD, LAST RECORD, NEXT RECORD, PREVIOUS RECORD.

End subselection

Syntax

End subselection (*subfilename*)

Description

End subselection returns TRUE when NEXT SUBRECORD moves the subrecord pointer for the current record past the last subrecord in the current subselection of *subfilename*. The example shows a portion of a procedure that writes file data, including subrecord data to a text file, using SEND PACKET.

Example

```
.
.
.
While (Not(End subselection([I.Card]Phone))) `Write subfile items
  SEND PACKET([I.Card]Phone'PhNum+Char(13))
  SEND PACKET([I.Card]Phone'Comment+Char(13))
  NEXT SUBRECORD([I.Card]Phone)
End while
```

References

Before subselection, FIRST SUBRECORD, LAST SUBRECORD, NEXT SUBRECORD, PREVIOUS SUBRECORD.

ERASE WINDOW

Syntax ERASE WINDOW

Description ERASE WINDOW clears the contents of the window created by OPEN WINDOW and returns the cursor to the upper left corner of the window, the GO TO XY(0;0) position. Don't confuse ERASE WINDOW, which clears the contents of a window with CLOSE WINDOW, which takes the window off the screen. The example causes the window to display a maximum of five lines at a time.

Example

```
 `Demo ERASE WINDOW
gLine:=0
DEFAULT FILE([Catalog])
ALL RECORDS
OPEN WINDOW(5;40;250;300;0;"Inventory Window")
While (Not(End selection))
  gLine:=gLine+1
  MESSAGE([Catalog]Description+Char(13))
  NEXT RECORD
  If(Mod(gLine;5)=0)
    ERASE WINDOW
  End if
End while
```

References CLOSE WINDOW, GO TO XY, OPEN WINDOW.

EXECUTE

Syntax EXECUTE (*strexpr*)

Description 4th Dimension executes *strexpr* as a statement. The statement cannot contain local variables. You can include the name of a procedure in an EXECUTE statement.

strexpr can contain any command string except the control of flow statements If, While, Case, Else, and their End statements. If you want the value of *strexpr* to include double quotation marks, include them as Char(34) expressions.

You can use EXECUTE to build 4th Dimension instructions, like file functions "on the fly." For example, because you cannot pass filenames as parameters, as in MyProc([MyFile1]), you must pass them as strings—MyProc("MyFile1")—and use EXECUTE to build new commands: EXECUTE("FIRST RECORD(["+$1+"])").

The first example would create and execute the following statement, if gOp were a plus sign:

```
vResult:=Field1+Field2
```

The second example will execute the following statement:

```
v{i}:=Position("a";Field1)
```

Examples

```
EXECUTE("vResult:=Field1"+gOp+"Field2")
EXECUTE("v{i} := Position("+Char(34)+"a"+Char(34)+"; Field1)")
```

Reference None.

Exp

Syntax Exp (*numexpr*)

Description Exp raises the natural log base (e = 2.71828182845904524) by the power of *numexpr*. Exp is the inverse function of Log. The example assigns 7.38905609893065023, the exponential of 2, to v. (The log of v is 2.)

Example

```
v:=Exp(2)
```

References Arctan, Cos, Log, Sin, Tan.

EXPORT DIF

Syntax

EXPORT DIF «(«*filename*;»*docname*)»

Description

EXPORT DIF writes the current selection from *filename* to disk as a DIF document named *docname,* using the current output layout. If you supply a null string for *docname,* the standard file dialog box appears, so that the user can choose the file to save to. You can get the name of the file opened from the system variable Document. If you don't specify *filename,* EXPORT DIF uses the default file.

All EXPORT commands transfer fields and variables to *docname* based on the order they appear in the output layout. The first field or variable to be exported is the one closest to the top of the layout. If two fields are equally close, the leftmost is exported first. Make sure you choose an output layout that saves the fields and/or variables you want saved in the order you want them saved. You can use a layout procedure as a part of this process.

During the export operation, 4th Dimension displays a window to show the progress of the export. The window also includes a Stop button. If the user clicks the Stop button, the exporting operation stops. You can test the OK system variable to see what the user has done. If the export is completed, OK returns 1. If the user interrupts the export, OK returns 0. The export always takes place under the current ASCII map.

If you choose a special export layout, as in the example, you may want to respecify the database's default input layout near the end of the routine.

Example

```
DEFAULT FILE([Addresses])
SEARCH([Addresses]City="Palo Alto")
OUTPUT LAYOUT("Exporter")
EXPORT DIF("Palo Alto Addresses")
OUTPUT LAYOUT("Output1")
```

References

EXPORT SYLK, EXPORT TEXT, IMPORT DIF, IMPORT SYLK, IMPORT TEXT, RECEIVE PACKET, SEND PACKET, USE ASCII MAP.

EXPORT SYLK

Syntax

EXPORT SYLK «(«*filename*;»*docname*)»

Description

EXPORT SYLK writes the current selection from *filename* to disk as a SYLK document named *docname,* using the current output layout. If you supply a null string for *docname,* the standard file dialog box appears, so that the user can choose the file to save to. You can get the name of the file opened from the system variable Document. If you don't specify *filename,* EXPORT SYLK uses the default file.

All EXPORT commands transfer fields and variables to *docname* based on the order they appear in the output layout. The first field or variable to be exported is the one closest to the top of the layout. If two fields are equally close, the leftmost is exported first. Make sure you choose an output layout that saves the fields and/or variables you want saved in the order you want them saved. You can use a layout procedure as a part of this process.

During the export operation, 4th Dimension displays a window to show the progress of the export. The window also includes a Stop button. If the user clicks the Stop button, the exporting operation stops. You can test the OK system variable to see what the user has done. If the export is completed, OK returns 1. If the user interrupts the export, OK returns 0. The export always takes place under the current ASCII map.

Example

```
DEFAULT FILE([Addresses])
SEARCH([Addresses]City="Palo Alto")
OUTPUT LAYOUT("Exporter")
EXPORT SYLK("Palo Alto Addresses")
OUTPUT LAYOUT("Output1")
```

References

EXPORT DIF, EXPORT TEXT, IMPORT DIF, IMPORT SYLK, IMPORT TEXT, RECEIVE PACKET, SEND PACKET, USE ASCII MAP.

EXPORT TEXT

Syntax EXPORT TEXT «(«*filename*;»*docname*)»

Description EXPORT TEXT writes the current selection from *filename* to disk as a text document (ASCII file) named *docname,* using the current output layout. If you supply a null string for *docname,* the standard file dialog box appears, so that the user can choose the file to save to. You can get the name of the file opened from the system variable Document. If you don't specify *filename,* EXPORT TEXT uses the default file.

All EXPORT commands transfer fields and variables to *docname* based on the order they appear in the output layout. The first field or variable to be exported is the one closest to the top of the layout. If two fields are equally close, the leftmost is exported first. Make sure you choose an output layout that saves the fields and/or variables you want saved in the order you want them saved. You can use a layout procedure as a part of this process.

During the export operation, 4th Dimension displays a window to show the progress of the export. The window also includes a Stop button. If the user clicks the Stop button, the exporting operation stops. You can test the OK system variable to see what the user has done. If the export is completed, OK returns 1. If the user interrupts the export, OK returns 0. The export always takes place under the current ASCII map.

The default field delimiter is the TAB character (ASCII 9). The default record delimiter is the carriage return character (ASCII 13). You can change these by assigning values to the two system delimiter variables, FldDelimit and RecDelimit.

Important

Because text fields can contain carriage returns, don't use a carriage return as a delimiter if you are exporting text fields.

Fields are variable length. If you need fixed length fields, create a layout procedure that takes fields and creates fixed length variables. The layout should contain only variables and no fields. Here is an example:

```
ExportVar1:=Substring(MyField1+(" "*20);1;20)
```

Example

```
DEFAULT FILE([Addresses])
SEARCH([Addresses]City="Palo Alto")
OUTPUT LAYOUT("Exporter")
EXPORT TEXT("Palo Alto Addresses")
OUTPUT LAYOUT("Output1")
```

References EXPORT DIF, EXPORT SYLK, IMPORT DIF, IMPORT SYLK, IMPORT TEXT, RECEIVE PACKET, SEND PACKET, USE ASCII MAP.

False

Syntax False

Description False always returns the Boolean value FALSE.

Example

```
  `False demo
MyVar := False
If(Not(MyVar))
  str:="I'm false."
Else
  str:="I'm true."
End if
ALERT(str)
```

References Not, True.

FIRST RECORD

Syntax

FIRST RECORD «(*filename*)»

Description

FIRST RECORD sets the record pointer to the first record in current selection of *filename* and loads the record into memory, making it the current record. All search, selection, and sorting procedures automatically set the record pointer to the first record. Use FIRST RECORD when you need to move the pointer back to the first record. If you don't specify *filename,* FIRST RECORD uses the default file. If the current selection is empty, FIRST RECORD has no effect.

The example displays a file of models. To keep the display loop going, the program tests for End selection, meaning that the NEXT RECORD has pushed the pointer past the last record. If this condition is TRUE, FIRST RECORD returns the pointer to the first record, so that the display can continue uninterrupted.

Example

```
 `DisplayLoop: Displays records
DEFAULT FILE([Models])
ALL RECORDS
vTimes:=1
While (vTimes <= 5)
   While (Not(End selection))
      DISPLAY RECORD
      NEXT RECORD
   End while
   FIRST RECORD `Set record pointer to first record
   vTimes:=vTimes+1
End while
```

References

Before selection, End selection, LAST RECORD, NEXT RECORD, PREVIOUS RECORD.

FIRST SUBRECORD

Syntax

FIRST SUBRECORD (*subfilename*)

Description

FIRST SUBRECORD makes the first subrecord in the current subselection of *subfilename* the current subrecord. *subfilename* refers to a subfile belonging to the current record. All search, selection, and sorting procedures automatically set the record pointer to the first subrecord. If the subselection is empty, FIRST SUBRECORD has no effect. In the example, FIRST SUBRECORD loads the first subrecord in the Phone subfile.

Example

```
 `SendSub: Writes a file, including subfiles to disk
SET CHANNEL(10;"SubFile.Txt")
DEFAULT FILE([I.Card])
ALL RECORDS
While (Not(End selection))
  SEND PACKET([I.Card]Name+Char(13))
  ALL SUBRECORDS([I.Card]Phone)
  FIRST SUBRECORD([I.Card]Phone)
  While (Not(End subselection([I.Card]Phone)))
    SEND PACKET([I.Card]Phone'PhNum+Char(13))
    SEND PACKET([I.Card]Phone'Comment+Char(13))
    NEXT SUBRECORD([I.Card]Phone)
  End while
  SEND PACKET("******"+Char(13))  `end of record marker
  NEXT RECORD
End while
SEND PACKET("%%%%")  `end of file marker
SET CHANNEL(11)  `close file
```

References

Before subselection, End subselection, LAST SUBRECORD, NEXT SUBRECORD, PREVIOUS SUBRECORD.

FONT

FONT has two syntaxes. With the first, you specify the font through a numeric expression and with the second through a string expression. FONT comes in handy for highlighting variables in reports. For example, you could use a different font for all customers owing over $10,000. The numeric codes and their font names are presented in Table II-1 below and in Appendix F, "4th Dimension and Macintosh Codes."

Table II-1
Font numbers

Font number	Font	Font number	Font
0	systemFont (Chicago)	9	toronto
1	applFont	11	cairo
2	newYork	12	losAngeles
3	geneva	20	times
4	monaco	21	helvetica
5	venice	22	courier
6	london	23	symbol
7	athens	24	taliesin
8	sanFran		

Syntax 1 FONT (*var,posintexpr*)

Description 1 FONT sets the font for *var,* where *var* is a variable or scrollable area in a layout. *posintexpr* is any valid Apple font code.

Example 1 **FONT**(Salary;2)

Syntax 2 FONT (*var,strexpr*)

Description 2 FONT sets the font for *var,* where *var* is a variable or scrollable area in a layout. *strexpr* is any valid Apple font name.

Example 2 **FONT**(Salary;"Chicago")

References FONT SIZE, FONT STYLE.

FONT SIZE

Syntax FONT SIZE (*var;posintexpr*)

Description FONT SIZE sets the font size for *var,* where *var* is a variable or scrollable area in a layout. *posintexpr* is any integer between 1 and 128. If the size doesn't exist, characters will scale to the size given by *posintexpr.* If *posintexpr* is 0, the font size reverts to the size originally defined in the layout.

Example

```
FONT SIZE(Salary;12)
```

References FONT, FONT STYLE.

FONT STYLE

Syntax FONT STYLE (*var;posintexpr*)

Description FONT STYLE sets the font style for *var,* where *var* is a variable or scrollable area in a layout. *posintexpr* is Macintosh font style code. By adding codes together, you can display characters in combined style. If you want one style (like bold or italic), just give the style number. The numeric codes for FONT STYLE are presented in Table II-2 below and in Appendix F, "4th Dimension and Macintosh Codes."

Table II-2
Font styles

Style number	Style	Style number	Style
1	Bold	16	Shadow
2	Italic	32	Condensed
4	Underline	64	Extended
8	Outline		

Example

```
Bold:=1
Outline:=8
FONT STYLE(Salary;Bold+Outline)
```

References FONT, FONT SIZE.

FORM FEED

Syntax FORM FEED

Description FORM FEED issues a form feed, pushing paper to the next top of form position. FORM FEED is useful for creating breaks in reports. PRINT LAYOUT must be followed by the FORM FEED command to print.

❖ *Automatic form feeds:* PRINT LABEL and PRINT SELECTION automatically issue a form feed after the last element in the report is printed.

Example

```
FORM FEED
```

References PRINT LABEL, PRINT LAYOUT, PRINT SELECTION, PRINT SETTINGS.

GET HIGHLIGHTED TEXT

Syntax GET HIGHLIGHTED TEXT (*var*|*fieldname*;*numvar1*;*numvar2*)

Description GET HIGHLIGHTED TEXT places the position of the first character of selected text in *numvar1* and the position of the last character of the selected text plus one in *numvar2*. If *numvar1* and *numvar2* are equal, the user has no selected text and the insertion point is before the *numvar1* character. GET HIGHLIGHTED TEXT can work on *var* (a variable) or on *fieldname* (a field).

In the example, GET HIGHLIGHTED TEXT finds the start and end positions of the selected text. Then, the Substring function extracts the text.

Example

```
GET HIGHLIGHTED TEXT(MyText;v1;v2)
MySel := Substring(MyText;v1;(v2-v1))
```

References HIGHLIGHT TEXT, Substring.

GO TO FIELD

Syntax GO TO FIELD (*fieldname*)

Description GO TO FIELD selects the field named by *fieldname* in an input layout. Thus, it is like clicking a field. The example selects the Part number field for entry before the layout appears on the screen. Because the example is a layout procedure, no filename prefix appears before the *fieldname* PartNo.

Example

```
  `Layout procedure: Order Form
If (Before)
  GO TO FIELD(PartNo)
End if
```

References Modified, REJECT.

GO TO XY

Syntax GO TO XY (*posintexpr1;posintexpr2*)

Description GO TO XY places the cursor at the specified character position in a created window. *posintexpr1* is the X-axis value and *posintexpr2* is the Y-axis value. The top left corner is 0, 0. 4th Dimension automatically places the cursor at 0, 0 when it creates a window and when you call ERASE WINDOW. Use GO TO XY to position messages in a text window created with OPEN WINDOW. The example indents each line by 5 columns.

❖ *Monaco font:* GO TO XY handles precise cursor positioning, because 4th Dimension uses the nonproportional Monaco font to display text in a custom window. All characters in this font are the same width.

Example

```
  `Window5: Demo GO TO XY
gLine:=0
DEFAULT FILE([Catalog])
ALL RECORDS
OPEN WINDOW(5;40;250;300;0;"Window")
While (Not(End selection))
  gLine:=gLine+1
  GO TO XY(5;gLine)
  MESSAGE([Catalog]Description+Char(13))
  NEXT RECORD
End while
```

References ERASE WINDOW, OPEN WINDOW.

GRAPH

Syntax

GRAPH (*var*;*posintexpr*;*strvarX* | *subfieldnameX*;*numvarY* | *subfieldnameY*)

Description

GRAPH can graph data from subfields and/or from variables. The GRAPH syntax includes four elements:

- □ the graph area variable (*var*)
- □ a numeric expression stating the type of graph (*posintexpr*)
- □ an X-coordinate element, always an alphanumeric type (*strvarX* or *subfieldnameX*)
- □ a numeric Y-coordinate element (*numvarY* or *subfieldnameY*); you can have as many as eight when programming

var is the name you gave the graph area in the layout that displays the graph. (You must have prepared a graph area in a layout.) *posintexpr* is in the range of 1 through 8 and represents the type of graph to be drawn (see Table II-3). Each coordinate can be either a variable or a subfield. The X coordinate element must be an alphanumeric field, because it labels the x-axis. The Y coordinate element represents the values to be graphed on the y-axis and must be of a numeric type.

❖ *Graphing variable arrays:* When graphing an array of variables, you must include the number of elements in the array. This number is kept in the zero array element. For example, if you graph v1 through v32, assign 32 to v0.

Because the GRAPH command functions only in terms of a layout's graph area, *var*, place GRAPH only in the layout procedure for the layout to which *var* belongs.

Table II-3
Graph types

Graph type	**Graph number**	**Graph type**	**Graph number**
Column	1	Area	5
Proportional column	2	Scatter	6
Stacked column	3	Pie	7
Line	4	Picture	8

The first example sends data to a graph area named gSales to be drawn as a bar chart. It graphs by sales representative names along the x-axis taking data from the History file's subfield, Info. The y-axis compares actual sales against goals.

❖ *Pie charts:* Pie charts only graph the first Y coordinate.

Examples

```
GRAPH(gSales;1;[History]Info'SalesRep;[History]Info'Sales;[History]Info'Goal)
```

```
  `Display an example graph.
If (During)
  X0:=2
  X1:="Profit"
  X2:="Sales"
  Ya:="1990"   ` The Ya label
  Yb:="1991"   ` The Yb label
  Ya0:=2
  Ya1:=30
  Ya2:=40
  Yb0:=2
  Yb1:=50
  Yb2:=75
  GRAPH(vGraph;1;X;Ya;Yb)
End if
```

References

GRAPH FILE, OUTPUT LAYOUT.

GRAPH FILE

Syntax 1 GRAPH FILE («*filename*;»*posintexpr*;*fieldnameX*;*fieldnameY*)

Syntax 2 GRAPH FILE «(*filename*)»

Description 1 GRAPH FILE graphs the data in selected fields of a file. *posintexpr* is in the range of 1 through 8 and represents the type of graph to be drawn (see Table II-3). *fieldnameX* must be an alphanumeric field and labels the x-axis. You can have as many as eight *fieldnameY* (but only five in the User environment). *fieldnameY* represents the values to be graphed on the y-axis and must be of a numeric type.

❖ *Pie charts:* Pie charts graph only the first Y coordinate.

Both syntaxes let the user copy the graph from the screen to the Clipboard (and if you wish, into the Scrapbook). If you don't specify *filename,* GRAPH FILE uses the default file.

❖ *Graph menus:* Both syntaxes bring up two menus (Pictures and Graph type), so the user can regraph and paste.

❖ *Note:* GRAPH FILE will graph only the first 100 records in the selection.

Description 2 GRAPH FILE without graphing arguments brings up the standard graph window, so that the user can select fields, type, and so on. If you don't specify *filename,* GRAPH FILE applies to the default file.

Example

```
DEFAULT FILE([SALES])
ALL RECORDS
If(Records in selection>0)
      `Create a graph under Syntax 1
   GRAPH FILE(2;Seller;ActualSales;Goal)
      `Let the user create a graph (Syntax 2)
   GRAPH FILE
End if
```

Reference GRAPH.

HIGHLIGHT TEXT

Syntax

HIGHLIGHT TEXT (*var* | *fieldname;posintexpr1;posintexpr2*)

Description

HIGHLIGHT TEXT highlights a group of characters in *var* (a variable) or in *fieldname* (a field). *posintexpr1* represents the first character position you wish to highlight. *posintexpr2* represents the last character plus one in the group of characters to be highlighted. You can position the insertion point in a string without highlighting a character by giving the same value to *posintexpr1* and *posintexpr2*. HIGHLIGHT TEXT works on all fields.

If *posintexpr2* is greater than the number of characters between *posintexpr1* and the end of the text, 4th Dimension selects all characters between *posintexpr1* and the end of the text.

You can call HIGHLIGHT TEXT for a field before the cursor is in that field. When the cursor enters the field, the highlighted text will appear. Place HIGHLIGHT TEXT in a layout procedure only.

Examples

```
HIGHLIGHT TEXT(vText;3;3)   `insertion before 3rd character
HIGHLIGHT TEXT(vText;2;3)   `highlight the 2nd character
HIGHLIGHT TEXT(vText;4;32767)   `highlight 4th character and all the rest
```

Reference

GET HIGHLIGHTED TEXT.

If...Else...End if

Syntax

If (*boolexpr*)
{*statement(s)*}
{Else
{*statement(s)*}}
End if

Description

If executes *statement(s)* following it when *boolexpr* returns TRUE. If *boolexpr* returns FALSE, the *statement(s)* following Else executes. The Else clause is optional.

Important

Be sure to guard the precedence in Boolean expressions. If you have multiple conditions, enclose each condition in its own set of parentheses. For example: ((B>1) & (C<0)). A second example is B>(3*5).

You can nest If statements within If statements, as long as the close of an inner If statement does not appear after the close of an outer If statement. All If statements must begin and end within a given routine. (You cannot distribute parts of an If statement over two or more routines.)

The example tests for an undefined variable. If the variable is undefined, the statements following the If clause ask for a number and save the variable. If the variable already exists, the Else clause puts up an Alert box, stating the value of the variable.

Example

```
  `Creates a document for variables if none exists
CLEAR VARIABLE("MyVar")
LOAD VARIABLE("MyDoc";MyVar)
If (Undefined(MyVar))
  CONFIRM("'MyDoc' doesn't exist. Create it?")
  If (OK=1)
    MyVar:=Request("Enter value of 'MyVar'")
    SAVE VARIABLE("MyDoc";MyVar)
  End if
Else
  ALERT("Value of 'MyVar' is "+MyVar)
End if
```

References

Case of...Else...End case, While...End while.

IMPORT DIF

Syntax

IMPORT DIF «(«*filename*;» *docname*)»

Description

IMPORT DIF loads the DIF document specified by *docname* into *filename* in accordance with the current input layout. The command saves each record as it is imported. If you supply a null string for *docname*, the standard file dialog box appears, so that the user can choose the desired file to import. You can get the name of the file opened from the system variable Document. If you don't specify *filename*, IMPORT DIF uses the default file.

Be sure to import through an input layout that contains the correct fields and/or variables in the correct order. All IMPORT commands transfer data into layout fields and variables based on the order they appear in the input layout. The first data item read from the *docname* document is stored in the topmost field or variable in the input layout. If two fields or variables are equally close, the leftmost is imported first. You can make use of a layout procedure when importing data. 4th Dimension executes the layout procedure for each record before saving the record.

During the import operation, 4th Dimension displays a window to show the progress of the import. The window also includes a Stop button. If the user clicks the Stop button, importing stops, but previously imported records are already saved. You can test the OK system variable to see what the user has done. If the import is completed, OK returns 1. If the user interrupts the import, OK returns 0. The import always takes place under the current ASCII map.

If you choose a special import layout, as in the example, you may need to respecify the database's default input layout near the end of the routine.

Example

```
  `GetDIF; Mail List db, Addresses file
DEFAULT FILE([Addresses])
INPUT LAYOUT("Importer")
MESSAGE("Now loading DIF file.")
IMPORT DIF("Mail.DIF")
INPUT LAYOUT("Input1")  `Reset default
```

References

EXPORT DIF, EXPORT SYLK, EXPORT TEXT, IMPORT SYLK, IMPORT TEXT, RECEIVE PACKET, SEND PACKET, USE ASCII MAP.

IMPORT SYLK

Syntax

IMPORT SYLK «(«*filename*;» *docname*)»

Description

IMPORT SYLK loads the SYLK document specified by *docname* into *filename* in accordance with the current input layout. The command saves each record as it is imported. If you supply a null string for *docname*, the standard file dialog box appears, so that the user can choose the desired file to import. You can get the name of the file opened from the system variable Document. If you don't specify *filename,* IMPORT SYLK uses the default file.

Be sure to import through an input layout that contains the correct fields and/or variables in the correct order. All IMPORT commands transfer data into layout fields and variables based on the order they appear in the input layout. The first data item read from the *docname* document is stored in the topmost field or variable in the input layout. If two fields or variables are equally close, the leftmost is imported first. You can make use of a layout procedure when importing data. 4th Dimension executes the layout procedure for each record before saving the record.

During the import operation, 4th Dimension displays a window to show the progress of the import. The window also includes a Stop button. If the user clicks the Stop button, the importing operation stops, but previously imported records are already saved. You can test the OK system variable to see what the user has done. If the import is completed, OK returns 1. If the user interrupts the import, OK returns 0. The import always takes place under the current ASCII map.

If you choose a special import layout, as in the example, you may need to respecify the database's default input layout near the end of the routine.

Example

```
 `GetSYLK; Mail List db, Addresses file
DEFAULT FILE([Addresses])
INPUT LAYOUT("Importer")
MESSAGE("Now loading SYLK file.")
IMPORT SYLK("Mail.SYLK")
INPUT LAYOUT("Input1") `Reset default
```

References

EXPORT DIF, EXPORT SYLK, EXPORT TEXT, IMPORT DIF, IMPORT TEXT, RECEIVE PACKET, SEND PACKET, USE ASCII MAP.

IMPORT TEXT

Syntax

IMPORT TEXT «(«*filename*;»*docname*)»

Description

IMPORT TEXT loads the text (ASCII) document specified by *docname* into *filename* in accordance with the current input layout. The command saves each record as it is imported. If you supply a null string for *docname,* the standard file dialog box appears, so that the user can choose the desired file to import. You can get the name of the file opened from the system variable Document. If you don't specify *filename,* IMPORT TEXT uses the default file.

Be sure to import through an input layout that contains the correct fields and/or variables in the correct order. All IMPORT commands transfer data into layout fields and variables based on the order they appear in the input layout. The first data item read from the *docname* document is stored in the topmost field or variable in the input layout. If two fields or variables are equally close, the leftmost is imported first. You can make use of a layout procedure when importing data. 4th Dimension executes the layout procedure for each record before saving the record.

During the import operation, 4th Dimension displays a window to show the progress of the import. The window also includes an Stop button. If the user clicks the Stop button, importing stops, but previously imported records are already saved. You can test the OK system variable to see what the user has done. If the import is completed, OK returns 1. If the user interrupts the import, OK returns 0. The import always takes place under the current ASCII map.

If you choose a special import layout, as in the example, you may need to respecify the database's default input layout near the end of the routine.

By default, the TAB (ASCII 9) is the field delimiter and the carriage return (ASCII 13) character is the record delimiter. Before calling IMPORT TEXT you can change the delimiters for fields and records by assigning new values to the FldDelimit and RecDelimit system variables using the appropriate ASCII character codes.

Example

```
  `GetText; Mail List db, Addresses file
DEFAULT FILE([Addresses])
INPUT LAYOUT("Importer")
MESSAGE("Now loading text file.")
IMPORT TEXT("Mail.Text")
INPUT LAYOUT("Input1") `Reset default
```

References

EXPORT DIF, EXPORT SYLK, EXPORT TEXT, IMPORT DIF, IMPORT SYLK, RECEIVE PACKET, SEND PACKET, USE ASCII MAP.

In break

Syntax

In break

Description

In break returns TRUE when 4th Dimension prints the break area of a layout. That is, when a sort level changes. 4th Dimension assigns the grand total the sort level 0, the first sorted field level 1, and so on through the next to the last sort. The last sort is not counted.

To generate figures in a break, you must set up a variable in the layout's break area (the area between the D and B lines) and assign the particular figures to the variable within the In break portion of the layout procedure. Do not put In break in Before or During. Each of these is a distinct phase of the execution cycle and should stand alone. Place In break in an output layout procedure only.

In break is necessary for printing a sorted selection with subtotals. In this case, when In break returns TRUE, you can print a subtotal, using the Subtotal function. You can test the level of the subtotal with the Level function, as in the example.

Important

The Subtotal function must be present to print any break.

4th Dimension breaks on every sorted field except the last one. For example, if you sort on [Sales]Region, [Sales]Rep, and [Sales]Customer, 4th Dimension generates a level 1 break for [Sales]Region and a level 2 break for [Sales]Rep. [Sales]Customer has no break level. To generate a level 3 break for [Sales]Customer, do a fourth sort, even if it's on [Sales]Customer once again.

Example

```
 `Report3 output layout procedure
If (Before)
  gQuarter:=[Income]Quarter
  vStr1:=""
  vStr2:=""
End if
If (In break)
  Case of
    : (Level=1)
     vStr1:="Subtotal for Quarter "+String(gQuarter)+": $"+String(Subtotal(Sale))
    : (Level=0)
     vStr1:="Final figures to date....    Maximum: $"+String(Max([Income]Sale))
     vStr2:="                 Total: $"+String(Sum(Sale))
  End case
End if
```

References

Before, During, In footer, In header, Level, SORT, SORT SELECTION, Subtotal.

In footer

Syntax In footer

Description In footer returns TRUE when 4th Dimension prints the footer area of a layout. That is, at the end of every page. In footer is useful if you want to set up one or more variables within the footer area (the area between the B and F lines). Through variables, you can supply information like page number, date, and so on. Do not put In footer in Before, During, or other execution cycle phases. Each of these is a distinct phase of the execution cycle and should stand alone. Place In footer in an output layout procedure only.

❖ *Determining the end of a report:* You can use the End selection function in the footer phase of a layout procedure to determine the end of a document when using PRINT SELECTION.

Example

```
If(In footer)
  PageNo:=PageNo+1
End if
```

References Before, During, In break, In header, Level, Subtotal.

In header

Syntax In header

Description In header returns TRUE when 4th Dimension prints the header area of a layout. In header is useful for setting up one or more variables within the header area (the area between the top and the H line). Through variables, you can supply information like page number, date, and so on. Do not put In header in Before, During, or other execution cycle phases. Each of these is a distinct phase of the execution cycle and should stand alone. Place In header in an output layout procedure only .

❖ *Determining the beginning of a report:* You can use the Before selection function in the header phase of a layout procedure to determine the beginning of a document when using PRINT SELECTION.

Example

```
If(In header)
   If(Before selection)
     PageNo:=0
   End if
  PageNo:=PageNo+1
End if
```

References Before, During, In break, In footer.

INPUT LAYOUT

Syntax

INPUT LAYOUT(«*filename*;»*strexpr*)

Description

INPUT LAYOUT specifies which layout (named by *strexpr*) 4th Dimension will present for data entry to *filename*. The layout must belong to *filename*. If you don't specify *filename,* INPUT LAYOUT uses the default file.

INPUT LAYOUT keeps the specified layout as the default input layout. If you specify a different input layout than the default, be sure to specify the old default layout after the input operation, if you want to return to it as the default.

INPUT LAYOUT does not cause the layout to appear. Commands like ADD RECORD and MODIFY RECORD will present the specified layout for data entry and modification. When importing data, the input layout won't appear. It serves as an organizer and channel between the external file and a 4th Dimension database file. If you do not specify an input layout for activities like IMPORT TEXT, ADD RECORD, and MODIFY RECORD, 4th Dimension uses the default input layout.

You cannot type data into variables in a layout specified by INPUT LAYOUT (only fields serve this purpose). However, you can display information through variables and you can use button variables in an INPUT LAYOUT layout. If you want to gather information with variables, call the layout with DIALOG.

❖ *Special use of the input layout:* Both DISPLAY SELECTION and MODIFY SELECTION display a record in the current input layout if there is only one record in the current selection.

Example

```
 `GetText; Mail List db, Addresses file
DEFAULT FILE([Addresses])
INPUT LAYOUT("Importer")
MESSAGE("Now loading text file.")
IMPORT TEXT("Mail.Text")
INPUT LAYOUT("Input1") `Reset default
ALERT("Import successfully completed.")
```

References

ADD RECORD, DISPLAY RECORD, IMPORT DIF, IMPORT SYLK, IMPORT TEXT, MODIFY RECORD, OUTPUT LAYOUT, PRINT RECORD.

Int

Syntax Int (*numexpr*)

Description Int returns the largest whole number that is less than or equal to *numexpr*. If *numexpr* is negative, Int truncates it toward negative infinity.

Examples

```
x:=Int(123.4) `x gets 123
y:=Int(–123.4) `y gets –124
```

References Abs, Dec, Round, Trunc.

INTERSECTION

Syntax INTERSECTION (*strexpr1*;*strexpr2*;*strexpr3*)

Description INTERSECTION creates a set from the intersection of two sets: set1 (specified by *strexpr1)* and set2 (specified by *strexpr2*). *strexpr3* names the resulting set, set3. INTERSECTION includes only those elements common to both set1 and set2. Set3 needn't exist prior to issuing this command. In fact, if it does exist, 4th Dimension will delete the old set and recreate it after recomputing the new set. Further, Set3 can be either set1 or set2.

Example

```
INTERSECTION("New York";"Women";"NewList")
```

References CLEAR SET, DIFFERENCE, SAVE SET, UNION, USE SET.

INVERT BACKGROUND

Syntax INVERT BACKGROUND (*var*)

Description INVERT BACKGROUND operates as a toggle to invert the background behind a particular variable in a layout. INVERT BACKGROUND is local to its layout and the current record. Place INVERT BACKGROUND in the layout procedure for the layout containing *var* only. It is the display, not the data, that gets inverted. It works both when printing and when displaying.

Example

```
INVERT BACKGROUND(vDay)
```

References FONT, FONT SIZE, FONT STYLE, HIGHLIGHT TEXT.

LAST RECORD

Syntax LAST RECORD «(*filename*)»

Description LAST RECORD sets the record pointer on the last record in the current selection of *filename* and makes the last record the current record. If you don't specify *filename,* LAST RECORD uses the default file. The example loads the last record in the file and displays each record in reverse order.

Example

```
 `Backward: demo LAST RECORD & Before selection
DEFAULT FILE([Addresses])
ALL RECORDS
LAST RECORD
While (Not(Before selection))
  DISPLAY RECORD
  PREVIOUS RECORD
End while
```

References Before selection, End selection, FIRST RECORD, NEXT RECORD, PREVIOUS RECORD.

LAST SUBRECORD

Syntax LAST SUBRECORD (*subfilename*)

Description LAST SUBRECORD sets the subrecord pointer to the last subrecord and makes it the current subrecord of the current subselection of *subfilename*. *subfilename* refers to a subfile belonging to the current record. The example shows a portion of a procedure that uses LAST SUBRECORD and PREVIOUS SUBRECORD to start at the last subrecord of a subselection and move backward through the subselection.

Example

```
.
.
.
  ALL SUBRECORDS([Stats]Sales)
  LAST SUBRECORD([Stats]Sales)
  While (Not(Before subselection([Stats]Sales)))
    vReport:=vReport+String([Stats]Sales'Bucks)+Char(13)
    PREVIOUS SUBRECORD([Stats]Sales)
  End while
```

References Before subselection, End subselection, FIRST SUBRECORD, NEXT SUBRECORD, PREVIOUS SUBRECORD.

Length

Syntax Length (*strexpr*)

Description Length returns the number of characters in *strexpr*.

Example `vLength:=Length("abc") `vLength gets 3`

References Position, Substring.

Level

Syntax Level

Description Level returns the level at which the next total will take place. The range of Level is from 0 to 30. 4th Dimension assigns a number to the Level function when you print a sorted selection with PRINT SELECTION and when you call the Subtotal function in a layout procedure. The Level function returns 0 when 4th Dimension prints a grand total, 1 when 4th Dimension prints a break on the first sorted field, 2 when 4th Dimension prints a break on the second sorted field, and so on.

4th Dimension breaks on every sorted field except the last one. For example, if you sort on [Sales]Region, [Sales]Rep, and [Sales]Customer, 4th Dimension generates a level 1 break for [Sales]Region and a level 2 break for [Sales]Rep. [Sales]Customer has no break level. To generate a level 3 break for [Sales]Customer, do a fourth sort, even if it's on [Sales]Customer once again.

Use Level to customize different break levels. In the example, the level determines which strings 4th Dimension prints at each break and the subtotal amount. Place the Level function in the Break phase of a layout procedure.

Example

```
  `Report3 output layout procedure
If (Before)
  gQuarter:=[Income]Quarter
  vStr1:=""
  vStr2:=""
End if
If (In break)
  Case of
    : (Level=1)
      vStr1:="Subtotal for Quarter "+String(gQuarter)+": $"+String(Subtotal(Sale))
    : (Level=0)
      vStr1:="Final figures to date....   Maximum: $"+String(Max([Income]Sale))
      vStr2:="                                 Total: $"+String(Sum(Sale))
  End case
End if
```

References In break, In footer, SORT SELECTION, Subtotal.

LOAD LINKED RECORD

Syntax

LOAD LINKED RECORD (*fieldname1*«;*fieldname2*»)

Description

LOAD LINKED RECORD establishes a relationship between two fields (usually in two files) where *fieldname1* is the linking field (the field from which the arrow extends in the Design environment). A link, the basis of the relationship between the two files, is a pointer to the linked record. This pointer is stored in the linking record. Because the pointer is saved with the linking record, finding the linked record is extremely fast. Each link adds four bytes per linking record.

LOAD LINKED RECORD is activated the first time it's invoked for a record, and thereafter only when *fieldname1* is modified.

LOAD LINKED RECORD finds the linked record pointed to by *fieldname1,* points to it, and loads it into memory, making it the current record for its file.

❖ *Linked field:* A linked field is the specific field in the linked file, the value of which uniquely identifies its record to the linking procedure.

The Mandatory attribute applied to a linking field will require that a matching record can be linked. If the user enters data that has no corresponding record in the linked file, 4th Dimension will ask the user if the user wants to create a new record for the linked file. 4th Dimension will automatically link the new record.

If the Mandatory attribute is not set and LOAD LINKED RECORD does not find a matching record, 4th Dimension stores a null pointer with the linking record. A null pointer points to no record in the linked file and LOAD LINKED RECORD will load no record.

The optional *fieldname2* must be a field in the linked file. The type of *fieldname2* cannot be Text, Date, or Subfile. If you specify the second argument, *fieldname2,* 4th Dimension displays a two-column list of records that match the value in the linking field. The left column displays linked field values and the right column displays *fieldname2* values. If the user selects a record from the list, that record pointer is stored as the link. If there is only one match, the link is to that match, and the list does not appear. This is useful if you have more than one record containing the same field value in the linked file.

The user can also use a wildcard character (@) in the linking field, if the field is a field of type Alpha. In the example, the command

```
LOAD LINKED RECORD(Name;[Client]IdentNo)
```

lets the user enters A@ to see a list of all clients whose name begins with A along with each client's identification number.

❖ *Modifying the linking field:* If you call LOAD LINKED RECORD only in the After phase of the execution cycle, you must force the link with a field modification. You can do this by assigning the linking field to itself. For example: Name := Name.

LOAD LINKED RECORD works with links to subfiles, but you must have a link to the record-level file and to the subfile linked field. When using a link to a subrecord, you must first use LOAD LINKED RECORD to load the linked record into memory. Then do a second LOAD LINKED RECORD to the subfile. For other discussions on links, see *4th Dimension User's Guide* and *4th Dimension Programmer's Reference.*

In the example, the Name field receives data from the Client file through Client's Name field. The rest of the procedure states that, based on the link, the linking Name field will get its data from the [Client]IdentNo field and that City will get data from the [Client]City field. The procedure does not name the linking file, because the example routine is an input layout procedure.

Example

```
If(During)
  If(Modified(Name))
    LOAD LINKED RECORD(Name;[Client]IdentNo)
    ID:=[Client]IdentNo
    City:=[Client]City
  End if
End if
```

References

ACTIVATE LINK, CREATE LINKED RECORD, LOAD OLD LINKED RECORD, Old, SAVE LINKED RECORD, SAVE OLD LINKED RECORD.

LOAD OLD LINKED RECORD

Syntax

LOAD OLD LINKED RECORD (*fieldname*)

Description

When a record is modified, 4th Dimension makes the changes to a duplicate of the original record. LOAD OLD LINKED RECORD works like LOAD LINKED RECORD, except that it uses the record pointer stored in the original linking record.

You execute LOAD OLD LINKED RECORD to access the record previously linked to the linking record. If you want to modify this old linked record and save it, you must execute SAVE OLD LINKED RECORD. For other discussions on links, see *4th Dimension User's Guide* and *4th Dimension Programmer's Reference*.

Example

```
  `Layout procedure: AddOrderIn
Case of
   :(Before)
      Order No:=gOrderNo
      GO TO FIELD(PartNo)
   :(During) `User modifies linking field, creating a new linked rec
      LOAD LINKED RECORD(PartNo)
      Item:=[Catalog]Description
      ItemPrice:=[Catalog]Price
      vOnhand:=[Catalog]OnHand-Quantity
      Order Total:=ItemPrice*Quantity
   :(After) `This works in all cases--whether modified or not
         `After validation, Save the new data in linked record
         `Subtract order qty from catalog
      [Catalog]OnHand:=[Catalog]OnHand-Quantity
      SAVE LINKED RECORD(PartNo)
         `Be sure to update the old linked record,
         `Re-instate old Quantity
      LOAD OLD LINKED RECORD(PartNo)
      [Catalog]OnHand:=[Catalog]OnHand+Old(Quantity)
      SAVE OLD LINKED RECORD(PartNo)
End Case
```

References

ACTIVATE LINK, CREATE LINKED RECORD, LOAD LINKED RECORD, Old, SAVE LINKED RECORD, SAVE OLD LINKED RECORD.

LOAD RECORD

Syntax

LOAD RECORD «(*filename*)»

Description

LOAD RECORD loads the current record of *filename* into memory. See *4th Dimension Utilities and Developer's Notes* for details on this and other multi-user commands.

LOAD SET

Syntax

LOAD SET («*filename*;»*strexpr*;*docname*)

Description

LOAD SET loads a set into memory from disk. *filename* is the name of the file with which the set is associated. *strexpr* is the set name, and *docname* is the name of document to which you saved the set. If you don't specify *filename,* LOAD SET uses the default file.

The syntax gives you the option of giving the set and its document different names. A set document can contain only one set. If you supply a null string for *docname,* the standard file dialog box appears, so that the user can choose the file to load. You can save a set with the SAVE SET command.

You can test the OK system variable to make sure the set loaded correctly. If it did, OK returns 1. If not, OK returns 0.

Warning
If the data file has changed (additions and deletions), LOAD SET and USE SET may select records that were not in the original set.

Example

```
  `Export Palo Alto Text; Mail List db
  `Saves a text file of all Palo Alto customers
DEFAULT FILE([Addresses])
LOAD SET([Addresses];"Palo Alto";"Palo Alto Set")
USE SET("Palo Alto")
OUTPUT LAYOUT("Exporter")
EXPORT TEXT("Palo Alto Addresses")
CLEAR SET("Palo Alto")
OUTPUT LAYOUT("Output1")
```

References

CLEAR SET, SAVE SET, USE SET.

LOAD VARIABLE

Syntax LOAD VARIABLE (*docname*;*var* {;*})

Description LOAD VARIABLE reads all variables (*var*) into memory from a file named *docname*. You create the file and the list of variables with the SAVE VARIABLE command. If *docname* is a null string, 4th Dimension opens the standard file dialog box, so the user can select the file.

After LOAD VARIABLE, test to make sure that the variables were loaded as in the example. Trying to work with undefined variables returns an error message.

Example

```
  `Creates a document for variables if none exists
CLEAR VARIABLE("MyVar")
LOAD VARIABLE("MyDoc";MyVar)
If (Undefined(MyVar))
  CONFIRM("'MyDoc' doesn't exist. Create it?")
  If (OK=1)
    MyVar:=Request("Enter value of 'MyVar'")
    SAVE VARIABLE("MyDoc";MyVar)
  End if
Else
  ALERT("Value of 'MyVar' is "+MyVar)
End if
```

References CLEAR VARIABLE, SAVE VARIABLE, Undefined.

Locked

Syntax Locked «(*filename*)»

Description Locked works in a multi-user environment. Locked returns TRUE when a particular record already is in use by someone on the network. See *4th Dimension Utilities and Developer's Notes* for details on this and other multi-user commands.

Log

Syntax Log (*numexpr*)

Description Log returns the natural (Napierian) log of *numexpr*. Log is the inverse function of Exp. A natural log has a base of 2.71828182845904524 (*e*) and a common log has a base of 10.

To convert to common log (log10), multiply the log by 0.434294481903251828. To convert a common log to a natural log, multiply the common log by 2.30258509279404568. The example assigns the natural log of 2 (0.693147180559945309) to LogE and then converts this number to the common log of 2 (0.301029995663981195).

Example

```
LogE:=Log(2)
Log10:=LogE*0.434294481903251828
```

References Arctan, Cos, Exp, Sin, Tan.

Lowercase

Syntax Lowercase (*strexpr*)

Description Lowercase returns *strexpr* in which all alphabetic characters are in lowercase. The example capitalizes the first letter, and puts the rest of the string characters in lowercase.

Example

```
`Function Capitalize
`Puts first letter in upper case and rest in lowercase
$0:=Uppercase(Substring($1;1;1))+Lowercase(Substring($1;2;Length($1)-1))
```

References Ascii, Char, Length, Position, Substring, Uppercase.

Max

Max has two syntaxes. The first applies to finding the maximum value of a particular field in the current selection. It only works in a footer or break at print time. The second applies to finding the maximum value of a particular subfield in a specified subselection. In both cases, the field type must be numeric.

Syntax 1 Max (*fieldname*)

Description 1 Max returns the maximum value for *fieldname* in the current selection. Max works only in In footer and In break in an output layout procedure when printing with PRINT SELECTION. To find a maximum number, 4th Dimension must work with each record in the selection. At print time, 4th Dimension brings each record into memory and finds the maximum figure.

Important

Max does not clear after a break. Rather, it calculates a record-by-record maximum value for the entire selection. Max is only meaningful when printed in break level 0.

Example 1

```
If(In footer & End selection([Income]))
   vMax:=Max([Income]Sales)
End if
```

References 1 Average, In break, In footer, Min, Squares sum, Std deviation, Sum, Variance.

Syntax 2 Max (*subfieldname*)

Description 2 Max returns the maximum value of *subfieldname* in the current subselection of the subfile.

Example 2

```
vMax:=Max([Invoice]Ordered'ItemTotal)
```

References 2 Average, Min, Squares sum, Std deviation, Sum, Variance.

MENU BAR

Syntax

MENU BAR (*posintexpr*)

Description

MENU BAR sets the current menu bar to the bar identified by *posintexpr*. The Menu editor assigns a number to each menu as you create it. This assigned number is *posintexpr*. MENU BAR also changes all menu titles to their default state (either enabled or disabled). All menu items are displayed without check marks.

Example

```
MENU BAR(2)
```

References

CHECK ITEM, DISABLE ITEM, ENABLE ITEM, Menu selected.

Menu selected

Syntax

Menu selected

Description

Menu selected returns a long integer (a four-byte integer). The low integer number contains the menu item number and the high integer number contains the menu ID number. Menu selected returns 0 if no menu item was selected. Menu selected works only in the During phase of layout and file procedures.

The Edit and Apple menus are built in and aren't a part of the menu count. The File menu is menu title number one. The example uses Menu selected to supply the menu and item aguments to CHECK ITEM.

To find the menu ID, divide Menu selected by 65,536 and convert the result to an integer. To find the menu item ID, calculate the modulo of Menu selected with the modulus 65,536.

Example

```
  `MenuCheck: Call only in During
If(During)
  If (Menu selected#0)
    CHECK ITEM(Int(Menu selected/65536 );Mod(Menu selected;65536);
              Char(18))
  End if
End if
```

References

CHECK ITEM, DISABLE ITEM, During, ENABLE ITEM.

MESSAGE

Syntax MESSAGE (*strexpr*)

Description MESSAGE displays *strexpr* in a message box until another activity modifies the screen. If you opened a window (OPEN WINDOW), the message text (in Monaco font) will appear in it without the message box. If *strexpr* is a variable, your message can be up to 255 characters.

When you display a message in a window, if your message is wider than the window, 4th Dimension automatically performs wraparound on your text. If you want to control line breaks, concatenate the carriage return character into your message. If your message has more lines than the window, 4th Dimension automatically scrolls the message. If you send successive messages to the window, each message begins at the character position following the last character of the previous message.

❖ *Positioning messages:* You can use ERASE WINDOW and GO TO XY to position messages in a window.

Example

```
MESSAGE("Now loading text file.")
```

References Alert, OPEN WINDOW.

MESSAGES OFF

Syntax MESSAGES OFF

Description MESSAGES OFF turns off the standard 4th Dimension progress messages: those showing the time it takes to perform a task like sorting or printing.

Example

```
MESSAGES  OFF
SORT  SELECTION([Addresses]ZIP;>;[Addresses]Name2;>)
MESSAGES  ON
```

Reference MESSAGES ON.

MESSAGES ON

Syntax MESSAGES ON

Description MESSAGES ON returns 4th Dimension to its default state of showing progress messages.

Example

```
MESSAGES  OFF
SORT  SELECTION([Addresses]ZIP;>;[Addresses]Name2;>)
MESSAGES  ON
```

Reference MESSAGES OFF.

Min

Min has two syntaxes. The first applies to finding the minimum value of a particular field in the current selection. It only works in In break or In footer while you're printing. The second applies to finding the minimum value of a particular subfield in a specified subselection. In both cases, the field type must be numeric.

Syntax 1 Min (*fieldname*)

Description 1 Min returns the minimum value for *fieldname* in the current selection. This works only in In footer or In break in an output layout procedure when printing with PRINT SELECTION. To find a minimum number, 4th Dimension must work with each record in the selection. At print time, 4th Dimension brings each record into memory and finds the minimum.

Important

Min does not clear after a break. Rather, it calculates a record-by-record minimum value for the entire selection. Min is meaningful only when printed in break level 0.

Example 1

```
If(In footer & End selection([Income]))
   vMin:=Min([Income]Sales)
End if
```

References 1 Average, In break, In footer, Max, Squares sum, Std deviation, Sum, Variance.

Syntax 2 Min (*subfieldname*)

Description 2 Min returns the minimum value of *subfieldname* in the current subselection of the subfile.

Example 2

```
Min([Invoice]Ordered'ItemTotal)
```

References 2 Average, Max, Squares sum, Std deviation, Sum, Variance.

Mod

Syntax Mod (*numexpr1*;*numexpr2*)

Description Mod divides *numexpr1* by *numexpr2* and returns the remainder, an integer. The Menu selected function describes how the example works.

Example

```
  `MenuCheck: Call only in During
If(During)
  If (Menu selected#0)
    CHECK ITEM(Int(Menu selected/65536 );Mod (Menu selected;65536);
                Char(18))
  End if
End if
```

References Dec, Int, Random.

Modified

Syntax Modified (*fieldname*)

Description Modified returns TRUE if the user has modified *fieldname* during data entry. A field is considered modified when a user types characters in it and leaves the field by pressing Tab or by clicking in another field, on a button, or in an area (like a scrollable or external area). Use Modified to test a change to a field when you might want to take an action, like error checking or in the case of the example, changing the case of the characters. Place Modified only in an input layout procedure.

Example

```
If(Modified([Customers]Name))
  [Customers]Name := Uppercase([Customers]Name)
End if
```

References ADD RECORD, MODIFY RECORD, REJECT.

MODIFY RECORD

Syntax MODIFY RECORD «(*) | (*filename*«;*»)»

Description MODIFY RECORD summons the default data input form for the current record of *filename*. 4th Dimension saves the record if the user clicks an Accept button. You can check this by reading the OK system variable. If the user validates the entry, OK returns 1. Otherwise, it returns 0. If the user validates a record modification without having modified any of its fields, 4th Dimension cancels the modification and the OK variable returns 0.

❖ *Saving a canceled record:* After a Don't Accept action, the record remains in memory and can be saved with SAVE RECORD.

MODIFY RECORD calls input layout procedures and executes Before, During, and After phases (as described in *4th Dimension Programmer's Reference*).

4th Dimension displays the layout in the window with scroll bars and a grow box on the window. Specifying the optional asterisk argument causes the layout to appear without scroll bars and a grow box. If you don't specify *filename,* MODIFY RECORD uses the default file. If no current record exists, MODIFY RECORD has no effect.

Example

```
DEFAULT FILE([Addresses])
INPUT LAYOUT("Input1")
gNo:=Request("Enter customer number")
SEARCH([Addresses]CustNo=Num(gNo))
If (Records in selection=0)
  CONFIRM("Record number "+gNo+" doesn't exist. Add it?")
  If (OK=1)
    ADD RECORD
  End if
Else
  MODIFY RECORD
End if
```

References ADD RECORD, MODIFY SELECTION, MODIFY SUBRECORD, SAVE RECORD.

MODIFY SELECTION

Syntax

MODIFY SELECTION «(*) | (*filename*«;*»)»

Description

MODIFY SELECTION acts much like DISPLAY SELECTION, in that it displays a list. But if the user double-clicks a record, he or she can modify it through the current input layout. If you don't specify *filename,* MODIFY SELECTION uses the default file.

If the selection contains only one record, the record appears in the input layout, if you do not include the optional asterisk. If you include the asterisk, 4th Dimension will display a one-record selection in the output layout.

If you do not include a button in your layout, 4th Dimension will supply its own. For output layouts, the Done button appears in the lower-right corner of the window. For input layouts, the 4th Dimension button panel appears. When creating an output layout, place an Accept button in the footer area of the layout.

If the user clicks a button or selects a menu item, the layout procedure for the currently displayed layout will be activated. You can then test Menu selected or the button and perform an action such as a sort or search.

The user can scroll through the selection or click on a record to select it. Clicking a different record deselects the first record and selects the second record. To select a contiguous group of records, click the first record and Shift-click the last record. To select noncontiguous records, Command-click each desired record to select.

After MODIFY SELECTION, you can find the records selected by the user by using the 4th Dimension system set, UserSet. If you want to save this set, you should first do a UNION operation between UserSet and an empty set of the same file, because at the next MODIFY SELECTION 4th Dimension erases and then recalculates UserSet. Further, UserSet, as a system set, does not belong to any file.

❖ When MODIFY SELECTION completes, the current record is "undefined." Use FIRST RECORD, LAST RECORD, or ALL RECORDS to reselect it.

Example

```
DEFAULT FILE([Accounts])
SEARCH
MODIFY SELECTION(*)
```

References

ALL RECORDS, APPLY TO SELECTION, DISPLAY SELECTION, MODIFY RECORD, Records in selection, SEARCH SELECTION.

MODIFY SUBRECORD

Syntax

MODIFY SUBRECORD (*subfilename*;*strexpr* «;*»)

Description

MODIFY SUBRECORD summons the layout specified by *strexpr* to allow the user to modify the current subrecord in *subfilename*. If no current subrecord exists, MODIFY SUBRECORD has no effect. Place MODIFY SUBRECORD in a global procedure only.

During entry, 4th Dimension executes the input layout procedure (as described in *4th Dimension Programmer's Reference*).

Clicking an Accept button keeps the modified subrecord in memory. Clicking a Don't accept button removes the modifiication from memory. To actually save the modified subrecord, you must use SAVE RECORD, because a subrecord is only saved in terms of the current record to which it belongs.

Because MODIFY SUBRECORD uses the input form, you can't take input through variables. To modifiy a subrecord through a form containing variables instead of subfields, choose DIALOG and SAVE RECORD.

If you have a subfile within a subfile, be sure to select the desired higher level subrecord before modifiying the lower level subrecord. If you don't, you could modify any subrecord in the higher level subrecords.

4th Dimension displays the layout in the window with scroll bars and a grow box on the window. Specifying the optional asterisk argument causes the layout to appear without scroll bars and a grow box.

Example

MODIFY SUBRECORD([Customer]Address;"SubInput1")

References

ADD SUBRECORD, ALL SUBRECORDS, Records in subselection, SEARCH SUBRECORDS.

Month of

Syntax

Month of (*date*)

Description

Month of returns a numeric value equal to the month found in *date*. The example would assign 8 to the variable gMo.

Example

gMo:=**Month of**(!08/14/90!)

References

Current date, Date, Day number, Day of, Year of.

NEXT RECORD

Syntax

NEXT RECORD «(*filename*)»

Description

NEXT RECORD moves the record pointer to the next record in the current selection of *filename* and loads the record into memory. If no next record exists, End selection returns TRUE, and there is no current record. If the current selection is empty or Before selection is TRUE, NEXT RECORD has no effect. Use FIRST RECORD or LAST RECORD to move the pointer back into the current selection. If you don't specify *filename,* NEXT RECORD uses the default file.

In the example, each NEXT RECORD brings up a record for display. When NEXT RECORD tries to push the record pointer past the last record, End selection goes TRUE, causing the loop to terminate.

Example

```
 `Forward: Display each record
DEFAULT FILE([Addresses])
ALL RECORDS
While (Not(End selection))
  DISPLAY RECORD
  NEXT RECORD
End while
```

References

Before selection, End selection, FIRST RECORD, LAST RECORD, PREVIOUS RECORD.

NEXT SUBRECORD

Syntax

NEXT SUBRECORD(*subfilename*)

Description

NEXT SUBRECORD moves the subrecord pointer to the next subrecord in the current subselection of *subfilename*. If no next subrecord exists, End subselection returns TRUE, and there is no current subrecord. If the current subselection is empty or Before subselection is TRUE, NEXT SUBRECORD has no effect. Use FIRST SUBRECORD or LAST SUBRECORD to move the pointer back into the curent subselection.

The example shows a piece of a procedure that writes file data, including subrecord data to a text file, using SEND PACKET.

Example

```
.
.
.
While (Not(End subselection([I.Card]Phone)))  `Write subfile items
  SEND PACKET([I.Card]Phone'PhNum+Char(13))
  SEND PACKET([I.Card]Phone'Comment+Char(13))
  NEXT SUBRECORD([I.Card]Phone)
End while
```

References

Before subselection, End subselection, FIRST SUBRECORD, LAST SUBRECORD, PREVIOUS SUBRECORD.

Not

Syntax Not (*boolexpr*)

Description The Not function returns the negation of *boolexpr,* changing a TRUE to FALSE or a FALSE to TRUE. In the example, it keeps the While loop going until NEXT RECORD tries to push the record pointer past the last record.

Example

```
 `Forward: Display each record
DEFAULT FILE([Addresses])
ALL RECORDS
FIRST RECORD
While (Not(End selection))
  DISPLAY RECORD
  NEXT RECORD
End while
```

References False, True.

NO TRACE

Syntax NO TRACE

Description NO TRACE turns off the debugger, engaged by TRACE, by an error, or by the user. Using NO TRACE has the same effect as clicking the NO TRACE button in the debugger.

Example

```
NO TRACE
```

Reference TRACE.

Num

The Num function has two syntaxes. The first transforms a numeric string into a true numeric value. The second uses a Boolean expression to return 0 or 1.

Syntax 1

Num (*strexpr*)

Description 1

Num converts a numeric string (*strexpr*) into a numeric value. If the string consists only of one or more alphabetic characters Num returns a zero. If the string includes alphabetic characters mixed in with numeric characters, 4th Dimension ignores the alphabetic characters. Thus, Num transforms the string a1b2c3 into the number 123.

Example 1

```
x:=Num("123XYZ45")   `x gets 12345
```

Reference 1

String.

Syntax 2

Num (*boolexpr*)

Description 2

When you give a *boolexpr* argument to Num, Num returns 0 when a condition is FALSE and 1 when it is TRUE. In the example, Num of the customer debits returns either 0 or 1. The customer comment is then printed either once or not at all, because the asterisk (*) is used as a string repetition operator.

Example 2

```
  `If client owes less than 1000, a good risk.
  `If client owes more than 1000, a bad risk.
[Client]Risk:=("Good"*Num([Client]Debt<1000))+("Bad" *Num([Client]Debt>=1000))
```

References 2

False, Not, True.

Old

Syntax Old (*fieldname*)

Description Old returns the value of *fieldname* for the current record before *fieldname* was modified. In other words, the value of the record on disk. When 4th Dimension loads a record, it loads two copies. When you modify a record either through MODIFY RECORD or by modifying directly in the User environment, 4th Dimension changes the second copy while storing the first copy in its original form until the record is saved to disk or another record is loaded. Old returns field values of the stored original. Old may be applied to all field types, except fields of type subfile.

Important
Adding or creating a record is not modification. Modification takes place only through the MODIFY RECORD command and through the User environment Modify Record menu item.

Example

```
[Catalog]OnHand := [Catalog]OnHand + Old(Quantity)
```

References LOAD OLD LINKED RECORD, MODIFY RECORD, SAVE OLD LINKED RECORD.

ONE RECORD SELECT

Syntax ONE RECORD SELECT «(*filename*)»

Description ONE RECORD SELECT reduces the current selection to the current record. You must use ONE RECORD SELECT after pulling a record from the record stack with POP RECORD, because a popped record is not a part of the current selection. If no current record exists, ONE RECORD SELECT has no effect.

Example

```
DEFAULT FILE ([MyFile])
POP RECORD
ONE RECORD SELECT
```

References POP RECORD, PUSH RECORD.

ON ERR CALL

Syntax

ON ERR CALL (*strexpr*)

Description

ON ERR CALL installs the procedure named by *strexpr* as the interrupt procedure for managing errors. Giving a null string argument passes error handling back to 4th Dimension. After installation, 4th Dimension automatically calls the procedure named by *strexpr* when an error occurs.

Normally, place one ON ERR CALL at the beginning of a procedure to identify where the procedure should go if an error occurs during execution and a second ON ERR CALL with a null string argument at the end of the procedure

You can identify errors by reading the ERROR system variable and present your own error messages. Choose the ABORT procedure to terminate an installed error procedure. In the example, the called routine, Oops, concludes with an ABORT.

If you don't call ABORT in the installed procedure, 4th Dimension returns to the interrupted procedure.

❖ *4th Dimension error handling:* If you have installed a procedure with ON ERR CALL, 4th Dimension will not present its error window until you call ON ERR CALL with a null string. There is one exception. If your installed procedure causes an error, 4th Dimension will automatically take over error handling. The reason is that the installed procedure is not reentrant.

Example

```
ON ERR CALL("Oops")
kaboom   `The error: looks like a var
ON ERR CALL("")   `Pass err handling to 4th Dimension


  `Oops; called by ON ERR CALL
  `Error handling routine
CONFIRM("Error "+String(Error)+" Do you want to stop?")
If (OK=1)
  ABORT   `End procedure, return to menus
End if
```

References

ABORT, ON EVENT CALL, ON SERIAL PORT CALL.

ON EVENT CALL

Syntax

ON EVENT CALL (*strexpr*)

Description

ON EVENT CALL installs the procedure named by *strexpr* as the interrupt procedure for managing events. Giving a null string argument passes event handling back to 4th Dimension. After installation, 4th Dimension automatically calls the procedure named by *strexpr* when an event occurs: either a mouse click or a keystroke. A procedure must be executing for the event to be recognized and the installed procedure executed. This means that between execution phases of a layout procedure, the installed procedure will not be called. Rather 4th Dimension will handle events. In the procedure, you can read three system variables—MouseDown, KeyCode, and Modifiers. MouseDown returns 1 if the event is the user clicking the mouse button and 0 if not. KeyCode returns the ASCII code for a keystroke. Appendix E describes Modifiers and its use in writing external routines.

Important

MouseDown, KeyCode, and Modifiers contain significant values only within a routine called by an ON EVENT CALL procedure.

You normally write two ON EVENT CALL statements. Place the first one at the beginning of a procedure to identify what procedure to call if an event occurs during execution. Place the second ON EVENT CALL with a null string argument at the end of the calling procedure to turn off event handling. 4th Dimension returns to the interrupted procedure at the end of the installed procedure.

In the example, the TestEvent program calls the EventCity routine to trap events. Note that TestEvent initializes the system variable OK at the beginning of the program. Without this, the program won't work. Also, the program clears its ON EVENT CALL reference. Without clearing, the installed procedure will continue to be activated by events.

Example

```
  `TestEvent
OK:=1 `Init OK system variable
ON EVENT CALL("EventCity")
  `Give program something to do: write numbers
i:=1
While ((i<1000)&(OK=1))
  MESSAGE(String(i)+Char(13))
  i:=i+1
End while
ON EVENT CALL("")    `Turn off event call


  `EventCity What to do in case of an event...
  `Test the mouse
If (MouseDown=1)
  CONFIRM("You clicked the mouse. Click cancel to quit.")
  `Clicking cancel sets OK to 0.
End if
  `Test the keyboard
If (KeyCode#0)
  CONFIRM("You pressed the "+Char(KeyCode)+" key.")
  `Clicking cancel sets OK to 0.
End if
```

References Ascii, Char, ON ERR CALL, ON SERIAL PORT CALL.

ON SERIAL PORT CALL

Syntax ON SERIAL PORT CALL (*strexpr*)

Description ON SERIAL PORT CALL installs the procedure named by *strexpr* as the interrupt procedure for managing serial port events. The procedure named by *strexpr* is automatically called by 4th Dimension when one character enters the serial port buffer. Giving a null string turns off serial port event handling.

4th Dimension suspends the operation you're working on when port activity occurs, and does not return to it until it has executed the installed procedure. A procedure must be executing for the serial port event to be recognized and the installed procedure executed.

4th Dimension automatically calls the installed procedure when the serial port buffer contains one or more characters. If you decide to do nothing with the buffer contents, don't forget to clear the buffer contents by calling RECEIVE BUFFER. If you don't, 4th Dimension will call your installed procedure again.

Example

```
ON SERIAL PORT CALL ("DoBuffer")
```

References ON ERR CALL, ON EVENT CALL, RECEIVE BUFFER, RECEIVE PACKET, RECEIVE RECORD.

OPEN WINDOW

Syntax OPEN WINDOW (*posexpr1;posexpr2;posexpr3;posexpr4«;posexpr5«;strexpr»»*)

Description OPEN WINDOW draws a window with dimensions given by the first four *posexpr:*

- □ *posexpr1* is the distance in pixels from the left edge of the screen to the left edge of the window.
- □ *posexpr2* is the distance in pixels from the top of the screen to the top edge of the window. The top of the menu bar is the top pixel.
- □ *posexpr3* is the distance in pixels from the left edge of the screen to the right edge of the window.
- □ *posexpr4* is the distance in pixels from the top of the screen to the bottom edge of the window.

posexpr5 is optional. It represents the type of window you want to draw and corresponds to the six standard windows available through the Macintosh window manager (see Table II-4).

Table II-4
Six standard Macintosh windows

Window name	Number	Description
documentProc	0	Standard document window
dBoxProc	1	Alert box or modal dialog box
plainDBox	2	Plain box
altDBox	3	Plain box with drop shadow
noGrowDocProc	4	Document window without size box
	8	Document window with zoom box
rDocProc	16	Rounded-corner window

The sixth argument, *strexpr,* is the window's title.

If you omit the two last arguments, 4th Dimension automatically draws the dBoxProc window. Things you create, like dialog boxes and input layouts, will automatically appear inside a created window. In the example, text that would normally appear in a MESSAGE box appears in the window and will scroll when it reaches the bottom edge of the window, making the window behave like a terminal. Message text always appears in the Monaco font, a font in which all characters are the same size. When a window is open and 4th Dimension encounters commands like ADD RECORD and DISPLAY SELECTION, the layout will appear in the window.

❖ *One window:* 4th Dimension allows only one custom window at a time over the 4th Dimension window. Therefore, OPEN WINDOW has no effect if another custom window is already on the screen.

To make your windows independent of display size, you can use Screen height and Screen width to calculate the upper left and lower right corners of the window. The example does this.

Example

```
  `Created a centered window on the screen.
OK:=1
While (OK=1)
  Box height:=Num(Request("Height (click cancel if done)?"))
  If (OK=1)
    Box width:=Num(Request("Width?"))
    CLOSE WINDOW `Close any open window
    H1:=Screen height/2+10 `Add 10 to allow for the menubar
    W1:=Screen width/2
    H2:=Box height/2
    W2:=Box width/2
    OPEN WINDOW(W1–W2;H1–H2;W1+W2;H1+H2)
  End if
End while
```

References

ADD RECORD, CLOSE WINDOW, DIALOG, DISPLAY SELECTION, ERASE WINDOW, GO TO XY, MESSAGE, MODIFY RECORD, MODIFY SELECTION, Screen height, Screen width, SET WINDOW TITLE.

OUTPUT LAYOUT

Syntax

OUTPUT LAYOUT (*«filename;» strexpr*)

Description

OUTPUT LAYOUT specifies which layout (named by *strexpr*) 4th Dimension should use to output data from *filename*. Whether printing to the screen or a printer, or exporting data, 4th Dimension automatically uses the specified layout. The layout must belong to *filename*. If you don't specify *filename,* OUTPUT LAYOUT applies to the default file.

When exporting data, the output layout serves as an organizer and channel between the 4th Dimension database file and a document or serial port. If you do not specify an output layout for activities like EXPORT TEXT, PRINT SELECTION, and screen display, 4th Dimension uses the current output layout.

OUTPUT LAYOUT leaves the specified layout as the current output layout until you respecify it. Don't use OUTPUT LAYOUT within an output layout.

Example

```
  `Mail List db, Addresses file
DEFAULT FILE([Addresses])
ALL RECORDS
OUTPUT LAYOUT("OneWide")
PRINT LABEL
OUTPUT LAYOUT("Output1")    `restore default
```

References

DISPLAY SELECTION, EXPORT DIF, EXPORT SYLK, EXPORT TEXT, INPUT LAYOUT, MODIFY SELECTION, PRINT LABEL, PRINT SELECTION.

POP RECORD

Syntax POP RECORD «(*filename*)»

Description POP RECORD pops a record (and its subrecords, if any) belonging to *filename* from the *filename* record stack and makes it the current record. However, it may not be a part of the current selection. Therefore, it's up to you to call ONE RECORD SELECT after you pop a record to reduce the current selection to the current (popped) record.

If you push a record, change the selection so as not to include the pushed record, and then pop the record, the current record will not be in the current selection. If you want to designate the popped record as the current selection, use ONE RECORD SELECT. If you use any commands that move the record pointer before saving the record, you will lose the copy in memory.

You place records on the stack with PUSH RECORD. Each file has its own record stack. Stack capacity is limited by memory. POP RECORD is useful when working with recursive procedures on records and recursive links.

4th Dimension clears the stack of any unpopped records when you return to the menu at the end of the execution of your procedure.

Example **POP RECORD** ([Customers])

References ONE RECORD SELECT, PUSH RECORD.

Position

Syntax Position (*strexpr1*;*strexpr2*)

Description Position returns the position of the first occurrence of *strexpr1* in *strexpr2*. If Position fails to find the string, it returns a zero. If Position finds an occurence, it returns the postion of the first character of *strexpr1*. If you ask for the position of a null string within a null string, Position returns a 1.

Example Where:=**Position**("ta";"database")`Where is assigned 3.

References Ascii, Char, GET HIGHLIGHTED TEXT, HIGHLIGHT TEXT, Length, Lowercase, Num, String, Substring, Uppercase.

PREVIOUS RECORD

Syntax

PREVIOUS RECORD «(*filename*)»

Description

PREVIOUS RECORD moves the record pointer to the previous record in the current selection of *filename,* makes it the current record, and loads it into memory. If you don't specify *filename,* PREVIOUS RECORD uses the default file. If no previous record exists, Before selection returns TRUE, and there is no current record. If the current selection is empty, PREVIOUS RECORD has no effect.

Use FIRST RECORD or LAST RECORD to move the pointer back into the current selection. The example sets the record pointer on the last record and then moves the pointer toward the beginning of the selection, displaying each record as it goes.

Example

```
  `Backward: Show Last Record & Before selection
DEFAULT FILE([Addresses])
ALL RECORDS
LAST RECORD
While (Not(Before selection))
  DISPLAY RECORD
  PREVIOUS RECORD
End while
```

References

Before selection, End selection, FIRST RECORD, LAST RECORD, NEXT RECORD.

PREVIOUS SUBRECORD

Syntax PREVIOUS SUBRECORD (*subfilename*)

Description PREVIOUS SUBRECORD moves the subrecord pointer to the previous subrecord in the current subselection of *subfilename* and makes it the current subrecord. If no previous subrecord exists, Before subselection returns TRUE, and there is no current subrecord. If the current subselection is empty or End subselection is TRUE, PREVIOUS SUBRECORD has no effect. Use FIRST SUBRECORD or LAST SUBRECORD to move the pointer back into the current subselection.

Example

```
.
.
.
  LAST SUBRECORD([Stats]Sales)
  While (Not(Before subselection([Stats]Sales)))
    vReport:=vReport+String([Stats]Sales'Bucks)+Char(13)
    PREVIOUS SUBRECORD([Stats]Sales)
  End while
```

References Before subselection, End subselection, FIRST SUBRECORD, LAST SUBRECORD, NEXT SUBRECORD.

PRINT LABEL

Syntax PRINT LABEL «(*filename* «;*»)»

Description PRINT LABEL prints labels through the current output layout of *filename* for the current selection of *filename*. If you don't specify *filename*, PRINT LABEL uses the default file. You cannot print subfiles in a label. To print addresses from subfiles, choose PRINT SELECTION. If you follow *filename* with the optional asterisk argument, 4th Dimension suppresses the printer dialog boxes.

In the layout, you can use fields and/or variables. To prepare for printing labels, do these things:

1. Set the header line of the layout to zero.
2. Measure the distance between the top of one label and the top of the next label.
3. Set the break and the footer lines to the measured distance.

4a. For a single column of labels, move the label width line to the page width line (the right margin).

4b. For multi-columns of labels, measure the distance between the left edge of a label and the left edge of the next label. Set the label width line to that distance (from the left margin).

5. Create a label variable or place fields so that they will fit in the label area.

❖ *For details:* See *4th Dimension User's Guide* for details on creating labels.

To eliminate blank lines in a label, you can concatenate field data into a label variable. The procedure in the example does this, suppressing any blank third line that might appear. See "Printing Labels From Subfiles" in Appendix A.

Example

```
   `LabelOne
   `Mail List db, Addresses file
gCR:=Char(13)
DEFAULT FILE([Addresses])
ALL RECORDS
OUTPUT LAYOUT("L.Address")
PRINT LABEL
OUTPUT LAYOUT("Output1")    `restore default


   `L.Address layout procedure for printing labels
If (Before)
   vLabel:=Name1+" "+Name2+gCR+Addr1+gCR
   `If line not blank, concatenate
   If (Addr2#"")
      vLabel:=vLabel+Addr2+gCR
   End if
   vLabel:=vLabel+City+", "+St+" "+ZipCode
End if
```

References OUTPUT LAYOUT, PRINT LAYOUT, PRINT SELECTION, PRINT SETTINGS.

PRINT LAYOUT

Syntax

PRINT LAYOUT (*«filename;»strexpr*)

Description

PRINT LAYOUT prints the layout specified by *strexpr* that belongs to *filename* with the current values of fields and variables. If you omit *filename,* this command uses the default file. PRINT LAYOUT executes only the instructions found in the Before and During portion of the layout procedure.

PRINT LAYOUT doesn't issue a form feed after printing the layout. Therefore, you can print more than one layout on a page. In fact, you can print records from different files on the same page. This makes PRINT LAYOUT perfect for complex printing tasks, involving different files and different layouts.

You can force a form feed between layouts with the FORM FEED command. If you want the the printer dialog boxes to appear, you must include the PRINT SETTINGS command before any PRINT LAYOUT command.

❖ *Differences:* PRINT LAYOUT is different from the other 4th Dimension printing commands, in that it begins to fill a page in memory. Printing cannot begin until the page in memory is full. Therefore, to ensure the printing of the last page after any use of PRINT LAYOUT, you must conclude with the FORM FEED command to force the printing. Otherwise, the last page will stay in memory and not be printed.

Keep these three things in mind:

1. PRINT LAYOUT works only with record-level, not subfile-level layouts.
2. PRINT LAYOUT only prints between the header and detail lines.
3. You need to plan for your form feeds.

Example

```
PRINT LAYOUT([Customers];"Credit Report")
```

References

FORM FEED, PRINT LABEL, PRINT SELECTION, PRINT SETTINGS, REPORT.

PRINT SELECTION

Syntax

PRINT SELECTION «(*)|(*filename*«;*»)»

Description

PRINT SELECTION prints all records in the current selection of *filename* through the current output layout. If you follow *filename* with the optional asterisk argument, 4th Dimension suppresses the printer dialog boxes and uses the settings you defined when creating the layout. If you don't specify *filename,* PRINT SELECTION uses the default file. During printing, 4th Dimension executes the output layout procedure:the Header phase when printing a header, Before and During phase when printing each record, Break phase when printing the Break area and Footer phase when printing a footer.

To print a sorted selection with subtotals, using PRINT SELECTION, you must first sort the selection (with SORT SELECTION) and include the Subtotal function in the layout procedure. In this case, you can also use the Level function in the Break phase to detect which level of break 4th Dimension is currently printing.

You can check if 4th Dimension is printing the first header by testing Before selection in the Header phase. You can also check for the last footer by testing End selection in the Footer phase.

When you print a selection with PRINT SELECTION, you can also use statistical and arithmetic functions like Sum and Average in the Break phase or the Footer phase to assign values to variables in the Footer and/or Break area.

You can number pages either in the Header or Footer phases. You can also force form feeds during printing by placing FORM FEED in the layout procedure.

After PRINT SELECTION, you can see if the printing is completed by testing the OK system variable. When completed, OK returns 1. If the user has interrupted the printing process, OK returns 0.

You can print labels from subrecords with PRINT SELECTION. To do this, follow these steps:

1. Create a layout for the subfile.
2. Create a layout for the file.
3. Include a subfile area in the file layout, and choose the Fixed Frame (Multiple records) option.
4. To print, use PRINT SELECTION or the Print command in the File menu in User environment.

See *4th Dimension User's Guide* for details on subfile layouts. This method prints labels one column wide only.

Example

```
DEFAULT FILE([Accounts])
ALL RECORDS
DISPLAY SELECTION
USE SET("UserSet")
PRINT SELECTION
```

References Before, Before selection, During, End selection, In break, In footer, In header, PRINT LABEL, PRINT LAYOUT, PRINT SETTINGS, REPORT.

PRINT SETTINGS

Syntax PRINT SETTINGS

Description PRINT SETTINGS displays the two standard printer dialog boxes on screen. If the user clicks OK on both OK buttons or presses Enter, the system variable OK is set to 1. Otherwise, OK is set to 0. You should include this before any group of the PRINT LAYOUT commands.

Example

```
PRINT SETTINGS
```

References PRINT LABEL, PRINT LAYOUT, PRINT SELECTION, REPORT.

PUSH RECORD

Syntax

PUSH RECORD «(*filename*)»

Description

PUSH RECORD pushes the current record (and its subrecords, if any) onto the record stack. Each file has its own record stack. Stack capacity is limited by memory. PUSH RECORD is useful when working with recursive procedures on records and recursive links. You pop records off the stack with POP RECORD.

POP RECORD and PUSH RECORD are also useful when you want to examine records in the same file during data entry. To do this, you push a record (the current entry) on the stack. Then, you search and examine records in the file (copy values in variables, for example). Finally, you pop the record to restore the current entry.

4th Dimension clears the stack of any unpopped records when you return to the menu at the end of the execution of your procedure.

Example

PUSH RECORD ([Customers])

References

ADD RECORD, CREATE RECORD, MODIFY RECORD, ONE RECORD SELECT, POP RECORD.

Random

Syntax

Random

Description

Random returns a random integer value between 0 and 32,767 (inclusive). To define a range of integers, you can use the formula:

Mod(Random;(End-Start+1))+Start

Start is the first number in the range and End the last. The example would assign a random integer between 10 and 30:

Example

v := **Mod**(**Random**;21)+10

References

Dec, Int, Mod.

READ ONLY

Syntax READ ONLY «(*filename*)»

Description READ ONLY is useful only in multi-user applications. It assigns a file read-only status to all records in *filename*. See *4th Dimension Utilities and Developer's Notes* for details on this and other multi-user commands.

READ WRITE

Syntax READ WRITE «(*filename*)»

Description READ WRITE is useful only in multi-user applications. It assigns a file read-write status to all records in *filename*. See *4th Dimension Utilities and Developer's Notes* for details on this and other multi-user commands.

RECEIVE BUFFER

Syntax RECEIVE BUFFER (*strvar*)

Description RECEIVE BUFFER takes the contents of the serial port buffer assigned by SET CHANNEL and assigns it to *strvar* and clears the buffer. You can work with RECEIVE BUFFER without having to stop execution while you wait for data to enter the buffer. RECEIVE BUFFER is useful in a procedure installed by ON SERIAL PORT CALL.

RECEIVE BUFFER takes whatever is in the buffer. RECEIVE PACKET, on the other hand, waits until either it finds a specific character or until a certain number of characters are in the buffer. If RECEIVE PACKET cannot meet its goal, it waits until the specified character or the correct number of characters arrive in the buffer. RECEIVE BUFFER can only take characters from a serial port, whereas RECEIVE PACKET can take characters from a serial port and from documents.

Example **RECEIVE BUFFER**(CatchMe)

References ON SERIAL PORT CALL, RECEIVE PACKET, RECEIVE RECORD, RECEIVE VARIABLE, SEND PACKET, SET CHANNEL.

RECEIVE PACKET

RECEIVE PACKET has two syntaxes. The first syntax specifies the number of characters you want to receive. The second specifies a character at which 4th Dimension should stop transferring characters. You must specify a serial port or a document with SET CHANNEL before giving this command. You can use SEND PACKET and RECEIVE PACKET to write and read data from desktop documents.

If you receive characters from the serial port, the sending device can be a Macintosh running 4th Dimension, sending characters with SEND PACKET. It can also be any other computer that can send ASCII characters or any RS-232 device. If you receive characters from a Macintosh document, 4th Dimension reads them from the document.

To read an ASCII document with 4th Dimension, you call SET CHANNEL to open the document. If you use RECEIVE PACKET, 4th Dimension begins reading at the beginning of the document, using the specified syntax. The reading of each subsequent packet begins at the character following the last character read.

The user can interrupt a RECEIVE PACKET operation by pressing Option-Space. You can check this by testing the OK system variable. If the packet is successfully received, OK returns 1. Otherwise, OK returns 0. OK is set to 0 if the user cancels with Option-Space or if an error occurs. Because attempting to read past the end of file marker is an error, you can test for end of file status by testing OK equal to zero.

SEND PACKET is the counterpart of RECEIVE PACKET. You must have set a channel to a document or a serial port with SET CHANNEL before using either.

SEND PACKET and RECEIVE PACKET are commonly used to write data back and forth between 4th Dimension subfiles and desktop documents.

Syntax 1 RECEIVE PACKET (*strvar;posintexpr*)

Description 1 RECEIVE PACKET transfers the number of characters specified by *posintexpr* into *strvar*. This syntax works only in a global procedure. The example would transfer 20 characters from the serial port or from a document into the variable GetTwenty.

Example 1

```
RECEIVE PACKET(GetTwenty;20)
```

References 1 ON SERIAL PORT CALL, RECEIVE BUFFER, SEND PACKET, SET CHANNEL, USE ASCII MAP.

Syntax 2 RECEIVE PACKET (*strvar*;*strexpr*)

Description 2 RECEIVE PACKET transfers characters into *strvar* until it encounters the first character of *strexpr*. With this syntax, when you work with a serial port, 4th Dimension doesn't place the *strexpr* delimiter in *strvar*. When you work with a document, 4th Dimension also doesn't place the delimiter in *strvar*. When 4th Dimension reads the next packet, it skips the delimiter and begins with the character following the delimiter. The example uses a carriage return as the delimiter.

Example 2

```
RECEIVE PACKET(gField;Char(13))
```

References 2 ON SERIAL PORT CALL, RECEIVE BUFFER, SEND PACKET, SET CHANNEL, USE ASCII MAP.

RECEIVE RECORD

Syntax

RECEIVE RECORD «(*filename*)»

Description

RECEIVE RECORD receives all the values of a record through the serial port or from a document opened by SET CHANNEL and places them in the current record of *filename*. If you don't specify *filename*, RECEIVE RECORD uses the default file. If you receive the values into a record that already has values in it, you delete the values and replace them with the received values. If you want to create a new record to receive values, you must write a CREATE RECORD statement before using RECEIVE RECORD. In either case, you must call SAVE RECORD, if you want to save the modified or new record.

Important

When sending and receiving records between databases, the file that sends the record and the file that receives the record must have the same number of fields, type of fields, and order of fields.

If you receive the record from a serial port, the sending device must be a Macintosh running 4th Dimension, which sends the record with SEND RECORD. If you receive the record from a document, the record is read from the document. In either case, 4th Dimension receives the full record (along with any subrecords it might contain) in an internal data format that can only be correctly created with SEND RECORD. During the RECEIVE RECORD operation, the user can interrupt by pressing Option-Space. To check for interrupts, you can test the OK system variable. OK returns 1 if the record was received. Otherwise, OK returns 0.

Example

```
SET CHANNEL(10;"MyFile.Text")
OK:=1
While(OK=1)   `OK=0 when no more records to read
  CREATE RECORD([MyFile])
  RECEIVE RECORD([MyFile])
  SAVE RECORD([MyFile])
End while
SET CHANNEL(11)
```

References

RECEIVE PACKET, RECEIVE VARIABLE, SEND RECORD, SET CHANNEL.

RECEIVE VARIABLE

Syntax RECEIVE VARIABLE (*var*)

Description RECEIVE VARIABLE receives a variable (*var*) through the serial port or from a document opened by SET CHANNEL. If you receive the variable from a serial port, the sending device must be a Macintosh running 4th Dimension, which sends the variable with SEND VARIABLE. If you receive the variable from a document, the variable is read from the document. In either case, 4th Dimension receives the variable in an internal data format that can only be correctly created with SEND VARIABLE. During the RECEIVE VARIABLE operation, the user can interrupt by pressing Option-Space. To check for interrupts, you can test the OK system variable. OK returns 1 if the variable was received. Otherwise, OK returns 0.

Example

```
RECEIVE VARIABLE(AcctNo)
```

References CLEAR VARIABLE, LOAD VARIABLE, RECEIVE PACKET, RECEIVE RECORD, SAVE VARIABLE, SEND VARIABLE, SET CHANNEL.

Records in file

Syntax Records in file «(*filename*)»

Description Records in file returns the total number of records in *filename*. If you don't specify *filename,* Records in file uses the default file.

❖ *Comparison:* Records in selection returns the number of records in the current selection only.

Example

```
TotRecs:=Records in file([Customers])
```

References ALL RECORDS, Records in selection, Records in set, Records in subselection.

Records in selection

Syntax

Records in selection «(*filename*)»

Description

Records in selection returns the number of records in the current selection of *filename*. If you don't specify *filename,* Records in selection uses the default file. When working with subrecords, use Records in subselection.

❖ *Comparison:* Records in file returns the total number of records in the file.

Example

```
DEFAULT FILE ([ADDRESSES])
SEARCH([ADDRESSES]State="NY")
ALERT("You have "+String(Records in selection)+" customers in New York")
```

References

ALL RECORDS, Records in file, Records in set, Records in subselection, SEARCH, SEARCH BY INDEX, SEARCH SELECTION.

Records in set

Syntax

Records in set (*strexpr*)

Description

Records in set returns the number of records in the set named by *strexpr*.

Example

```
DEFAULT FILE([Addresses])
DISPLAY SELECTION
ALERT("You selected "+String(Records in set("UserSet"))+" records.")
```

References

ADD TO SET, CREATE EMPTY SET, CREATE SET, DIFFERENCE, INTERSECTION, LOAD SET, UNION, USE SET.

Records in subselection

Syntax

Records in subselection (*subfilename*)

Description

Records in subselection returns the number of subrecords in the current subselection of *subfilename*. Records in subselection applies only to subrecords in the current record. It is the subrecord equivalent of Records in selection.

Example

```
SEARCH([Inventory];[Inventory]Part No="A531")
SEARCH SUBRECORDS([Inventory]Sales;Year of([Inventory]Sales'Date)=1990)
ALERT ("Sales during 1990. "+String(Records in subselection([Inventory]Sales)))
```

References

ALL SUBRECORDS, Records in selection, SEARCH SUBRECORDS.

REDRAW

REDRAW has three syntaxes: one for subfile (Syntax 1), one for variable lists (Syntax 2), and one for files displayed in an included layout (Syntax 3). REDRAW is useful only when you want to force a screen display update.

Syntax 1 REDRAW (*subfilename*)

Description 1 During an entry, if you change the current subselection, 4th Dimension automatically redraws the subfile if the subfile is present in the layout. You must call REDRAW for a subfile only when you change the order of subrecords with SORT SUBSELECTION.

Example 1 **REDRAW**([Invoice]Detail)

Reference 1 SORT SUBSELECTION.

Syntax 2 REDRAW (*var*)

Description 2 You must call REDRAW for a variable list if you change the contents without using the assignment operator. For example, when you get a variable with LOAD VARIABLE. For a scrollable area, 4th Dimension automatically redraws the area when you add a value to or delete a value from the list. You must call REDRAW for a scrollable area only when you modify a value in the list.

Example 2 **REDRAW**(vItemList)

Reference 2 None.

Syntax 3 REDRAW (*filename*)

Description 3 You must call REDRAW for an included file when you change anything in the file. This includes changing the selection, the order of the records, or a value in a record.

Example 3 **REDRAW**([Invoice])

References 3 SEARCH, SEARCH BY INDEX, SEARCH SELECTION, SORT BY INDEX, SORT SELECTION.

REJECT

REJECT has two syntaxes: one to reject a field entry and one to reject a record entry.

Syntax 1

REJECT (*fieldname*)

Description 1

Use REJECT to force the user to reenter an incorrect value. REJECT only has an effect if you call it during the During phase of an input layout procedure. REJECT in the After phase is too late and has no effect. REJECT *subfieldname* within a parent layout has no effect. If *fieldname* has just been modified during data entry, then REJECT leaves the cursor in *fieldname* on the layout. Use HIGHLIGHT TEXT to select *fieldname*. REJECT can be used in input layout procedures only.

Example 1

```
If (During)
  If (Modified(Salary)&Not(bCancel))
      `Field changed and record not cancelled
    If(Salary<10000)
      ALERT ("Salary cannot exceed $10,000")
      REJECT(Salary)
    End if
  End if
End if
```

References 1

ADD RECORD, ADD SUBRECORD, During, GO TO FIELD, HIGHLIGHT TEXT, Modified, MODIFY RECORD, MODIFY SUBRECORD.

Syntax 2

REJECT

Description 2

REJECT rejects the entire entry. If the user tries to validate the entry, REJECT cancels the validation and continues data entry. The user must either correct a field or cancel. REJECT only has an effect if you call it during the During phase of an input layout procedure. The example comes from a doctor's patient file:

Example 2

```
If(During)
  If((Gender="Male") & (Pregnant="Yes"))
    ALERT("Are you sure????")
    REJECT
    GO TO FIELD (Pregnant)
  End if
End if
```

References 2

ADD RECORD, ADD SUBRECORD, During, GO TO FIELD, Modified, MODIFY RECORD, MODIFY SUBRECORD.

REPORT

Syntax REPORT (*«filename;»strexpr*)

Description REPORT prints the report specified by *strexpr* for the current selection of *filename*. If you specify a null string for *strexpr*, 4th Dimension automatically brings up the standard file dialog box, and the user can choose the report. In this case, if the user clicks the Cancel button of the dialog, 4th Dimension brings up the standard Quick report window as in the User environment. In this case, the user can create or modify any report for *filename*. If you omit *filename*, REPORT uses the default file.

Example

```
If(DoReport)
  REPORT([Invoices] ; "")
End if
```

References PRINT LABEL, PRINT LAYOUT, PRINT SELECTION, PRINT SETTINGS.

Request

Syntax

Request (*strexpr1* «;*strexpr2*»)

Description

Request creates a dialog box with a prompt, *strexpr1,* a text input area with an optional default value specified by *strexpr2,* and two buttons (OK and Cancel). The OK button is the default button. The user can either click it or press Enter to enter a response. The function returns the string entered by the user. If the user clicks OK, the OK system variable returns 1. Otherwise OK returns 0.

Request and CONFIRM are two different ways of handling the user interface. With CONFIRM, you ask the user to confirm an action. With Request, you ask the user for a value. If you ask the user for a numeric or a date value, be sure to convert the string value returned by Request to the proper type, with the Num or Date functions.

4th Dimension truncates any prompt message that is too long for its display area.

❖ *By the way:* If you need to get several pieces of information from a user, design a dialog box, rather than presenting a succession of Request dialogs.

Example

```
DEFAULT FILE([Invoice History])
ALL RECORDS
vDate:=Request("Enter invoice date:"; String(Current date))
If (OK = 1)
  SEARCH BY INDEX([InvoiceHistory]InvoiceDate=Date(vDate))
  DISPLAY SELECTION
End if
```

References

Alert, Confirm, Date, DIALOG, Num, String.

Round

Syntax

Round (*numexpr*;*intexpr*)

Description

Round returns *numexpr* rounded to the number of decimal places given by *intexpr*. If *intexpr* is positive, the number is rounded to the *intexpr* decimal place. If *intexpr* is negative, the number is rounded to the *intexpr*+1 digit to the left of the decimal point.

If the digit following *intexpr* is 5 though 9, Round rounds toward positive infinity for a positive number and negative infinity for a negative number. If the digit following *intexpr* is 0 through 4, Round rounds toward zero.

Examples

Tot:=**Round**(16.857;2)	`Tot gets 16.86
RoundOff:=**Round**(32345.67;-3)	`RoundOff gets 32000
s:=**Round**(29.8725;3)	`s gets 29.872
vTot:=**Round**(–1.5;0)	`vTot gets –2

References

Abs, Dec, Int, Mod, Random, Trunc.

SAVE LINKED RECORD

Syntax

SAVE LINKED RECORD (*fieldname*)

Description

SAVE LINKED RECORD saves the record pointed to by *fieldname*. You must write a SAVE LINKED RECORD statement to save any record created with CREATE LINKED RECORD or when you want to save modifications to a record loaded with LOAD LINKED RECORD. SAVE LINKED RECORD does not apply to subfiles because saving the parent record automatically saves the subrecords. For other discussions on links, see *4th Dimension User's Guide* and *4th Dimension Programmer's Reference*.

Example

```
Case of
  :(During)
    LOAD LINKED RECORD(PartNo)
   .
   .
   .
  :(After)
    SAVE LINKED RECORD(PartNo)
End case
```

References

ACTIVATE LINK, CREATE LINKED RECORD, LOAD LINKED RECORD, LOAD OLD LINKED RECORD, Old, SAVE OLD LINKED RECORD.

SAVE OLD LINKED RECORD

Syntax

SAVE OLD LINKED RECORD (*fieldname*)

Description

SAVE OLD LINKED RECORD operates the same as SAVE LINKED RECORD, but uses the old link to the *fieldname,* to save the old linked record. You must have previously loaded the record with LOAD OLD LINKED RECORD. Use SAVE OLD LINKED RECORD when you want to save modifications to a record loaded with LOAD OLD LINKED RECORD. For other discussions on links, see *4th Dimension User's Guide* and *4th Dimension Programmer's Reference.*

Example

```
Case of
.
.
.
    :(After)
          `After validation, save the new data in linked record
          `Subtract order quantity from catalog
      [Catalog]OnHand:=[Catalog]OnHand-Quantity
      SAVE LINKED RECORD(PartNo)
          `Be sure to update the old linked record,
          `Re-instate old Quantity
      LOAD OLD LINKED RECORD(PartNo)
      [Catalog]OnHand:=[Catalog]OnHand+Old(Quantity)
      SAVE OLD LINKED RECORD(PartNo)
End Case
```

References

ACTIVATE LINK, CREATE LINKED RECORD, LOAD LINKED RECORD, LOAD OLD LINKED RECORD, Old, SAVE LINKED RECORD.

SAVE RECORD

Syntax

SAVE RECORD «(*filename*)»

Description

SAVE RECORD saves the current record and all its subrecords, if any, to *filename*. If you don't specify *filename,* SAVE RECORD saves the current record of the default file.

SAVE RECORD becomes important in several instances:

- □ to save data entered after a CREATE RECORD statement
- □ to save data from RECEIVE RECORD
- □ to save a record modified by a procedure
- □ to save new or modified subrecord data following an ADD SUBRECORD, CREATE SUBRECORD, or MODIFY SUBRECORD statement

Example

```
CREATE RECORD([Customers])
RECEIVE RECORD([Customers])
If(OK=1)
  SAVE RECORD([Customers])
End if
```

References

ADD RECORD, ADD SUBRECORD, CREATE RECORD, CREATE SUBRECORD, MODIFY RECORD, MODIFY SUBRECORD, RECEIVE RECORD.

SAVE SET

Syntax

SAVE SET (*strexpr*;*docname*)

Description

SAVE SET saves the set named by *strexpr* to a document on disk named *docname*. If you supply a null string for *docname,* the standard file dialog box appears, so the user can choose the name of the file. You can bring a saved set back into memory with the LOAD SET command. If the SAVE SET operation is successful, the OK system variable returns 1. Otherwise, OK returns 0, for example, if the user clicks Cancel in the standard file dialog box.

❖ *Saved sets:* Sets are not saved in sorted order. Sets are not valid after a utility repairs a database. A saved set may or may not reflect changes to a database.

Example

```
  `GetPA; Mail List db
  `Creates & saves a set of all Palo Alto customers
DEFAULT FILE([Addresses])
ALL RECORDS
SEARCH([Addresses]City="Palo Alto")
CREATE SET("Palo Alto")   `Makes current selection a set
MESSAGE("Now saving Palo Alto set.")
SAVE SET("Palo Alto";"Palo Alto Set")
CLEAR SET("Palo Alto")
```

References

ADD TO SET, CLEAR SET, CREATE EMPTY SET, CREATE SET, DIFFERENCE, INTERSECTION, LOAD SET, Records in set, UNION, USE SET.

SAVE VARIABLE

Syntax SAVE VARIABLE (*docname*; *var*«;*»)

Description SAVE VARIABLE writes each *var* to the file named by *docname*. If *docname* is a null string, 4th Dimension presents the standard file dialog, so the user can create the variable document. If the SAVE VARIABLE operation is successful, the OK system variable returns 1. Otherwise, OK returns 0. If the user clicks Cancel in the standard file dialog box, OK returns 0.

You can retrieve these variables with the LOAD VARIABLE procedure.

❖ *Internal format used:* When you write variables to documents with SAVE VARIABLE, 4th Dimension uses an internal data format. Therefore, you can only retrieve the true values of these variables with the LOAD VARIABLE command. For example, don't use RECEIVE VARIABLE or RECEIVE PACKET to read a document created by SAVE VARIABLE.

This example clears the variable MyVar, because, if it didn't, the test wouldn't work. It then opens a document (MyDoc) and loads the desired variable (MyVar), if the variable exists. If the document and/or variable don't exist Undefined will return TRUE. Thus, the example tests for the existence of the document. If it does not exist, it gives the user the opportunity to create a variable and save it in a document.

Examples

```
 `Creates a document for variables if none exists
CLEAR VARIABLE("MyVar")
LOAD VARIABLE("MyDoc";MyVar)
If (Undefined(MyVar))
  CONFIRM("'MyDoc' doesn't exist. Create it?")
  If (OK=1)
    MyVar:=Request("Enter value of 'MyVar'")
    SAVE VARIABLE("MyDoc";MyVar)
  End if
Else
  ALERT("Value of 'MyVar' is "+MyVar)
End if

SAVE VARIABLE("Array1";X;X0;X1;X2;X3;X4)    `Save Array
```

References LOAD VARIABLE, SEND VARIABLE.

Screen height

Syntax

Screen height

Description

Screen height returns the height of the screen from the top of the menu bar to the bottom of the screen in pixels. Together with Screen width, you can create windows that will always retain the same screen position, regardless of changes in Macintosh screen size.

Example

```
  `Created a centered window on the screen.
OK:=1
While (OK=1)
   Box height:=Num(Request("Height (click cancel if done)?"))
  If (OK=1)
     Box width:=Num(Request("Width?"))
     CLOSE WINDOW `Close any open window
     H1:=Screen height/2+10 `Add 10 to allow for the menubar
     W1:=Screen width/2
     H2:=Box height/2
     W2:=Box width/2
     OPEN WINDOW(W1-W2;H1-H2;W1+W2;H1+H2)
  End if
End while
```

References

OPEN WINDOW, Screen width.

Screen width

Syntax

Screen width

Description

Screen width returns the width of the screen in pixels. Together with Screen height, you can create windows that will always retain the same screen position, regardless of changes in Macintosh screen size.

Example

```
  `Created a centered window on the screen.
OK:=1
While (OK=1)
  Box height:=Num(Request("Height (click cancel if done)?"))
  If (OK=1)
    Box width:=Num(Request("Width?"))
    CLOSE WINDOW `Close any open window
    H1:=Screen height/2+10 `Add 10 to allow for the menubar
    W1:=Screen width/2
    H2:=Box height/2
    W2:=Box width/2
    OPEN WINDOW(W1–W2;H1–H2;W1+W2;H1+H2)
  End if
End while
```

References

OPEN WINDOW, Screen height.

SEARCH

Syntax

SEARCH «(*boolexpr*) |(*filename*«;*boolexpr*»)»

Description

SEARCH creates a new selection for *filename.* SEARCH applies *boolexpr* to each record in the file. If *boolexpr* evaluates as TRUE, the record is added to the new selection. It works on all records in the file, not just those in the current selection. (If you need to search within the current selection, choose SEARCH SELECTION.) SEARCH starts with the first record and performs a sequential search, going through all records in *filename*. Each time *boolexpr* is TRUE, it places a pointer to the current record in the current selection table. When the search is complete, SEARCH loads the first record of the new selection and makes it the current record . If you don't specify *filename,* SEARCH uses the default file.

Typically, *boolexpr* tests a field or subfield against a variable or a constant, using a relational operator. *boolexpr* can contain multiple tests that are joined by AND (&) relational and/or OR (|) relational operations. *boolexpr* can also be or contain the name of a developer-written function.

You can use wildcards in a search formula when working with Alpha fields or text.

Important
Be sure to guard the precedence in Boolean expressions. If you have multiple conditions, enclose each condition in its own set of parentheses. For example: ((B>1) & (C<0)). A second example is B>(3*5).

SEARCH is significantly slower than SEARCH BY INDEX (used on indexed fields). The processing time is proportional to the number of records in the file.

❖ *User environment:* SEARCH corresponds to the Search and Search by Formula menu items in the User environment.

If you omit *boolexpr,* 4th Dimension will display the standard search window. In this case, the user can create, save, and load formulas.

During the search process, 4th Dimension displays the standard progress window, unless you have previously called MESSAGES OFF. The user can click either the Stop button in the standard progress window or the Cancel button in the standard search window. After a search, you can test the OK system variable to see if the search was completed. OK returns 1 if it was, and 0 if it wasn't. If you want to view records found by a search command, use DISPLAY SELECTION or MODIFY SELECTION.

The following examples show a number of ways in which you can use SEARCH and different search formulas.

Examples

SEARCH([Customers];[Customers]Name="Smith")

This search returns a selection of all records with Smith in the Name field.

SEARCH(([Employees]Salary>=10000)&([Employees]Salary<=50000))

This search returns a selection of all records where salary is between $10,000 and $50,000, when the Employees file is the default file.

SEARCH

Here, the search is perfomed on the default file, and the standard Search window is displayed on the screen. The user specifies the search fields and formulas in that window.

SEARCH([Invoices])

This is the same example as the above except that the statement specifies the file.

SEARCH([Customers];([Customers]ZipCode="94@")|([Customers]ZipCode="90@"))

This search returns a selection of all customers living in the San Francisco or Los Angeles areas (ZIP codes beginning with 94 or 90).

SEARCH([Laws];[Laws]Text=Var)

This statement can return several results:

1. If Var equals "Copyright@", the file selection contains all laws with texts beginning with "Copyright".
2. If Var equals "@Copyright@", the file selection contains all laws with texts containing at least one occurence of "Copyright".
3. If Var equals "@Copyright", the file selection contains all laws with texts ending with "Copyright".

SEARCH([Documents];[Documents]Keyword'Word="Macintosh")

The file selection consists of all documents with at least one subrecord whose Word subfield equals "Macintosh".

SEARCH([Invoice];MyFunc)

MyFunc is a global function and contains these instructions:

LOAD LINKED RECORD([Invoice]CustomerID)
$0 := (**Substring**([Customer]ZipCode;1;2)="94")

The invoice file selection will consist of all invoices made out to customers in the San Francisco area (when ZIP codes begin with 94).

SEARCH([filename];[filename]fieldname="")

This statement searches an empty alphanumeric field by searching for a null string.

References

Records in file, Records in selection, SEARCH BY INDEX, SEARCH SELECTION, SEARCH SUBRECORDS.

SEARCH BY INDEX

Syntax

SEARCH BY INDEX «(*fieldname*{= | ±}*expr*«{; *}»)»

Description

SEARCH BY INDEX works only on indexed fields and selects all records in the file (not just the current selection) that match the search formula, thus creating a new current selection for the file of *fieldname.expr* must be of the same type as *fieldname*. When the search is complete, SEARCH BY INDEX loads the first record of the new current selection and makes it the current record.

SEARCH BY INDEX recognizes only two operators: The equal operator (=) and the between operator (±), also known as the "plus minus" operator. The equal operator (=) tests alphanumeric, numeric, or date equality. The between operator (±) tests for alphabetic, numeric, or date values that equal or fall between its parameters. (Shift-Option-+ displays "±".)

Important

You can only use the between operator once and that is as the last test.

You can write multiple conditions, separating each condition with a semicolon. 4th Dimension automatically performs an AND operation on these conditions. If you do not give an argument, SEARCH BY INDEX displays the same dialog box as Search and Modify in the User environment, allowing the user to specify the search arguments. In this case, you can test the OK system variable. OK returns 1 if the user clicked the OK button, and 0 if the user clicked the Cancel button.

The wildcard character (@) only works with alphanumeric expressions. You can place the wildcard character at the end of an alphanumeric expression only.

Often, you can significantly speed up a complex search operation by combining SEARCH BY INDEX, set operations, and SEARCH SELECTION, instead of using SEARCH.

You can index a field in a subfile. In certain cases, like a search for a keyword in a document (the keyword being stored in an indexed subfield), you can do a document search in only a few seconds by doing a SEARCH BY INDEX on a subfield instead of doing a string search through each document in the file.

SEARCH BY INDEX does not display its findings. If you want to see the results of the search, follow the search with a command like DISPLAY SELECTION or MODIFY SELECTION.

Examples

SEARCH BY INDEX([Catalog]ItemNo=5001)

This line would select any record, bearing the item number 5001.

SEARCH BY INDEX([Catalog]ItemNo±5003;5009)

In this case, the new selection contains all records whose item number falls between 5003 (included) and 5009 (included).

SEARCH BY INDEX([Customers]Date of sale±!1/1/1990!;**Current date**)

In this case, the new selection contains all records whose date of sale falls between January 1, 1990 (included) and the current date (included).

SEARCH BY INDEX([Catalog]Customer name±"A";"M@")

In this case, the new selection contains all records whose customer name falls between "A" (included) and all names beginning with "M".

The following example searches for all people alive at a given date (d).

```
DEFAULT FILE([Persons])
  `Create a set of all people born before a date
SEARCH BY INDEX([Persons]Date of birth ± !00/00/0000!; d)
CREATE SET("Born before")
  `Create a set of all people who died after a date
SEARCH BY INDEX([Persons]Date of death ± d;!12/31/32000!)
CREATE SET("Dead after")
  `Create a set of all living people
SEARCH BY INDEX([Persons]Date of death = !00/00/0000!)
CREATE SET("Not dead")
  `Combine last 2 sets
UNION("Dead after"; "Not dead";"Alive at this date")
  `Make sure they weren't born too late
INTERSECTION("Alive at this date" ; "Born before" ; "Alive at this date")
  `Change the current selection
USE SET("Alive at this date")
  `Clean up.
CLEAR SET("Born before")
CLEAR SET("Not dead")
CLEAR SET("Dead after")
DISPLAY SELECTION
```

This procedure takes only a few seconds even if you have a file of 5000 persons. To obtain the same results with SEARCH could take several minutes.

References

CREATE SET, DIFFERENCE, INTERSECTION, Records in file, Records in selection, SEARCH, SEARCH SELECTION, UNION, USE SET.

SEARCH SELECTION

Syntax

SEARCH SELECTION (*«filename;» boolexpr*)

Description

Like SEARCH, SEARCH SELECTION performs a sequential search, but only in the current selection. SEARCH SELECTION applies *boolexpr* to each record in the current selection. If *boolexpr* evaluates as TRUE, the record is added to the new current selection. When the search is complete, SEARCH SELECTION loads the first record of the new current selection and makes it the current record. If you don't specify *filename,* SEARCH SELECTION uses the default file. You can significantly speed up a search by using SEARCH SELECTION after SEARCH BY INDEX or SEARCH to further define the selection. If the current selection is empty, SEARCH SELECTION has no effect. MESSAGES OFF turns off the standard 4th Dimension progress thermometer for this command.

Typically, *boolexpr* tests a field or subfield against a variable or a constant, using a relational operator. *boolexpr* can contain multiple tests that are joined by AND (&) and/or OR (|) operations. *boolexpr* can also be or contain the name of developer-written function.

Important

Be sure to guard the precedence in Boolean expressions. If you have multiple conditions, enclose each condition in its own set of parentheses. For example: ((B>1) & (C<0)). A second example is B>(3*5).

The example includes a file named Invoices with an indexed Date field and a field for unpaid invoices named Outstanding. To search for all the unpaid invoices in a given time period (with the variables vBegin and vEnd defining the time period) you write the following procedure:

Examples

```
SEARCH BY INDEX([Invoices]Date±vBegin;vEnd)
SEARCH SELECTION([Invoices]Outstanding#0)
DISPLAY SELECTION
```

This would usually be faster than using the SEARCH command:

```
SEARCH([Invoices];([Invoices]Date>=vBegin)&([Invoices]Date<=vEnd)&
        ([Invoices]Outstanding#0))
```

```
SEARCH SELECTION(([Customer]State="CA")&([Customer]Sales>1000))
```

References

Records in file, Records in selection, SEARCH, SEARCH BY INDEX.

SEARCH SUBRECORDS

Syntax

SEARCH SUBRECORDS (*subfilename;boolexpr*)

Description

SEARCH SUBRECORDS creates a new subselection for *subfilename.* SEARCH SUBRECORDS applies *boolexpr* to each subrecord in the subfile. If *boolexpr* evaluates as TRUE, the subrecord is added to the new subselection. When the search is complete, SEARCH SUBRECORDS makes the first subrecord the current subrecord of the current subselection. The wildcard character (@) works in string arguments.

Typically, *boolexpr* tests a subfield against a variable or a constant, using a relational operator. *boolexpr* can contain multiple tests that are joined by AND (&) and/or OR (|) operations. *boolexpr* can also be or contain the name of developer-written function.

❖ *Using filenames and field names:* In either a global or a layout procedure, you must prefix the field name with the name of the file that contains it and each subfield name with the filename and the field name that contains it.

If neither a current record nor a higher level subrecord exists, SEARCH SUBRECORDS has no effect.

Important

Be sure to guard the precedence in Boolean expressions. If you have multiple conditions, enclose each condition in its own set of parentheses. For example: ((B>1) & (C<0)). A second example is B>(3*5).

Example

SEARCH SUBRECORDS([Addresses]PhoneNos;[Addresses]PhoneNos'Area="408")

References

Records in subselection, SEARCH.

Semaphore

Syntax

Semaphore (*strexpr*)

Description

Semaphore is a multi-user function that returns TRUE if *strexpr* exists. If it does not exist, Semaphore returns FALSE and creates it. See *4th Dimension Utilities and Developer's Notes* for details on this and other multi-user commands.

SEND PACKET

Syntax

SEND PACKET (*strexpr*)

Description

SEND PACKET sends *strexpr* to the serial port or document opened by the SET CHANNEL command. SEND PACKET writes data in ASCII format. If you send the alphanumeric expression to the serial port, the receiving device can be a Macintosh executing 4th Dimension which can receive characters with RECEIVE PACKET. It can also be any other computer which can receive ASCII or any device with an RS-232 interface.

One way to export data from subfiles to files used by other applications, is to write them out with SEND PACKET. If you need to send ASCII characters directly to a serial device, you can also use SEND PACKET.

Important

Be careful; Macintosh characters are coded on 8 bits and ImageWriter characters are coded on 7 bits. If you use this method, be sure to create an ASCII map and execute USE ASCII MAP before SEND PACKET.

To write an ASCII document with 4th Dimension, first use SET CHANNEL to open the document. SEND PACKET begins writing at the beginning of the document. The writing of each subsequent packet begins immediately after the last character written.

The example uses SEND PACKET to write data from a file including subfiles to a text file.

Example

```
  `Writes a file, including subfiles to disk
SET CHANNEL(10;"SubFile.Txt")
DEFAULT FILE([I.Card])
ALL RECORDS
While (Not(End selection))
  SEND PACKET([I.Card]Name+Char(13))
  FIRST SUBRECORD([I.Card]Phone)
  While (Not(End subselection([I.Card]Phone)))
    SEND PACKET([I.Card]Phone'PhNum+Char(13))
    SEND PACKET([I.Card]Phone'Comment+Char(13))
    NEXT SUBRECORD([I.Card]Phone)
  End while
  SEND PACKET("******"+Char(13))  `End of record marker
  NEXT RECORD
End while
SEND PACKET("%%%%")  `End of file marker
SET CHANNEL(11)  `Close file
```

References

EXPORT TEXT, RECEIVE PACKET, SET CHANNEL, USE ASCII MAP.

SEND RECORD

Syntax

SEND RECORD «(*filename*)»

Description

SEND RECORD sends the current record of *filename* to the serial port or document opened by the SET CHANNEL command. If you don't specify *filename,* SEND RECORD uses the default file. If no current record exists, SEND RECORD has no effect.

Important

When sending and receiving records between databases, the file that sends the record and the file that receives the record must have the same number of fields, type of fields, and order of fields.

If you send the record to a serial port, the receiving device must be a Macintosh running 4th Dimension, which can receive the record with RECEIVE RECORD. If you send the record to a document, the record is written into the document. In either case, 4th Dimension sends the complete record (along with any subrecords it might contain) in an internal data format, which can only be received with RECEIVE RECORD. SEND RECORD and RECEIVE RECORD are useful for archiving records in a separate document, before deleting records. See the archiving example in Appendix A.

If you want to send and receive data in ASCII format, choose SEND PACKET and RECEIVE PACKET or EXPORT TEXT and IMPORT TEXT.

Example

```
DEFAULT FILE([My File])
SET CHANNEL(10;"MyFile.Text")
While(Not(End selection))
  SEND RECORD
  NEXT RECORD
End while
SET CHANNEL(11)
```

References

ON SERIAL PORT CALL, RECEIVE RECORD, RECEIVE VARIABLE, SEND VARIABLE, SET CHANNEL.

SEND VARIABLE

Syntax SEND VARIABLE (*var*)

Description SEND VARIABLE sends a variable (*var*) to the serial port or document opened by the SET CHANNEL command. If you send the variable to a serial port, the receiving device must be a Macintosh running 4th Dimension, which can receive the variable with RECEIVE VARIABLE. If you send the variable to a document, the variable is written into the document. In either case, 4th Dimension sends the complete variable (including its type and value) in an internal data format, which can only be received with RECEIVE VARIABLE. Don't confuse SEND VARIABLE with SAVE VARIABLE.

Example

```
SEND VARIABLE(AcctNo)
```

References RECEIVE VARIABLE, SAVE VARIABLE, SEND PACKET, SEND RECORD, SET CHANNEL.

SET CHANNEL

SET CHANNEL has two syntaxes. The first identifies a serial port, protocol, and other serial port information for setting a communications port. The second syntax identifies and acts on a document. In either case, serial port or document, you send data by using SEND PACKET, SEND RECORD, or SEND VARIABLE and you receive data with RECEIVE BUFFER (only for serial port), RECEIVE PACKET, RECEIVE RECORD, or RECEIVE VARIABLE. The numeric codes for both syntaxes are in Appendix F, "4th Dimension and Macintosh Codes."

Syntax 1 SET CHANNEL (*posintexpr1*;*posintexpr2*)

Description 1 SET CHANNEL sets the serial port number (*posintexpr1*). You set the speed, number of data bits, number of stop bits, and parity by adding the numeric codes for these arguments to form *posintexpr2*. SET CHANNEL selects a serial port as the current serial port. The current serial port is the default serial port for ON SERIAL PORT CALL and all subsequent data receiving operations. The example sets a channel to print through the printer port to an Imagewriter II (to send data with SEND PACKET).

Example 1

```
SET CHANNEL(0;10+3072+16384+8192)    `Printer port, ImageWriter
```

References 1 ON SERIAL PORT CALL, RECEIVE BUFFER, RECEIVE PACKET, RECEIVE RECORD, RECEIVE VARIABLE, SEND PACKET, SEND RECORD, SEND VARIABLE.

Syntax 2 SET CHANNEL (*posintexpr*«;*docname*»)

Description 2 SET CHANNEL can create, open, and close files. The *posintexpr* argument specifies whether to open (10,12, and 13) or close (11) a file. Files created with SET CHANNEL are Text files.

❖ *Documents and the serial port:* SET CHANNEL on a document doesn't affect the current serial port, but you can only send and receive data from the document and not to and from the serial port until you close the document.

Here are the values for *posintexpr* and *docname*.

Create a document

- ☐ SET CHANNEL(12;""): With this syntax, SET CHANNEL displays the standard Create document dialog. If the user creates (and consequently opens) a document, OK returns 1 and the Document variable returns the name of the opened document. Otherwise, OK returns 0. If you pass 12 as *posintexpr,* you must pass an empty string as the second argument.

Open a document

- ☐ SET CHANNEL(10;*docname*): With this syntax, SET CHANNEL tries to open a Macintosh document specified by *docname*. If the document exists, SET CHANNEL opens it. If not, SET CHANNEL creates a new document and opens it. With this syntax, the OK and Document variables return no significant values.
- ☐ SET CHANNEL(10;""): With this syntax, SET CHANNEL displays the standard Open document dialog. If the user opens a document OK returns 1 and the Document variable returns the name of the opened document. Otherwise, OK returns 0.
- ☐ SET CHANNEL(13;""): With this syntax, SET CHANNEL displays the standard Open document dialog. If the user opens a document, OK returns 1 and the Document variable returns the name of the opened document. Otherwise, OK returns 0. If you pass 13 as *posintexpr,* you must pass an empty string as the second argument.

❖ *File types:* SET CHANNEL(10;"") and SET CHANNEL(13;"") are different. With value 10, you can open any type of file, while with value 13, only the text type files are displayed in the standard window.

Close a document

- ☐ SET CHANNEL(11): If you pass 11 to *posintexpr,* you must not pass a second argument. With this syntax, SET CHANNEL closes the document you previously opened.

Warning

You must ALWAYS close a document (you previously opened) with SET CHANNEL(11): if you don't, your new document won't be saved and the document you modified may have its contents damaged.

❖ *HFS and MFS:* The contents of the Document variable depend on the file system you're using. Suppose you access a file named Sales Table contained in a folder named Sales Folder, nested in a folder named Business saved on a disk named Office.

If you work on an MFS volume, the Document variable returns Office:SalesTable since the MFS does not work with hierarchical folders. On the other hand, if you work on an HFS volume, the Document variable returns Office:Business:Sales Folder:Sales Table, because with HFS every folder represents a level in the volume directory.

Example 2

```
  `Writes a file, including subfiles to disk
i:=1
SET CHANNEL(10;"SubFile.Txt")
DEFAULT FILE([I.Card])
ALL RECORDS
While (Not(End selection))
  SEND PACKET([I.Card]Name+Char(13))
  FIRST SUBRECORD([I.Card]Phone)
  While (Not(End subselection([I.Card]Phone)))
    SEND PACKET([I.Card]Phone'PhNum+Char(13))
    SEND PACKET([I.Card]Phone'Comment+Char(13))
    NEXT SUBRECORD([I.Card]Phone)
  End while
  SEND PACKET("******"+Char(13))  `End of record marker
  NEXT RECORD
  MESSAGE("Record "+String(i)+" saved to text file.")
  i:=i+1
End while
SEND PACKET("%%%%")  `End of file marker
SET CHANNEL(11)
```

References 2 ON SERIAL PORT CALL, RECEIVE BUFFER, RECEIVE PACKET, RECEIVE RECORD, RECEIVE VARIABLE, SEND PACKET, SEND RECORD, SEND VARIABLE.

SET WINDOW TITLE

Syntax SET WINDOW TITLE (*strexpr*)

Description SET WINDOW TITLE changes the title of the current window to that specified by *strexpr*. The current window may be the 4th Dimension window or the custom window, opened with OPEN WINDOW. If you work in custom menu, 4th Dimension gives the menu window the default title "Custom Menus." You can change this to the title of your choice with SET WINDOW TITLE. Your custom title will remain until you change it with another SET WINDOW TITLE command. If you create a custom window with a title bar, you specify the title in the OPEN WINDOW command. You can also change it with SET WINDOW TITLE.

❖ *Titles in the User environment:* You can also use SET WINDOW TITLE in the User environment, but remember that 4th Dimension automatically changes the title of its window when you select User environment commands. For example, when you select a command like New Record in the Entry menu, 4th Dimension sets the title to "Entry for filename".

Example

```
SET WINDOW TITLE("My First Application")
```

Reference OPEN WINDOW.

Sin

Syntax Sin (*numexpr*)

Description Sin returns the sine of *numexpr*, where *numexpr* is given in radians. (One degree equals 0.0174532925199432958 radians.) The example returns the sine of 45°, 0.70710678118654725.

Example

```
vSine:=Sin(45*0.0174532925199432958)
```

References Arctan, Cos, Exp, Log, Tan.

SORT BY INDEX

Syntax SORT BY INDEX (*fieldname*;> | <)

Description SORT BY INDEX sorts the current selection of the file containing *fieldname* into ascending or descending order. *fieldname* must be an indexed field.

Important
SORT BY INDEX can sort only one level.

You indicate ascending order with the "greater than" sign (>) and a descending sort selection with the "less than" sign (<). Once the sort is completed, 4th Dimension loads the first record in the sorted selection and makes it the current record. SORT BY INDEX is useful when you want to sort a big selection on only one level, because SORT BY INDEX is much faster than SORT SELECTION.

During a sort operation, 4th Dimension displays the standard progress window unless you have previously called MESSAGES OFF. After a sort operation, you can test to see if the operation was completed by checking the OK system variable. OK returns 1 if the operation was completed and 0 if the user has clicked the Stop button.

Example

```
SORT BY INDEX([Addresses]ZIP;>)
```

Reference SORT SELECTION.

SORT SELECTION

Syntax 1 SORT SELECTION (*fieldname*;>|<{;*})

Syntax 2 SORT SELECTION «(*filename*)»

Description SORT SELECTION sorts the current selection of the file containing *fieldname* into ascending or descending order. You can sort on multiple fields within one statement. You indicate ascending order with the "greater than" sign (>) and a descending sort selection with the "less than" sign (<). Once the sort is completed, 4th Dimension loads the first record of the sorted selection and makes it the current record.

Syntax 1 requires that you explicitly state all arguments. The sort will take place exactly as you demand.

Important
With SORT SELECTION, you must prefix every use of fieldname with the filename of the file to which the field belongs. This is true whether the procedure is global or layout.

Syntax 2 opens a dialog box. When you give *filename* as an argument, 4th Dimension brings up the standard sort window for *filename*. If you omit *filename,* the default file is used.

During a sort operation, 4th Dimension displays the standard progress window, unless you have previously called MESSAGES OFF. After a sort operation, you can test to see if the operation was completed by checking the OK system variable. OK returns 1 if the operation was completed and 0 if it wasn't. The user clicking the Cancel button in the standard sort window or the Stop button in the standard progress window causes OK to return a 0.

When sorting for report breaks, remember that you must sort on one extra field. Breaks are generated for all but the last sorted field. This gives you the ability to have the last field in a desired order without that order generating a break. If you want to break on the last sorted field, sort it twice.

Example 1

```
SORT SELECTION([Addresses]ZIP;>;[Addresses]Name2;>)
```

Example 2

```
SORT SELECTION([Addresses])
```

References In break, Level, SORT BY INDEX, Subtotal.

SORT SUBSELECTION

Syntax SORT SUBSELECTION (*subfilename*;*subfieldname*;> | <{;*})

Description SORT SUBSELECTION sorts the current subselection of a subfile specified by *subfilename* within the current record on specified *subfieldname(s)* into ascending or descending order. You indicate ascending order with the "greater than" sign (>) and a descending sort selection with the "less than" sign (<). Once the sort is completed, 4th Dimension makes the first subrecord of the sorted subselection the current subrecord. Sorting subrecords is a dynamic process. Subrecords are never saved in their sorted order. If neither a current record nor a higher level subrecord exists, SORT SUBSELECTION has no effect.

In either a global or a layout procedure, you must prefix the field name with the name of the file that contains it and each subfield name with the filename and the field name that contains it.

If you do a SORT SUBSELECTION in the Before phase of a layout procedure, the subselection appears sorted. If you do the same thing in the During phase of a layout procedure for a screen display, you must call REDRAW to show the new order.

Example

```
SORT SUBSELECTION([Stats]Sales;[Stats]Sales'Bucks;>)
```

Reference None.

Squares sum

Squares sum has two syntaxes. The first applies to finding the squares sum of a particular field in the current selection. It only works in a footer or break in an output layout procedure with PRINT SELECTION. The second applies to finding the squares sum of a particular subfield in a specified subfile. In both cases, the field type must be numeric.

Syntax 1 Squares sum (*fieldname*)

Description 1 Squares sum returns the sum of squares for *fieldname* in the current selection. This works only in a footer or break when printing with PRINT SELECTION, because to compute a squares sum figure, 4th Dimension must work with each record in the selection. At print time, 4th Dimension brings each record into memory and builds the sum of squares figure.

Important

Squares sum does not clear after a break. Rather, it calculates a record-by-record sum of squares value for the entire selection. Squares sum is only meaningful when printed in break level 0.

Example 1

```
If(In footer&End selection([Income]))
   vSqSum:=Squares sum([Income]Sales)
End if
```

References 1 Average, In break, In footer, Max, Min, PRINT SELECTION, Std deviation, Subtotal, Sum, Variance.

Syntax 2 Squares sum (*subfieldname*)

Description 2 Squares sum returns the sum of squares of *subfieldname* of the current subselection of the current record.

Example 2

```
vSqSum:=Squares sum([Invoice]Ordered'ItemTotal)
```

References 2 Average, Max, Min, Std deviation, Sum, Variance.

Std deviation

Std deviation has two syntaxes. The first applies to finding the standard deviation of a particular field in the current selection. It only works in a footer or break in an output layout with PRINT SELECTION. The second applies to finding the standard deviation of a particular subfield. In both cases, the field type must be numeric.

Syntax 1 Std deviation (*fieldname*)

Description 1 Std deviation returns the standard deviation for *fieldname* in the current selection. This works only in a footer or break when printing, because to compute a standard deviation figure, 4th Dimension must work with each record in the selection. At print time, 4th Dimension brings each record into memory and builds the figure.

Important

Std deviation does not clear after a break. Rather, it calculates a record-by-record standard deviation value for the entire selection. Std deviation is only meaningful when printed in break level 0.

Example 1

```
If(In footer & End selection([Income]))
   vStdDev:=Std deviation([Income]Sales)
End if
```

References 1 Average, In break, In footer, Max, Min, PRINT SELECTION, Squares sum, Subtotal, Sum, Variance.

Syntax 2 Std deviation (*subfieldname*)

Description 2 Std deviation returns the standard deviation of *subfieldname* of the current subselection of the current record.

Example 2

```
vStdDev:=Std deviation([Invoice]Ordered'ItemTotal)
```

References 2 Average, Max, Min, Squares sum, Sum, Variance.

String

String has two syntaxes.

Syntax 1 String (*numexpr«;strexpr»*)

Description 1 String returns a string, the numeric value specified by *numexpr*. The format can be set by the optional *strexpr*. Choose this format when you want to display the result in a specific format. The second example would return the appropriate string, depending on the number tested.

Example 1

```
ALERT("You now have "+String(Records in file)+" records in this file.")
v:=String(vNum;"Positive;Negative;Zero")
```

References 1 Ascii, Char, Length, Num, Position, Substring.

Syntax 2 String (*date*)

Description 2 String returns in an alphanumeric form, the date value specified by *date* in the format MM/DD/YY.

Example 2

```
ALERT ("Most recent sale for "+gRep+" was on "+ String([Sales]SalesDate))
```

References 2 Current date, Date string, Position, Substring.

Substring

Syntax

Substring (*strexpr;posintexpr1;posintexpr2*)

Description

Substring returns the portion of *strexpr* defined by *posintexpr1* and *posintexpr2*. *posintexpr1* points to the nth character in *strexpr*. *posintexpr2* gives the number of characters to be returned from and including the nth character.

❖ *Note:* If the sum of *posintexpr1* and *posintexpr2* exceeds 32,766, the results are undefined.

If *posintexpr1* plus *posintexpr2* is greater than the number of characters in the *strexpr,* Substring returns the last character(s) in the string, starting with the character specified by *posintexpr1*. If *posintexpr1* is greater than the number of characters in the string, Substring returns a null string ("").

Example

```
Daycode:=Substring("05/26/87";4;2)
```

References

GET HIGHLIGHTED TEXT, HIGHLIGHT TEXT, Length, Position, String.

Subtotal

Syntax Subtotal (*fieldname*«;*posintexpr*»)

Description Subtotal works only when printing a sorted selection with PRINT SELECTION. It creates subtotals by sort level for *fieldname*. You must call Subtotal in the Break phase of the output layout procedure. The *fieldname* argument must be of type Real, Integer, or Long Integer. To display subtotals while printing, assign the result of the Subtotal function to a variable placed in the Break area of the print layout.

The second optional argument is used to cause page breaks during printing. If *posintexpr* is 0, Subtotal will not issue a page break. If *posintexpr* equals 1, Subtotal issues a page break for each level 1 break. If *posintexpr* equals 2, Subtotal issues a page break for each level 1 and level 2 break, and so on.

❖ *Sum or Subtotal:* The figure returned by Subtotal totals figures only for the current break. 4th Dimension clears the Subtotal figure after each break. Sum, on the other hand, computes a total for the entire selection being printed. This makes Sum ideal for generating running totals. Sum does not issue breaks.

If you want to have breaks on *n* sort levels, you must sort the current selection on *n* + 1 levels. This feature lets you sort on a last field, so that it doesn't create unwanted breaks. If you want the last sort field to generate a break, sort it twice.

Subtotal must be present in the output layout procedure and sorting must be done for any breaks other than level zero to be generated by PRINT SELECTION. Subtotal in a global procedure called by the output layout procedure does not trigger breaks.

❖ *Provoking page breaks:* You can use Subtotal to provoke a page break without printing a subtotal results. To do this, you can call Subtotal anywhere in the output layout procedure and do not assign the result to an ouput layout variable.

Example

```
If (Before)
  gQuarter:=[Income]Quarter
  vStr1:=""
  vStr2:=""
End if
If (In break)
  Case of
    : (Level=1)
      vStr1:="Subtotal for Quarter "+String(gQuarter)+": $"+String(Subtotal(Sale))
    : (Level=0)
     vStr1:="Final figures to date....    Maximum: $"+String(Max([Income]Sale))
     vStr2:="                             Total: $"+String(Sum(Sale))
  End case
End if
```

References In break, In footer, Level, SORT SELECTION, Sum.

Sum

Sum has two syntaxes. The first applies to finding the sum of a particular field in the current selection. It only works in a footer or break at print time with PRINT SELECTION. The second applies to finding the sum of a particular subfield for the current subselection in a specified subfile. In both cases, the field type must be numeric.

Syntax 1 Sum (*fieldname*)

Description 1 Sum returns the sum for *fieldname* in the current selection. This works only in a footer or break of an output layout procedure when printing with PRINT SELECTION, because to compute a sum, 4th Dimension must work with each record in the selection. At print time, 4th Dimension brings each record into memory and builds the sum figure.

❖ *Sum or Subtotal:* The figure returned by Subtotal totals figures only for the current break. 4th Dimension clears the Subtotal figure after each break. Sum, on the other hand, computes a total for the entire selection being printed. This makes Sum ideal for generating running totals.

Important

Sum does not clear after a break. Rather, it calculates a record-by-record total for the entire selection. Sum is only meaningful when printed in break level 0.

Example 1

```
If(In footer&End selection([Income]))
   vSum:=Sum([Income]Sales)
End if
```

References 1 Average, In break, In footer, Max, Min, Squares sum, Std deviation, Subtotal, Variance.

Syntax 2 Sum (*subfieldname*)

Description 2 Sum returns the sum of *subfieldname* for the current subselection of the current record.

Example 2

```
vSum:=Sum([Invoice]Ordered'ItemTotal)
```

References 2 Average, Max, Min, Squares sum, Std deviation, Variance.

Tan

Syntax Tan (*numexpr*)

Description Tan returns the tangent of *numexpr*, where *numexpr* is given in radians. (One degree equals 0.0174532925199432958 radians.) The example returns the tangent of 45°, 1.00.

Example

```
vTan:=Tan(45*0.0174532925199432958)
```

References Arctan, Cos, Exp, Log, Sin.

Time

Syntax Time (*strexpr*)

Description Time returns a numeric value equal to the number of seconds between midnight and the time specified by *strexpr*. *strexpr* must follow the HH:MM:SS format. *strexpr* must be in 24-hour time. The example displays an Alert box with the message "The time at 1:00PM is 46800 seconds."

Example

```
ALERT("The time at 1:00PM is "+String(Time("13:00:00"))+" seconds.")
```

References Current time, Time string.

Time string

Syntax

Time string (*posintexpr*)

Description

Time string returns a string value that expresses the time in the HH:MM:SS format for *posintexpr,* where *posintexpr* is a positive integer representing the number of seconds since midnight.

If you go beyond the number of seconds in a day (86,400), Time string continues to add hours, minutes, and seconds. For example, Time string(86401) yields 24:00:01.

Example

```
  `Convert 24 hour time to AM/PM
$0:=""
  `First, is it AM or PM
If($1<43200)
  End Time:=" AM"
Else
  End Time:=" PM"
  $1:=$1-43200   `Subtract 12 hours for PM
End if
If ($1<3600)   `If the time is less than 1:00 add 12 hours
  $1:=$1+43200
End if
If ($1<36000)
    `If the time is before 10:00 strip the leading zero
   $0:=Substring(Time string($1);2;4)+End Time
Else
   $0:=Substring(Time string($1);1;5)+End Time
End if
```

References

Current time, Time.

TRACE

Syntax TRACE

Description After calling TRACE, 4th Dimension automatically displays the Trace window every time a statement is executed. This state ends either when you click the No Trace button in the Trace window or when 4th Dimension executes a NO TRACE command in a subsequent statement. The TRACE facility has a global scope. It remains in effect for all procedures executed: global procedures and layout procedures for data entry, printing, and importing and exporting data.

❖ *Option-Click:* You can invoke the TRACE facility by pressing Option and clicking during the execution of any procedure. This brings up the standard error window. To invoke TRACE, click the Trace button. If the database is password protected, you can use the Option-click method only if you hold the master password. Also see Appendix B, "Using TRACE."

Example

```
TRACE
```

Reference NO TRACE.

True

Syntax True

Description True returns the Boolean value TRUE.

Example

```
  `True demo
MyVar := True
If(MyVar)
  str:="I'm true."
Else
  str:="I'm false."
End if
ALERT(str)
```

References False, Not.

Trunc

Syntax

Trunc (*numexpr;intexpr*)

Description

Trunc returns *numexpr* with its decimal part truncated by the number of decimals specified by *intexpr*. Trunc always truncates toward negative infinity. Like any other numeric operation, Trunc affects the value in memory and not the display format.

Examples

```
vTot:=Trunc(216.897;1)    `vTot gets 216.8
vTot:=Trunc(216.897;-1)   `vTot gets 210
vTot:=Trunc(-216.897;1)   `vTot gets -216.9
vTot:=Trunc(-216.897;-1)  `vTot gets -220
```

References

Abs, Dec, Int, Mod, Random, Round.

Undefined

Syntax

Undefined (*var*)

Description

Undefined returns TRUE if no value has been assigned to *var* or if you used CLEAR VARIABLE on *var*. As shown in the example, Undefined comes in handy for checking whether documents read by LOAD VARIABLE exist and whether variables have been assigned. If you try to assign an undefined variable to a field or to another variable or if you try to compare it with an expression or value, you may get an error message.

Another use is to determine if a parameter has been passed to a subroutine:

```
If(Undefined($3))
  $3:="None"
End if
```

❖ *OK undefined:* The system variable OK is not always defined. See Appendix E for a discussion of OK and other system variables.

Important

Always define variables.

Example

```
  `Creates a document for variables if none exists
CLEAR VARIABLE("MyVar")
LOAD VARIABLE("MyDoc";MyVar)
If (Undefined(MyVar))
  CONFIRM("'MyDoc' doesn't exist. Create it?")
  If (OK=1)
    MyVar:=Request("Enter value of 'MyVar'")
    SAVE VARIABLE("MyDoc";MyVar)
  End if
Else
  ALERT("Value of 'MyVar' is "+MyVar)
End if
```

References

CLEAR VARIABLE, LOAD VARIABLE, SAVE VARIABLE.

UNION

Syntax

UNION (*strexpr1;strexpr2;strexpr3*)

Description

UNION creates a set from the union of two sets: set1 (specified by *strexpr1)* and set2 (specified by *strexpr2*). *strexpr3* names the resulting set, set3. UNION includes all elements of both sets. Set3 needn't exist prior to issuing this command. In fact, if it does exist, 4th Dimension will replace the old set after computing the new set. Further, Set3 can be either set1 or set2.

Because SEARCH BY INDEX does not let you perform an OR operation on search conditions, you can do indexed searches, assigning the results of each search to a set and then perform an OR operation on the sets with UNION. When you have completed the UNION operations, call USE SET to make your current selection reflect the contents of the new set.

Example

```
  `Find all customers in California and New York
SEARCH BY INDEX([Customer]State="CA")
CREATE SET("CA")
SEARCH BY INDEX([Customer]State="NY")
CREATE SET("NY")
UNION("CA";"NY";"CA AND NY")
USE SET("CA AND NY")
CLEAR SET("CA")
CLEAR SET("NY")
CLEAR SET("CA AND NY")
```

References

CLEAR SET, DIFFERENCE, INTERSECTION, SAVE SET, USE SET.

UNLOAD RECORD

Syntax

UNLOAD RECORD «(*filename*)»

Description

UNLOAD RECORD is used in multi-user applications. You must call UNLOAD RECORD when you want to unlock the current record of *filename*. See *4th Dimension Utilities and Developer's Notes* for details on this and other multi-user commands.

Uppercase

Syntax Uppercase (*strexpr*)

Description Uppercase returns a string in which all the characters of *strexpr* are in uppercase. The example is a function which capitalizes the argument.

Example

```
  `Function Capitalize: Puts first letter in upper case and rest in lowercase
$0:=Uppercase(Substring($1;1;1))+Lowercase(Substring($1;2;(Length($1)–1)))
```

References Ascii, Char, Length, Lowercase, Position, Substring.

USE ASCII MAP

Syntax USE ASCII MAP (*docname*|*)

Description USE ASCII MAP loads into memory the ASCII map *docname* and makes it the current ASCII map. This document must have been previously created with the ASCII map dialog in the User environment. Once an ASCII map is loaded, 4th Dimension uses it during transfer of data between the database and a document or a serial port. This includes the import and export of text (ASCII), DIF, and SYLK files. It also works on data transferred through SEND PACKET, RECEIVE PACKET, and RECEIVE BUFFER. It has no effect on transfers of data done with SEND RECORD, SEND VARIABLE, RECEIVE RECORD, and RECEIVE VARIABLE.

If you give a null string for *docname,* 4th Dimension displays the standard open file dialog so the user can specify an ASCII map. Whenever you execute USE ASCII MAP, OK returns 1 if the map is successfully loaded, and 0 if not.

If you give the alternative asterisk argument instead of *docname,* 4th Dimension will restore the default Macintosh ASCII map. Normally, you should give this instruction after your map-oriented activities are finished. Otherwise, you will run under the map you invoked.

xample

```
USE ASCII MAP("MyChars")
EXPORT TEXT([MyFile];"MyText")
USE ASCII MAP(*)
```

eferences EXPORT DIF, EXPORT SYLK, EXPORT TEXT, IMPORT DIF, IMPORT SYLK, IMPORT TEXT, RECEIVE BUFFER, RECEIVE PACKET, SEND PACKET.

USE SET

Syntax

USE SET (*strexpr*)

Description

USE SET makes the records in the set named *strexpr* the current selection for the file to which the set belongs. You create the set named by *strexpr* with commands like UNION, DIFFERENCE, INTERSECTION, and ADD TO SET. This is useful for managing groups of records with set operations, and from these operations creating a selection that reflect these groupings. CREATE SET is the counterpart of USE SET; it creates a set whose records are the same as those in the current selection.

❖ *Sets and the current record:* When you create a set, the position of the currrent record is kept in the set. USE SET retrieves the position of this record and makes it the new current record. If you delete this record before you execute USE SET, 4th Dimension selects the first record in the set as the current record. Also, if you form a set that does not contain the position of the current record, USE SET selects the first record in the set as the current record.

Warning

If the data file has changed (additions and deletions), LOAD SET and USE SET may select records that were not in the original set.

Example

```
  `Saves a text file of all Palo Alto customers
DEFAULT FILE([Addresses])
LOAD SET([Addresses];"Palo Alto";"Palo Alto Set")
USE SET("Palo Alto")
OUTPUT LAYOUT("Exporter")
EXPORT TEXT("Palo Alto Addresses")
CLEAR SET("Palo Alto")
OUTPUT LAYOUT("Output1")
```

References

ADD TO SET, CLEAR SET, CREATE EMPTY SET, CREATE SET, DIFFERENCE, INTERSECTION, LOAD SET, Records in set, SAVE SET, UNION.

Variance

Variance has two syntaxes. The first applies to finding the variance of values of a particular field in the current selection. It only works in a footer or break at print time with PRINT SELECTION. The second applies to finding the variance of values of a particular subfield. In both cases, the field type must be numeric.

Syntax 1 Variance (*fieldname*)

Description 1 Variance returns the variance for *fieldname* in the current selection. This works only in In footer or In break in an output layout procedure and when printing with PRINT SELECTION, because to compute a variance, 4th Dimension must work with each record in the selection. At print time, 4th Dimension brings each record into memory and builds the variance figure.

Important

Variance does not clear after a break. Rather, it calculates a record-by-record variance for the entire selection. Variance is meaningful only when printed in break level 0.

Example 1

```
If(In footer & End selection([Income]))
   vVariance:=Variance([Income]Sales)
End if
```

References 1 Average, In break, In footer, Max, Min, Squares sum, Std deviation, Sum.

Syntax 2 Variance(*subfieldname*)

Description 2 Variance returns the variance of *subfieldname* for the current subselection of the current record.

Example 2

```
vVar:=Variance([Invoice]Ordered'ItemTotal)
```

References 2 Average, Max, Min, Squares sum, Std deviation, Sum.

While...End while

Syntax

While(*boolexpr*)
«*statement(s)*»
End while

Description

While...End while loops as long as *boolexpr* evaluates to TRUE and executes the *statement(s)*.

Important
Be sure to guard the precedence in Boolean expressions. If you have multiple conditions, enclose each condition in its own set of parentheses. For example: ((B>1) & (C<0)). A second example is B>(3*5).

You can nest While statements within While statements, as long as the close of an inner While statement does not appear after the close of an outer While statement.

Example

```
bOK:=1
DEFAULT FILE([CUSTOMERS])
INPUT LAYOUT("AddRecs")
While(bOK=1)
  ADD RECORD
End while
```

References

Case of...Else...End case, If...Else...End if.

Year of

Syntax Year of (*date*)

Description Year of returns the numeric value for year from *date*. The example searches for all records in where [Invoice History]Invoice Date (a field of type Date) shows the year 1990. Then, upon the user's permission, it displays this selection.

Example

```
  `Display 1990 Invoices
DEFAULT FILE([Invoice History])
ALL RECORDS
SEARCH(Year of([Invoice History]Invoice Date)=1990)
CONFIRM("You have "+String(Records in selection)+" invoices. Display them?")
If (OK=1)
  DISPLAY SELECTION
End if
```

References Current date, Date, Day number, Day of, Month of.

Appendix A

Example Programs

Appendix A gives examples that you can use as models in building your own 4th Dimension applications.

Searching, sorting, and printing

This example shows you how to set up a layout and a procedure through which the user can search, sort, and print selected records. The example uses an animal database with one file, Families, containing four fields, Species, Origin, Friend, and Population. The demo layout is shown in Figure A-1.

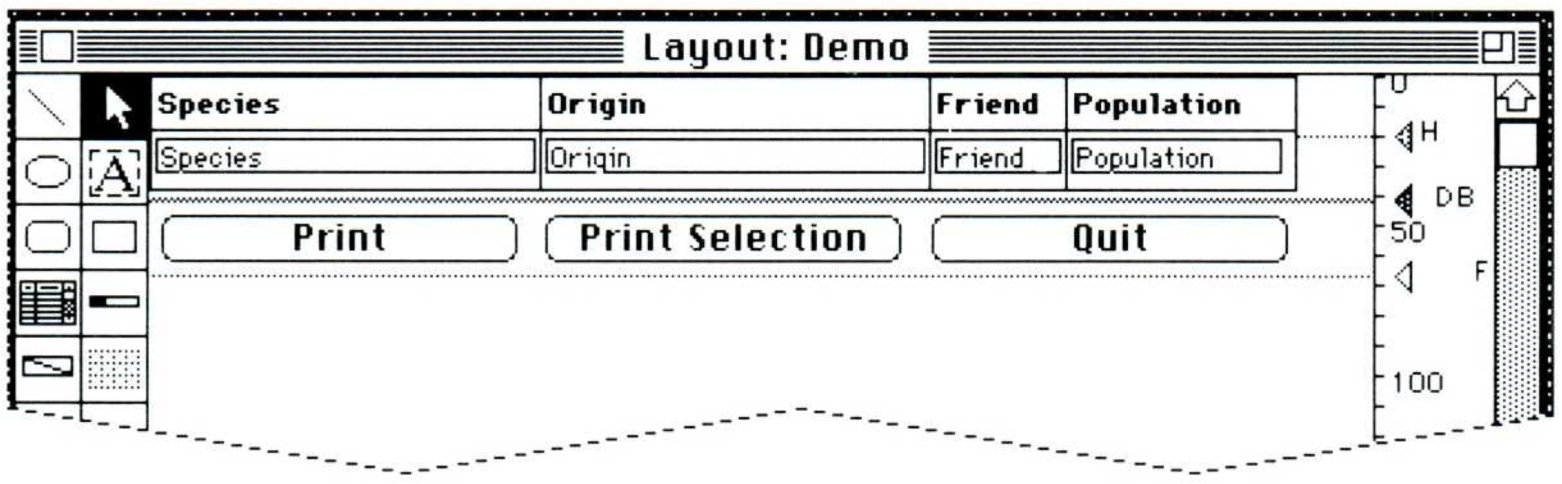
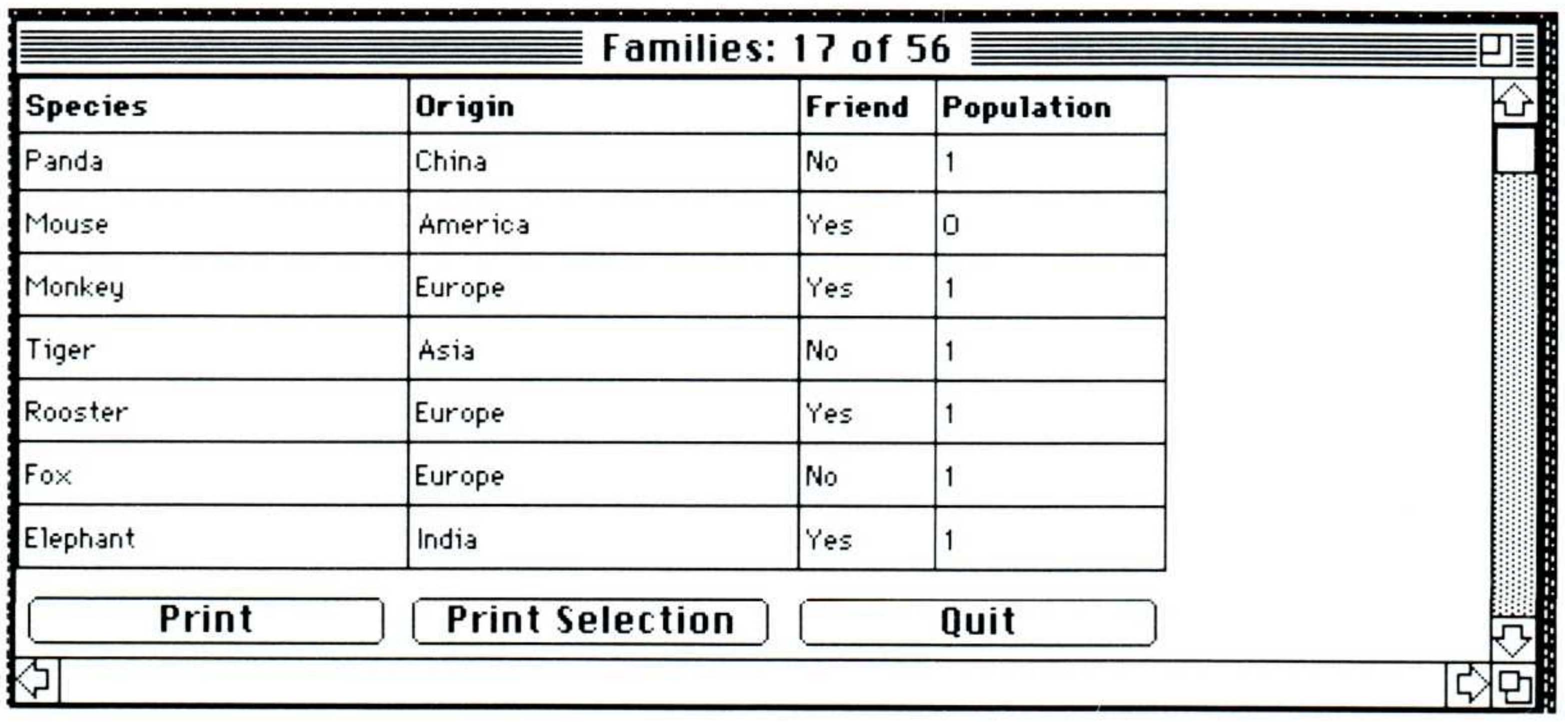


Figure A-1
Output layout

The three buttons are Accept buttons: Print is named bPrint, Print Selection is named bRecords, and Quit is named bQuit. Typically, you would attach the following global procedure to a menu item:

```
  `Allows the user to search, sort and make selections to be printed.
DEFAULT FILE([Families])
OUTPUT LAYOUT("Demo")
  `Display on the screen the standard Search window.
SEARCH
If ((OK=1)&(Records in selection>=1))
    `If the user validates the search, display the standard Sort window.
  SORT SELECTION
  If (OK=1)
      `If the user validates the sort:
    bQuit:=0
    While (bQuit=0)
          `While the user doesn't click the Quit button, display the selection.
      DISPLAY SELECTION
      If (bPrint=1)
            `If the user clicks the Print button, print the selection.
        PRINT SELECTION
      End if
      If (bRecords=1)
            `If the user clicks the Print Selection button,
            `save the current selection in the set named "Selection".
        CREATE SET("Selection")
            `Build the new selection with the "UserSet" set that the user selected.
        USE SET("UserSet")
            `Print that selection.
        PRINT SELECTION
            `Restore the original selection (Unsorted!).
        USE SET("Selection")
            `Delete the "Selection" set which is no longer useful.
        CLEAR SET("Selection")
      End if
    End while   `Until Quit button is pressed
  End if   `Sort is OK
End if   `Search is OK
```

Figure A-2 shows the list as it appears on the screen.



Species	Origin	Friend	Population
Panda	China	No	1
Mouse	America	Yes	0
Monkey	Europe	Yes	1
Tiger	Asia	No	1
Rooster	Europe	Yes	1
Fox	Europe	No	1
Elephant	India	Yes	1

Figure A-2
List displayed on the screen

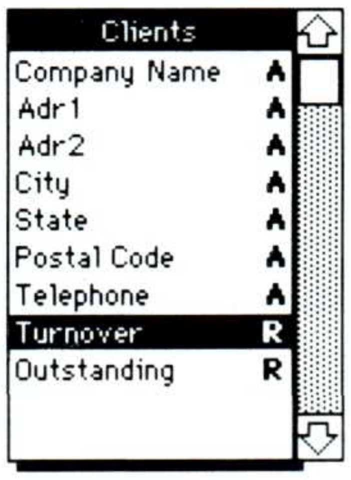
Sets and the current selection

To delete the current record while keeping the same current selection:

```
  `Deletes the current record in [INTL CONVENTION] file while maintaining the rest of the
  `current selection.

  `Create a set containing the current selection:
CREATE SET([INTL CONVENTION];"Selection")
  `Delete the current record.
DELETE RECORD([INTL CONVENTION])
  `Create a new current selection using the set named "Selection". This will cause all
  `undeleted records to be used as the current selection. Note: The current record gets
  `reset to the first record of the selection.
USE SET("Selection")
CLEAR SET("Selection")
```

Printing

This example will print the current selection and show subtotals for the annual turnover and outstanding debts for every city of the file shown in Figure A-3.


Clients

Company Name	A
Adr1	A
Adr2	A
City	A
State	A
Postal Code	A
Telephone	A
Turnover	R
Outstanding	R

Figure A-3
File for Printing example

Create the layout named Printing shown in Figure A-4.

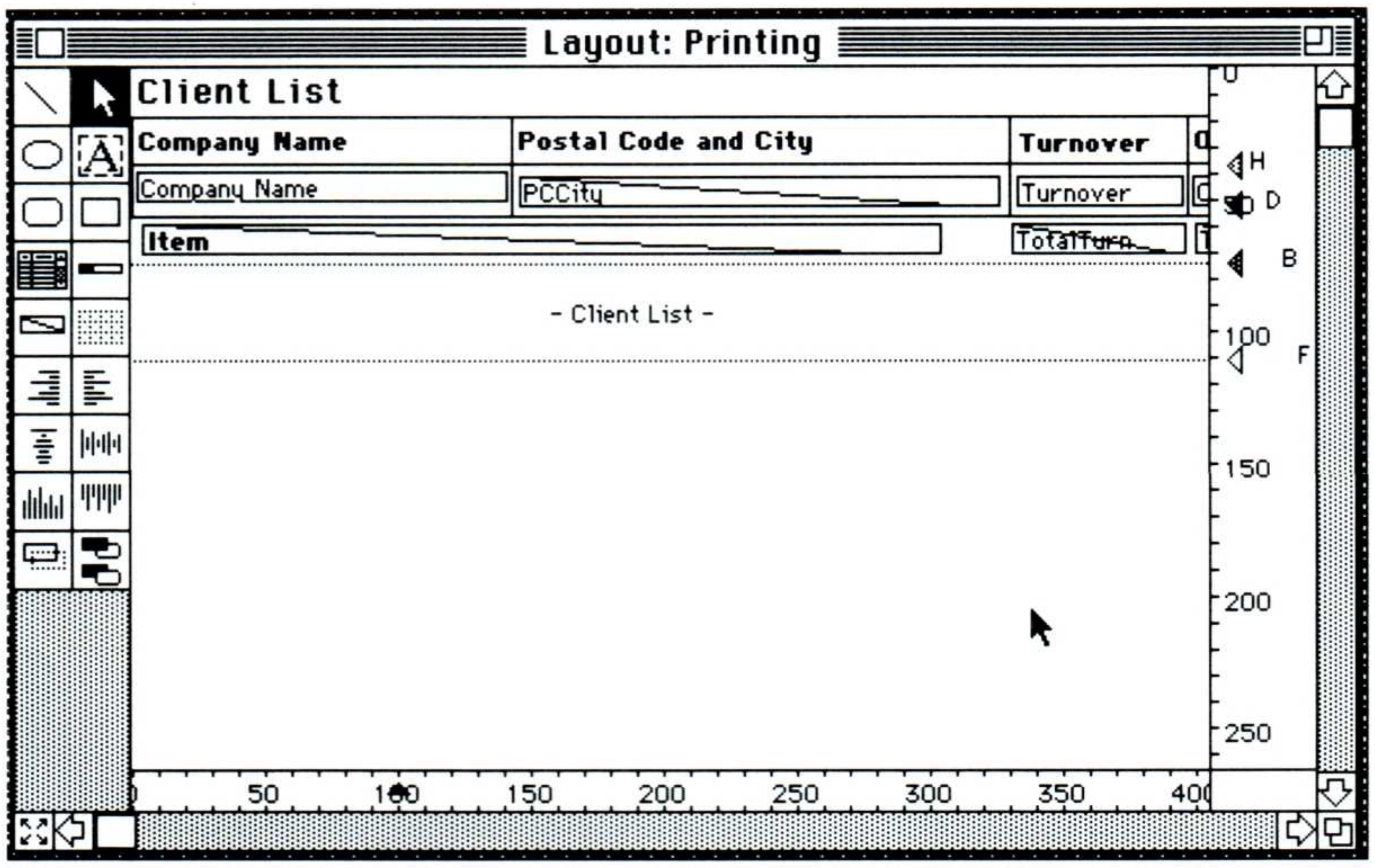


Figure A-4
Output layout named Printing

Note the location of the H, D, B, and F markers indicating the position of the layout Header, Detail, Break, and Footer.

In the Header area, specify the page number in the vPage variable. (The variable is placed in the right part of the layout and is not shown on the screen above.)

In the Detail area, place the Company Name field, the PCCity variable in which the Postal Code and City are concatenated, and the Turnover and Outstanding fields (the latter does not appear in the screen above).

In the Break area, insert the Item, TotalTurn and TotalOut variables. In the Item variable, you specify whether total is a subtotal or a grand total. Place in the TotalTurn and TotalOut variables the subtotals of the Turnover and Outstanding fields.

All these values are assigned through layout procedures (see below).

Here is the Customer List procedure:

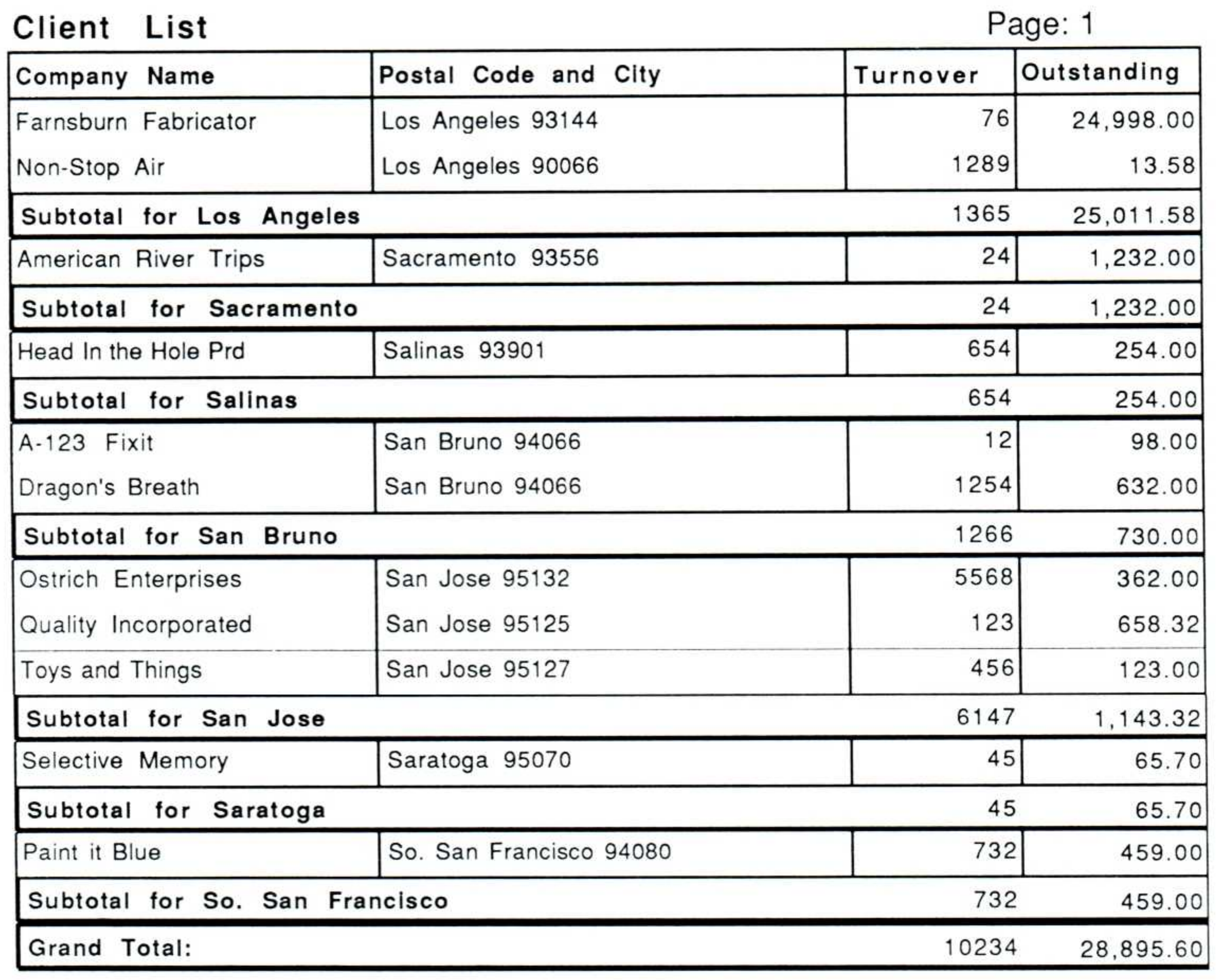
```
 `Sorts and prints a user selected part of the Customer List.
 `Uses the layout Printing, and its layout procedure
DEFAULT FILE([Clients])
SEARCH  `Let user select the record(s) to be printed any way he/she wants to.
 `Only do it if a) The OK button was pressed and b) There is at least one record to print.
If ((OK=1)&(Records in selection>0))
 SORT SELECTION([Clients]City;>;[Clients]Company Name;>)
 OUTPUT LAYOUT("Printing")
 PRINT SELECTION
End if
```

Here is the output layout procedure for the Printing layout:

```
 `Used to print client report sorted by city and client.

If (In header)  `About to print the page header?
 If (Before selection([Clients])  `Only happened once, at the very beginning
  n:=0  `Start page number.
 End if
 n:=n+1  `Increment page number once for each page.
 vPage:="Page: "+String(n)
End if  `In header
If (During)  `Assign City and Zip code to layout variable once for each record.
 PCCity:=City+" "+String(Postal Code;"00000")
End if
```

```
If (In break)   `About to print a break line?
   `Place the subtotal of the Turnover field in TotalTurn
 TotalTurn:=Subtotal(Turnover)
   `Place the subtotal of the Outstanding field in TotalOut
 TotalOut:=Subtotal(Outstanding)
 Case of
   : (Level=0)   `Level zero is at report end, Grand total.
     Item:="Grand Total: "
   : (Level=1)   `Level one is when the City changes.
       `(See the sort done in the global calling routine.)
     Item:="Subtotal for "+[Clients]City
 End case   `Level
End if   `In break
```

Figure A-5 shows what the resulting printed page should look like.



Client List   Page: 1

Company Name	Postal Code and City	Turnover	Outstanding
Farnsburn Fabricator	Los Angeles 93144	76	24,998.00
Non-Stop Air	Los Angeles 90066	1289	13.58
Subtotal for Los Angeles		1365	25,011.58
American River Trips	Sacramento 93556	24	1,232.00
Subtotal for Sacramento		24	1,232.00
Head In the Hole Prd	Salinas 93901	654	254.00
Subtotal for Salinas		654	254.00
A-123 Fixit	San Bruno 94066	12	98.00
Dragon's Breath	San Bruno 94066	1254	632.00
Subtotal for San Bruno		1266	730.00
Ostrich Enterprises	San Jose 95132	5568	362.00
Quality Incorporated	San Jose 95125	123	658.32
Toys and Things	San Jose 95127	456	123.00
Subtotal for San Jose		6147	1,143.32
Selective Memory	Saratoga 95070	45	65.70
Subtotal for Saratoga		45	65.70
Paint it Blue	So. San Francisco 94080	732	459.00
Subtotal for So. San Francisco		732	459.00
Grand Total:		10234	28,895.60

Figure A-5
Printed output from Printing layout

Printing labels from subfiles

Unlike PRINT SELECTION, PRINT LABEL does not work with subfile. To print labels with addresses saved in a subfile, use the PRINT SELECTION command. Figure A-6 shows a typical file structure with subfile addresses.

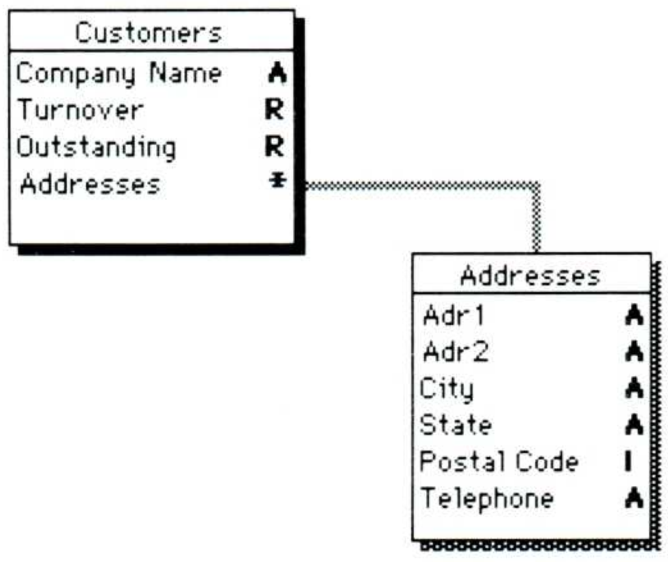


Figure A-6
File with address data in a subfile

Create a Multi-line layout for the subfile to print labels using the [Customers]Addresses subfile field, as shown in Figure A-7.

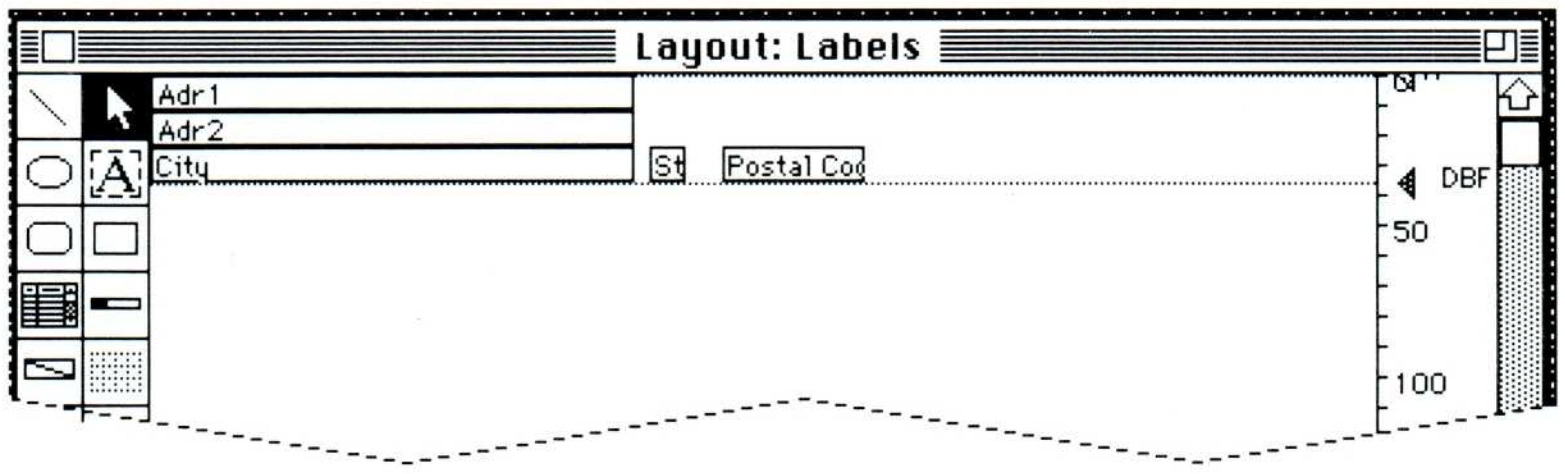


Figure A-7
Multi-line layout for printing address data

Create a layout for the Customers file in which you specify the Addresses subfile field in an included subfile area. See Figure A-8.

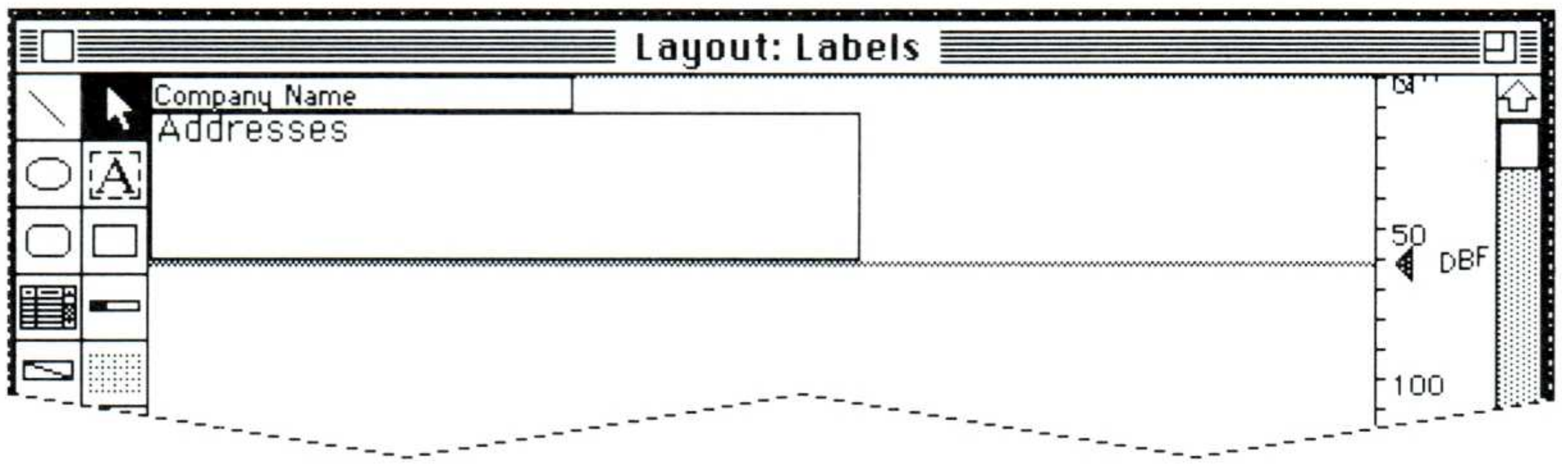
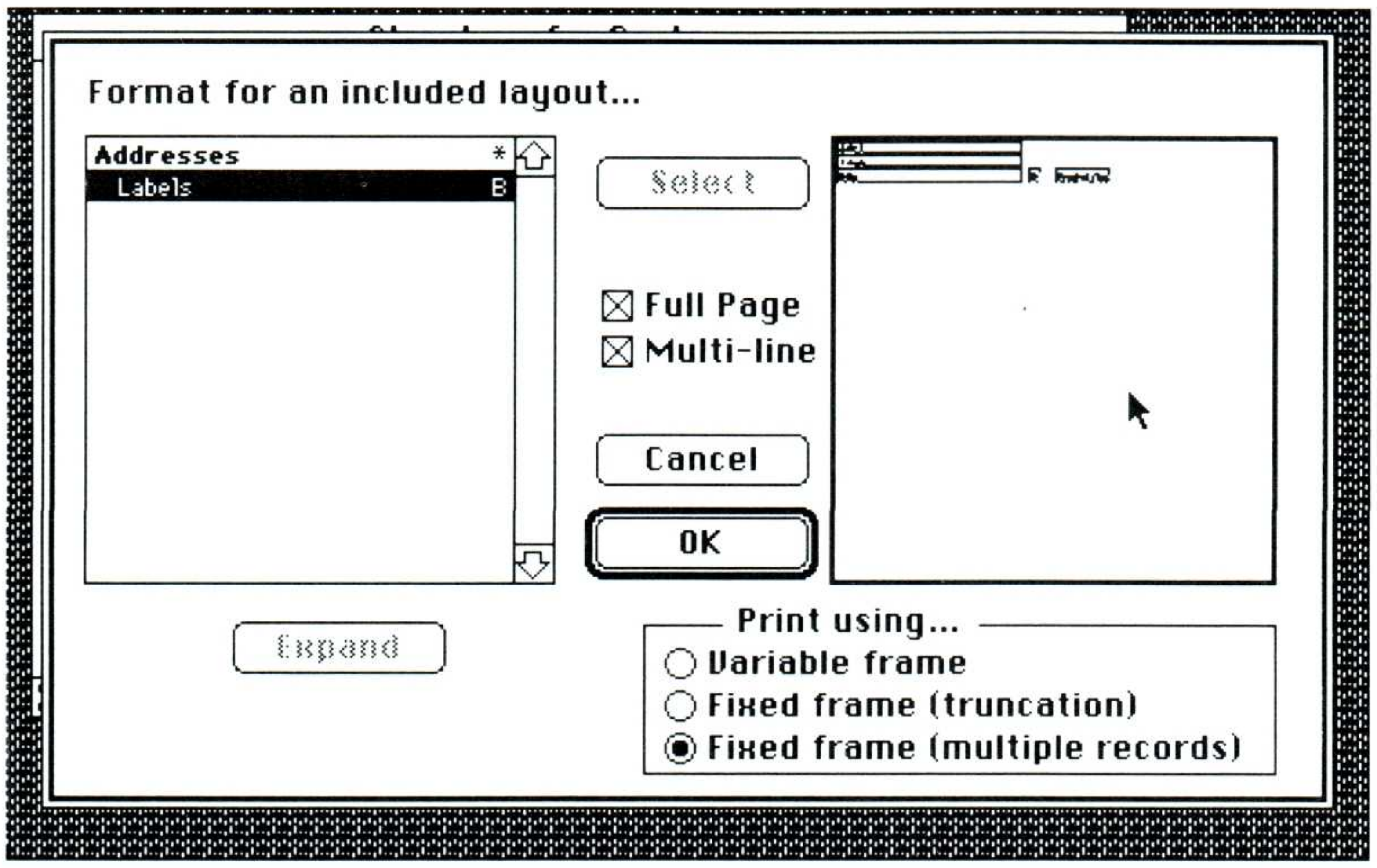


Figure A-8
Mailing label layout with included subfile area

Draw a subfile area slightly taller (by one or two points) than the detail area. To do so, draw a box in the subfile layout window over the subfile fields which covers up exactly that area. Cut the box you drew and paste it in the record layout. Now, you just need to draw a subfile area having the same size as the box you pasted to make sure that it is the same size as the subfile layout. Eventually, increase the height of that area by one or two points. Be sure to specify the area as Fixed frame (multiple records).

Figure A-9 shows the Format for an included layout dialog box.


Format for an included layout...

Addresses *
Labels B

Select

☒ Full Page
☒ Multi-line

Cancel
OK

Expand

Print using...
○ Variable frame
○ Fixed frame (truncation)
● Fixed frame (multiple records)

Figure A-9
Format for an included layout dialog box

To print labels, call the PRINT SELECTION command. In the User environment, choose the Print command from the File menu instead of the Print Labels command. When 4th Dimension prints that layout, it will print one subrecord in the area, because the number of subrecords that can be printed in that area is limited to one. Nevertheless, each record will be printed as many time as it contains addresses, because you selected the Fixed frame (multiple records). That way all the labels you need will be printed. The disadvantage, when printing subrecords as labels, is that you cannot have more than one column of labels per page.

Printing with PRINT LAYOUT

Figure A-10 shows a Products file with a subfile [Products]Sales that keeps the quantity sold of each item by month and year of sale.

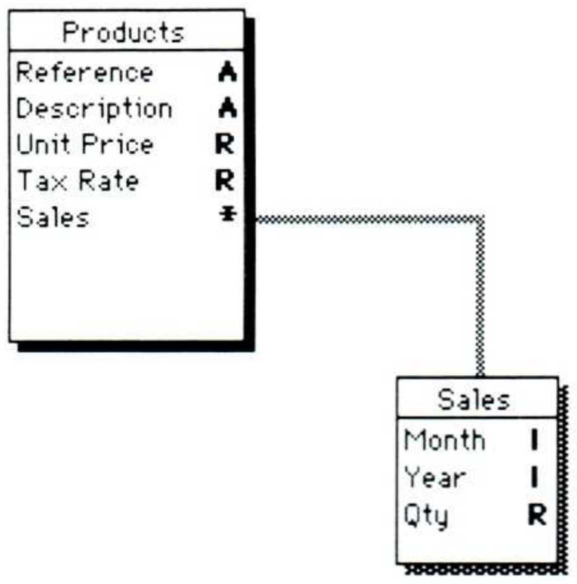
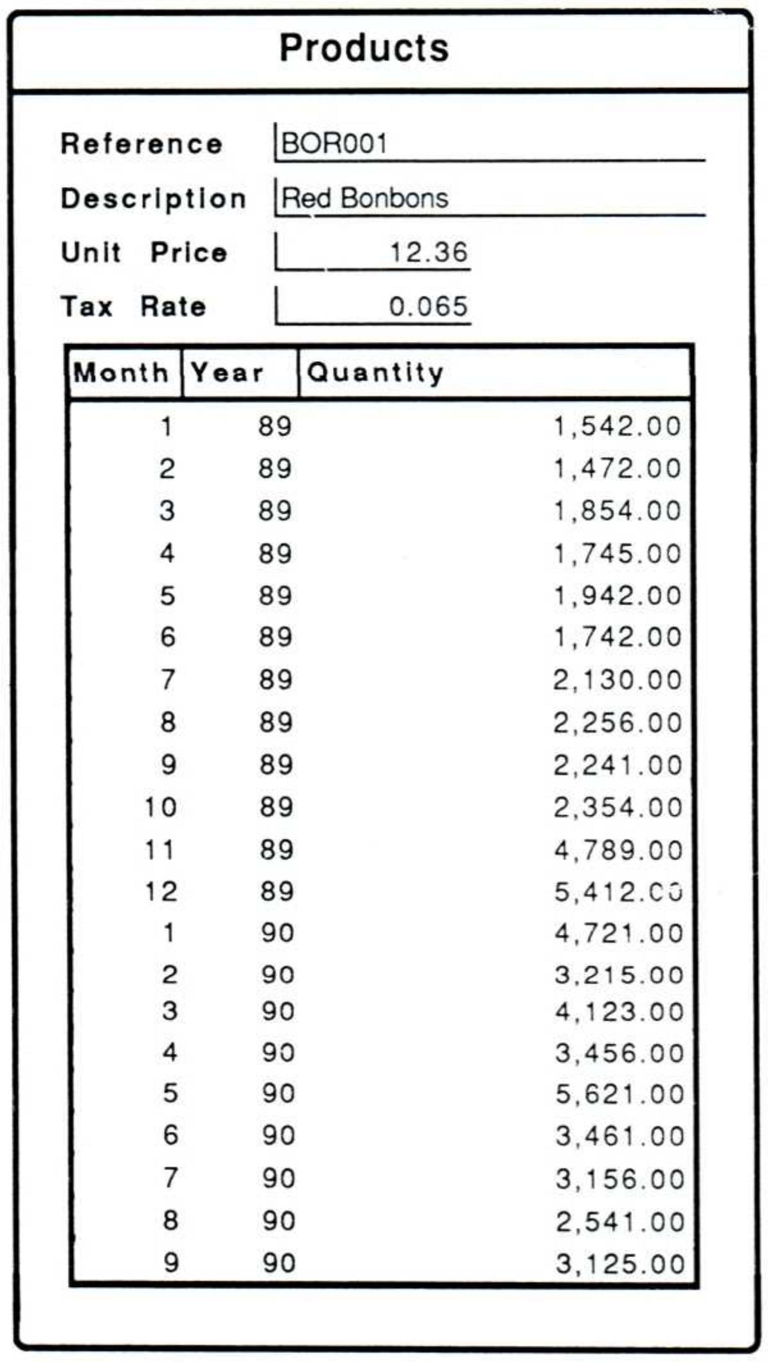


Figure A-10
Products file with Sales subfile

Figure A-11 shows a record with its included subfile area.



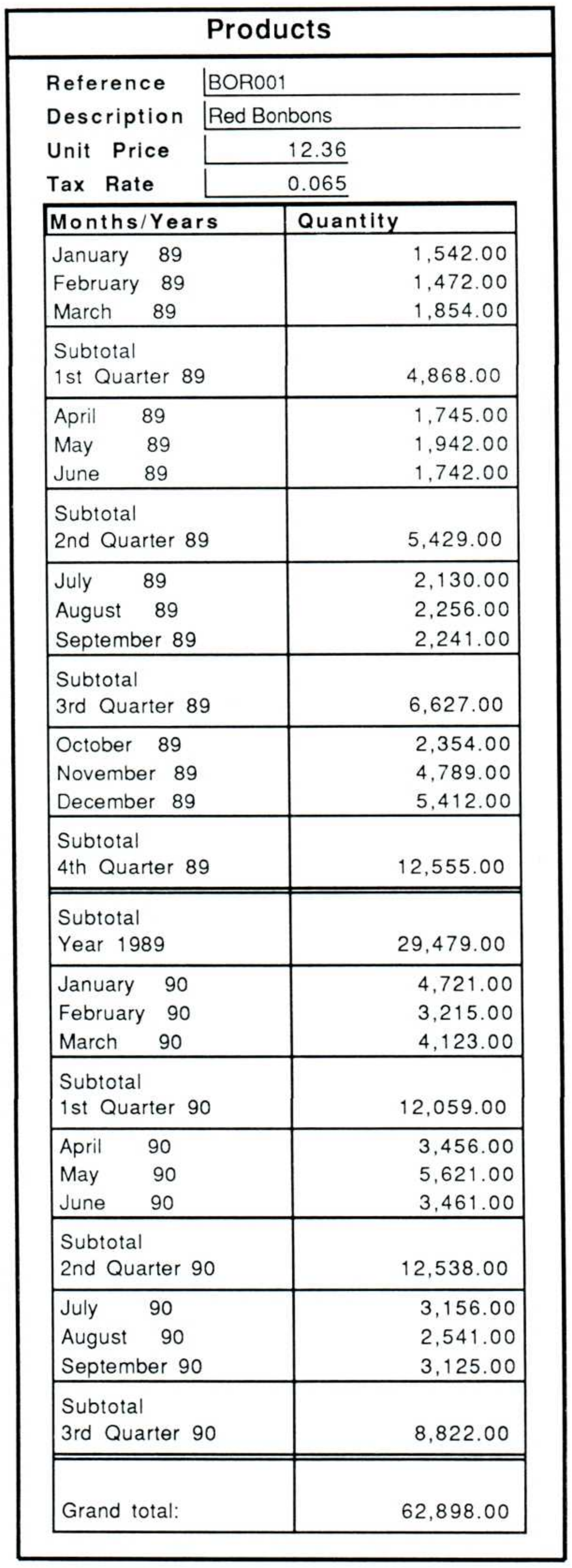
Products

Reference	BOR001
Description	Red Bonbons
Unit Price	12.36
Tax Rate	0.065

Month	Year	Quantity
1	89	1,542.00
2	89	1,472.00
3	89	1,854.00
4	89	1,745.00
5	89	1,942.00
6	89	1,742.00
7	89	2,130.00
8	89	2,256.00
9	89	2,241.00
10	89	2,354.00
11	89	4,789.00
12	89	5,412.00
1	90	4,721.00
2	90	3,215.00
3	90	4,123.00
4	90	3,456.00
5	90	5,621.00
6	90	3,461.00
7	90	3,156.00
8	90	2,541.00
9	90	3,125.00

Figure A-11
Products file record

To print the record above with a subtotal for the subfile field named [Products]Sales'Qty, use the PRINT SETTINGS, PRINT LAYOUT, and FORM FEED commands. Figure A-12 shows you the results.



Products

Reference	BOR001
Description	Red Bonbons
Unit Price	12.36
Tax Rate	0.065

Months/Years	Quantity
January 89	1,542.00
February 89	1,472.00
March 89	1,854.00
Subtotal 1st Quarter 89	4,868.00
April 89	1,745.00
May 89	1,942.00
June 89	1,742.00
Subtotal 2nd Quarter 89	5,429.00
July 89	2,130.00
August 89	2,256.00
September 89	2,241.00
Subtotal 3rd Quarter 89	6,627.00
October 89	2,354.00
November 89	4,789.00
December 89	5,412.00
Subtotal 4th Quarter 89	12,555.00
Subtotal Year 1989	29,479.00
January 90	4,721.00
February 90	3,215.00
March 90	4,123.00
Subtotal 1st Quarter 90	12,059.00
April 90	3,456.00
May 90	5,621.00
June 90	3,461.00
Subtotal 2nd Quarter 90	12,538.00
July 90	3,156.00
August 90	2,541.00
September 90	3,125.00
Subtotal 3rd Quarter 90	8,822.00
Grand total:	62,898.00

Figure A-12
Output showing quarterly and annual sales

To create an output like that shown in Figure A-12, you must create four layouts (shown in Figures A-13 through A-16) and print them one under another using PRINT LAYOUT.

To print the record header detail and the subfile header, create the record detail layout (pl.ProdResult.H) shown in Figure A-13.

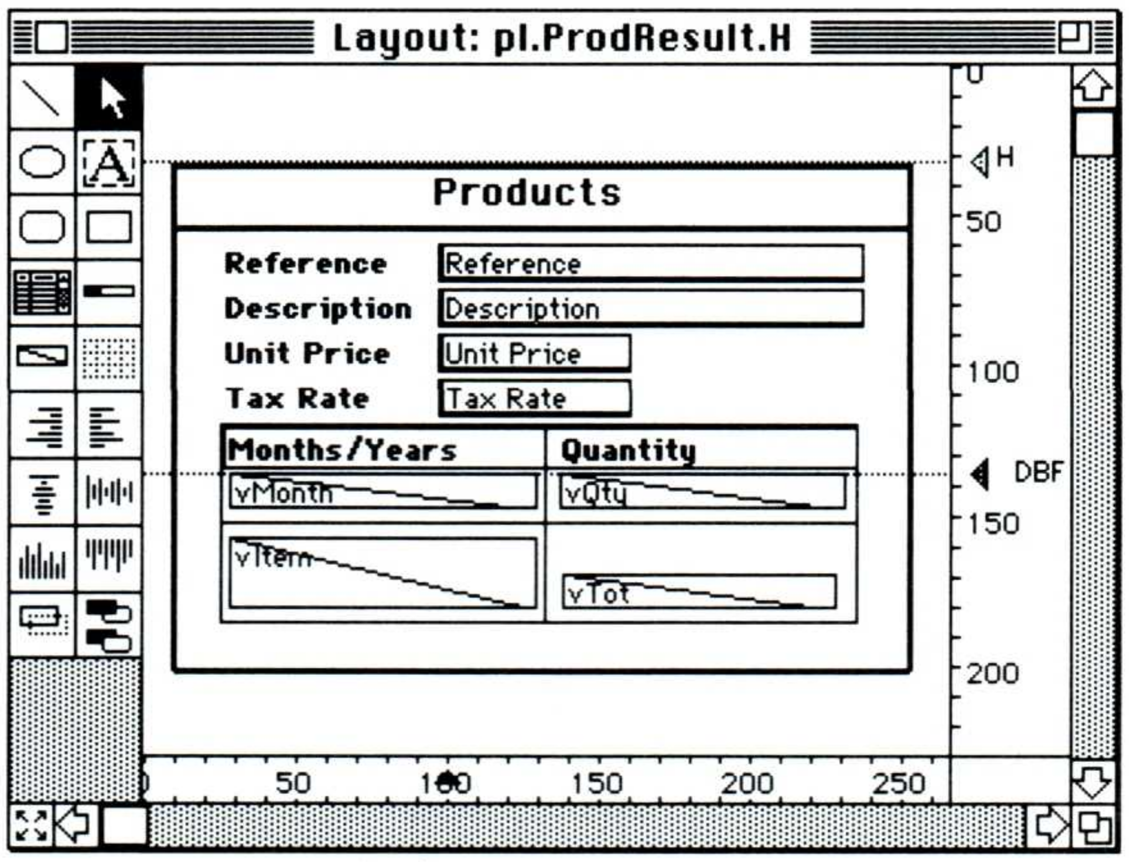


Figure A-13
Record detail layout

To print the detail of every subrecord, create the subrecord detail layout (pl.ProdRes.Sub) shown in Figure A-14.

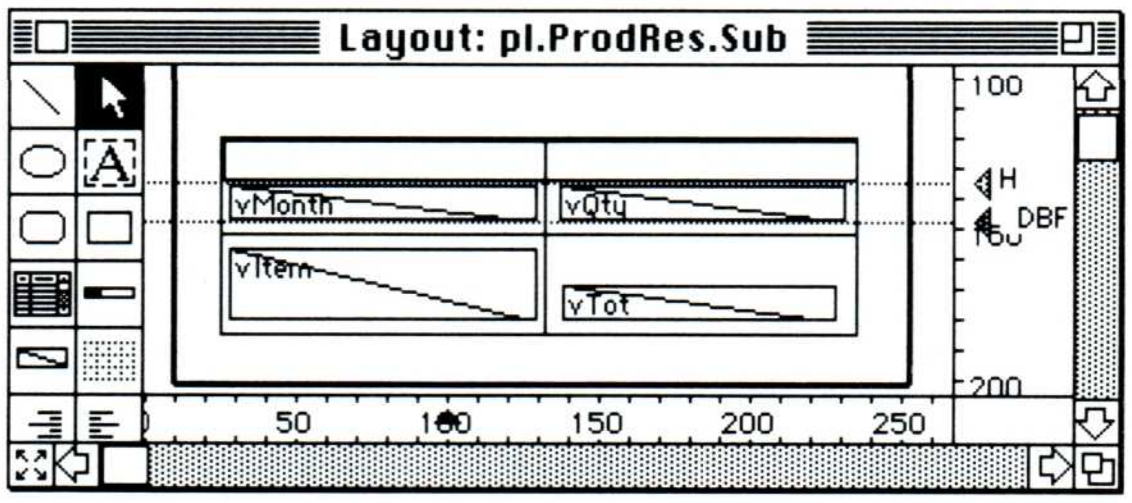


Figure A-14
Subrecord detail layout

To print subtotals and the grand total, create the subtotal layout (pl.ProdResult.B) shown in Figure A-15.

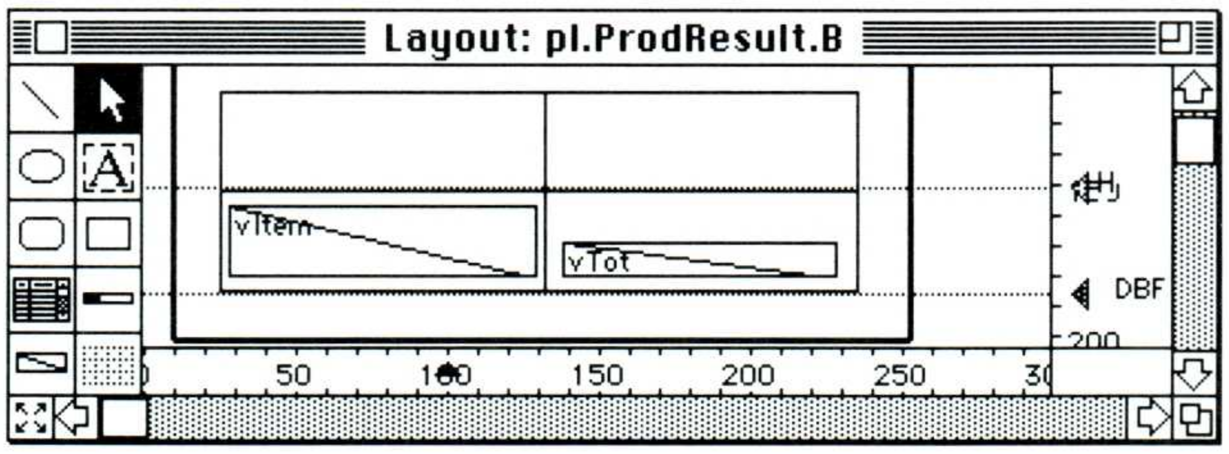


Figure A-15
Subtotal layout

As mentioned earlier, the PRINT LAYOUT command only prints the layout detail line. Furthermore, the layout must belong to a file and not to a subfile. The three layouts above are related to the same file named Products and not to the subfile named [Products]Sales. The "absolute" position of the layout detail is not that important because PRINT LAYOUT only considers the height of the layout detail, i.e. the interval between the H and D markers. Objects located elsewhere in the layout are ignored and will not be printed.

❖ *Note:* To align the various layouts consistantly, use the Cut, Copy, and Paste commands from the Edit menu.

To print the end of the record detail, create the end record layout (pl.ProdResult.F) shown in Figure A-16.

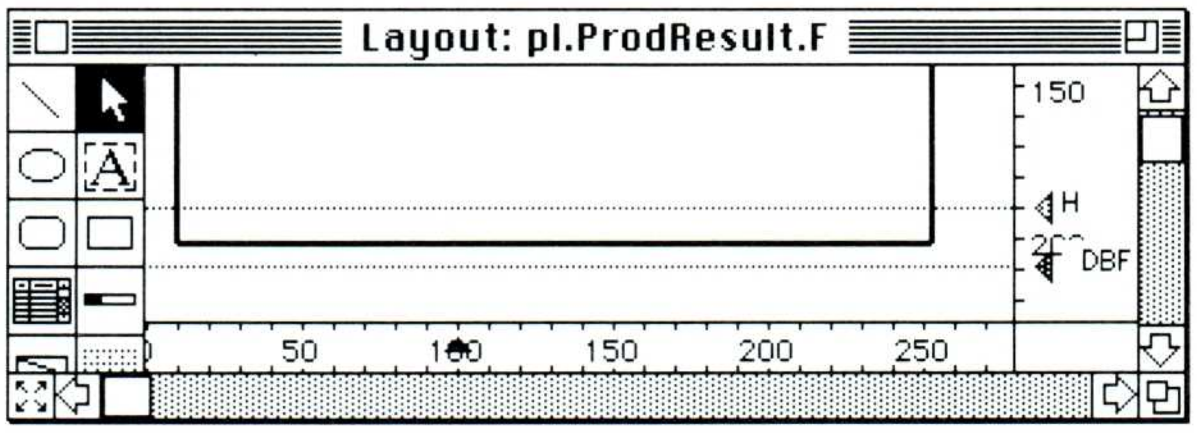


Figure A-16
End record layout

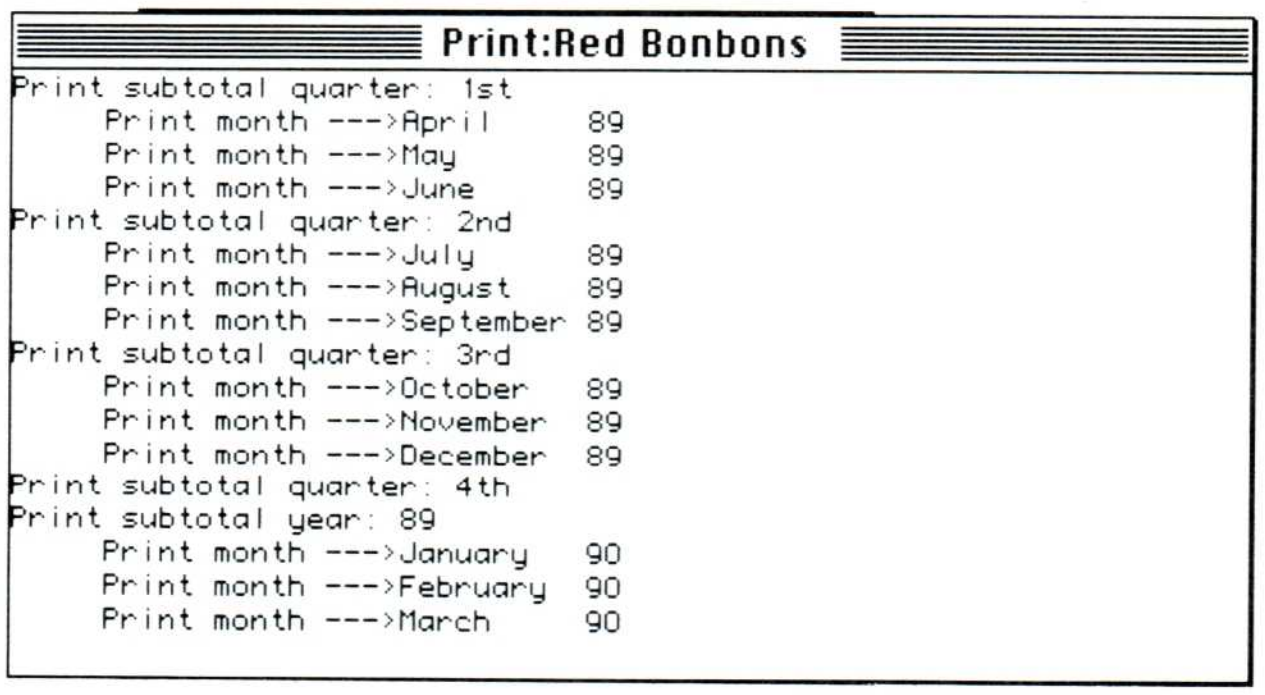
Write the following procedure named "Product Results" to print records:

```
  `This procedure prints a report using the PRINT LAYOUT command

DEFAULT FILE([Products])
  `Ask the user the reference of the product to be printed.
S:=Request("Specify the product reference:")

  `If the user validates the request and supplies a reference...
If ((OK=1)&(S#""))
    `...search the record.
  SEARCH([Products]Reference=S)
    `If the record is found:
  If (Records in selection#0)
      `Dislay on the screen the two standard print dialogs.
    PRINT SETTINGS
    If (OK=1)
        `If the user validates, calculate the center of the screen to open the window.
      $H:=Screen width/2
      $V:=Screen height/2
        `Place the Return character in the $vCR variable.
      $vCR:=Char(13)
        `Place 5 Space characters in the $v5SP variable.
      $v5SP:=" "*5
        `Display custom window by placing the product name in the title bar.
      OPEN WINDOW($H-200;$V-100;$H+200;$V+100;0;"Print: "+[Products]Description+" ")
        `Send a message to that window
      MESSAGE($vCR+$vCR+$v5SP+"Print record detail"+$vCR)
        `Print the header and record detail.
      PRINT LAYOUT([Products];"pl.ProdResult.H")
        `Store the quarterly subtotals in QuarterTot variable and the yearly subtotals in YearTot variable.
      QuarterTot:=0
      YearTot:=0
        `Select all the Sales subrecords of the product record.
      ALL SUBRECORDS([Products]Sales)
        `Sort the subrecords by year and month.
      SORT SUBSELECTION([Products]Sales;[Products]Sales'Year;>;[Products]Sales'Month;>)

        `Count the subrecords with the i variable. Initialize i variable to 1.
      i:=1
        `Count the quarters with the q variable. Initialize q variable to 1.
      q:=1
```

```
While (Not(End subselection([Products]Sales)))
   `While we are within the subrecord selection.
   `Calculate the vMonth variable in the "pl.ProdResult.D" layout
 vMonth:=Month name ([Products]Sales'Month)+" "+String([Products]Sales'Year)
   `The global user procedure "Month name" returns the name of the month based on the month number.
 MESSAGE($v5SP+"Print month --->"+vMonth+$vCR)
   `Calculate the vQty variable in the Subrec. Detail layout
 vQty:=[Products]Sales'Qty
   `Print the subrecord.
 PRINT LAYOUT([Products];"pl.ProdRes.Sub")
   `Add the subrecord quantity to QuarterTot and YearTot.
 QuarterTot:=QuarterTot+[Products]Sales'Qty
 YearTot:=YearTot+[Products]Sales'Qty
 If (Mod(i;3)=0)  `Quarterly total?
        `If "i" can be divided by 3, we are dealing with a quarter, so we print the quarterly total.
     Case of
     :(q=1)
       vQ:="1st"
     :(q=2)
       vQ:="2nd"
     : (q=3)
       vQ:="3rd"
     :(q=4)
       vQ:="4th"
     End case
        `Calculate the vItem variable contained in the "SubTot" layout.
     vItem:="Subtotal"+$vCR+vQ+" Quarter "+String([Products]Sales'Year)
        `Assign vTot the month accumulated total from QuarterTot
     vTot:=QuarterTot
        `Inform the user.
     MESSAGE("Print subtotal quarter: "+vQ+$vCR)
        `Print the subtotal.
     PRINT LAYOUT([Products];"pl.ProdResult.B")
        `Increment the quarter count
     q:=q+1
     If (q=5)
        `If it's gone beyond 4th Quarter, set the count to 1.
        q:=1
     End if
        `Reset the quarterly subtotal.
     QuarterTot:=0
  End if  `Quarterly total
  If (Mod(i;12)=0)  `Yearly total?
        `If "i" can be divided by 12, we are dealing with a year end and must then print the yearly subtotal.
        `Calculate the vItem variable.
   vItem:="Subtotal"+$vCR+"Year 19"+String([Products]Sales'Year)
        `Place in vTot the accumulated total of the year in YearTot.
   vTot:=YearTot
   YearTot:=0  `Reset yeat total
        `Inform the user.
```

```
          MESSAGE("Print subtotal year: "+String([Products]Sales'Year)+$vCR)
              `Print the yearly subtotal.
          PRINT LAYOUT([Products];"pl.ProdResult.B")
              `Reset the yearly subtotal.
          vYearTot:=0
        End if   `Yearly total
            `Increment the subrecord count.
        i:=i+1
            `Go to the next subrecord.
        NEXT SUBRECORD([Products]Sales)
      End while   `Subrecords remain

        `Calculate the grand totals and print them.
      vItem:=$vCR+"Grand total: "
      vTot:=Sum([Products]Sales'Qty)
        `Print the grand total and footer.
      PRINT LAYOUT([Products];"pl.ProdResult.B")
      PRINT LAYOUT([Products];"pl.ProdResult.F")
        `Inform the user.
      MESSAGE($v5SP+"Start printing..."+$vCR)
        `Call FORM FEED to print the page
      FORM FEED
        `Close the custom window.
      CLOSE WINDOW
     End if
   End if
 End if
```

To execute the procedure in the User environment, choose Execute Procedure from the Special menu. While printing the record, the procedure lets you view the various steps on your screen, as shown in Figure A-17.

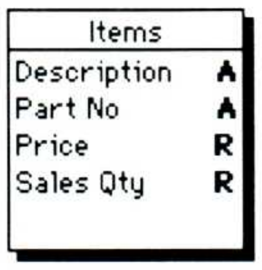


```
Print subtotal quarter: 1st
     Print month --->April     89
     Print month --->May       89
     Print month --->June      89
Print subtotal quarter: 2nd
     Print month --->July      89
     Print month --->August    89
     Print month --->September 89
Print subtotal quarter: 3rd
     Print month --->October   89
     Print month --->November  89
     Print month --->December  89
Print subtotal quarter: 4th
Print subtotal year: 89
     Print month --->January   90
     Print month --->February  90
     Print month --->March     90
```

Figure A-17
Screen view of report progress

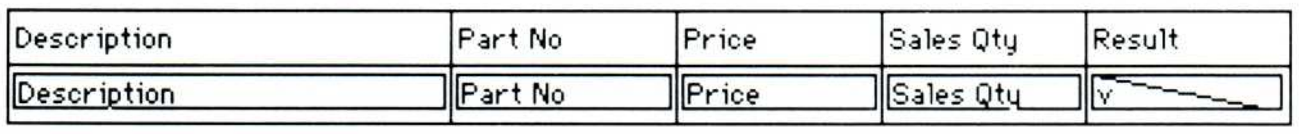
Using the FONT STYLE command

The FONT STYLE command is useful for highlighting output. For example, you might want to use different type styles to emphasize a comment on sales of each product in your inventory. This example uses the file shown in Figure A-18 and the output layout shown in Figure A-19.



Items	
Description	A
Part No	A
Price	R
Sales Qty	R

Figure A-18
Items file



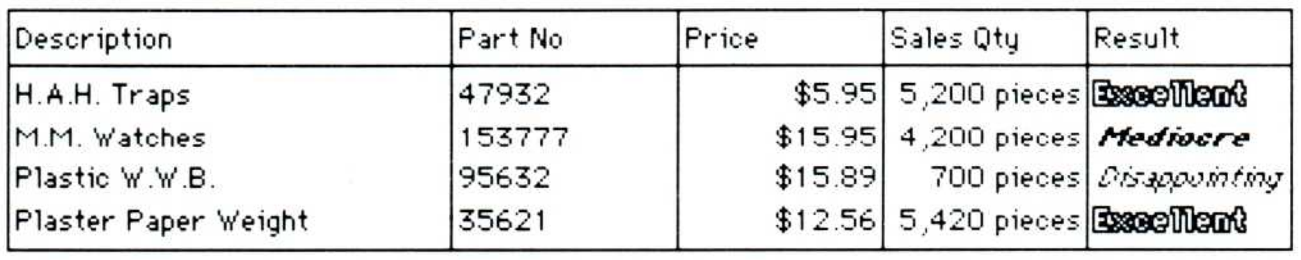
Description	Part No	Price	Sales Qty	Result
Description	Part No	Price	Sales Qty	v

Figure A-19
Items file output layout

The following layout procedure does the job:

```
Case of    `Determine quality of sales, based on quantity.
  : (Sales Qty<1000)
    $Style:=2
    v:="Disappointing"
  : ((Sales Qty>=1000)&(Sales Qty<5000))
    $Style:=1+2
    v:="Mediocre"
  : (Sales Qty>=5000)
    $Style:=8
    v:="Excellent"
End case
  `Change font style to reflect
FONT STYLE(v;$Style)
```

The current selection will look like that in Figure A-20 when displayed or printed.



Description	Part No	Price	Sales Qty	Result
H.A.H. Traps	47932	$5.95	5,200 pieces	Excellent
M.M. Watches	153777	$15.95	4,200 pieces	***Mediocre***
Plastic W.W.B.	95632	$15.89	700 pieces	*Disappointing*
Plaster Paper Weight	35621	$12.56	5,420 pieces	Excellent

Figure A-20
Selection displayed with FONT STYLE command

Subfiles

When working on several nested subfiles, using ALL SUBRECORDS selects all the subrecords of *n*-level subfile in the current *n–1*-level subrecord. This example uses the Factories database shown in Figure A-21.

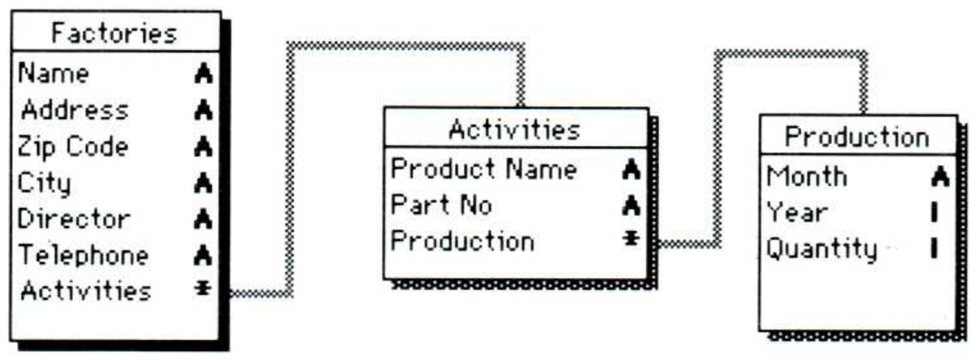
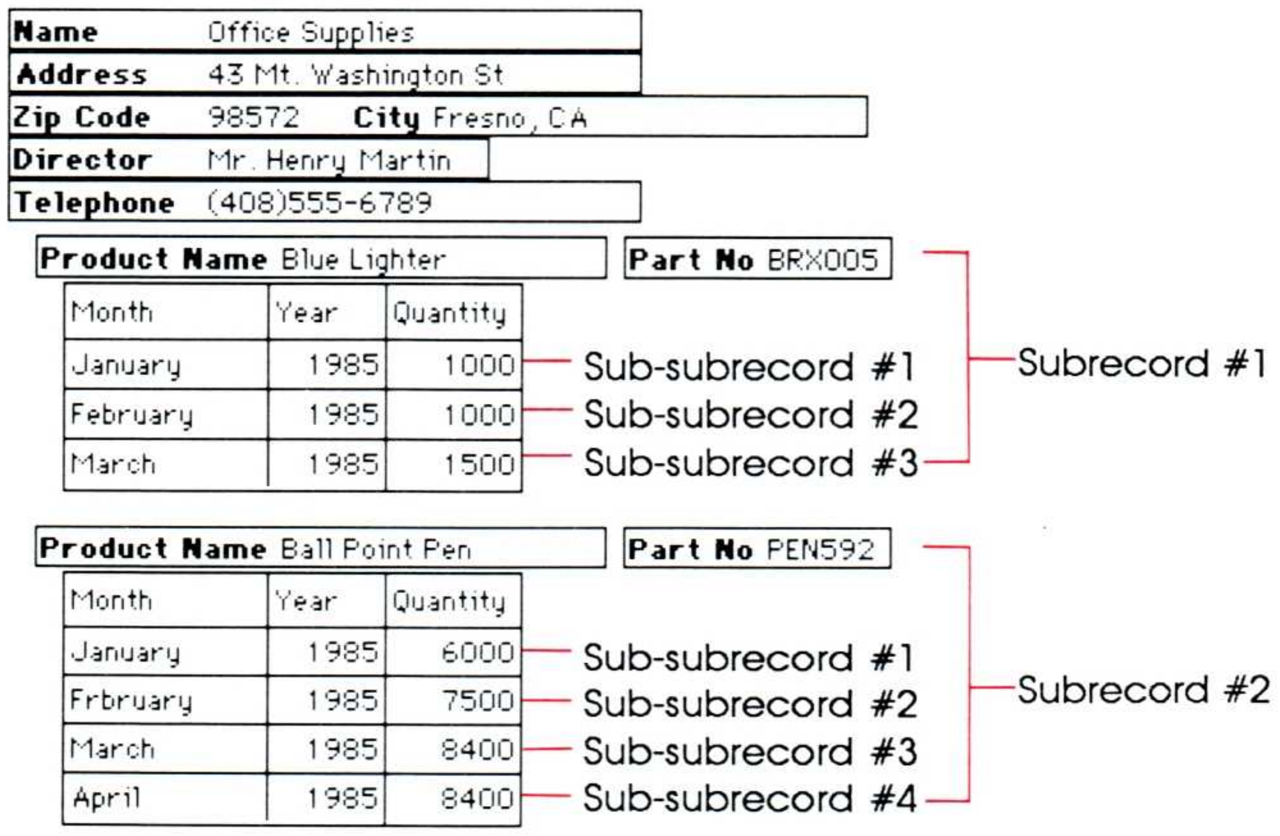


Figure A-21
Factories database structure

For every record of the Factories file you may have a set of [Factories]Activities subrecords. Each subrecord can in turn contain a set of [Factories]Activites'Production subrecords. Suppose the current Factories file record contains the data shown in Figure A-22.



Name	Office Supplies
Address	43 Mt. Washington St
Zip Code	98572 **City** Fresno, CA
Director	Mr. Henry Martin
Telephone	(408)555-6789

Product Name Blue Lighter **Part No** BRX005 — Subrecord #1

Month	Year	Quantity	
January	1985	1000	Sub-subrecord #1
February	1985	1000	Sub-subrecord #2
March	1985	1500	Sub-subrecord #3

Product Name Ball Point Pen **Part No** PEN592 — Subrecord #2

Month	Year	Quantity	
January	1985	6000	Sub-subrecord #1
Frbruary	1985	7500	Sub-subrecord #2
March	1985	8400	Sub-subrecord #3
April	1985	8400	Sub-subrecord #4

Figure A-22
Current Factories record

This record has two subrecords in the [Factories]Activities subfile. After executing

```
ALL SUBRECORDS([Factories]Activities)
```

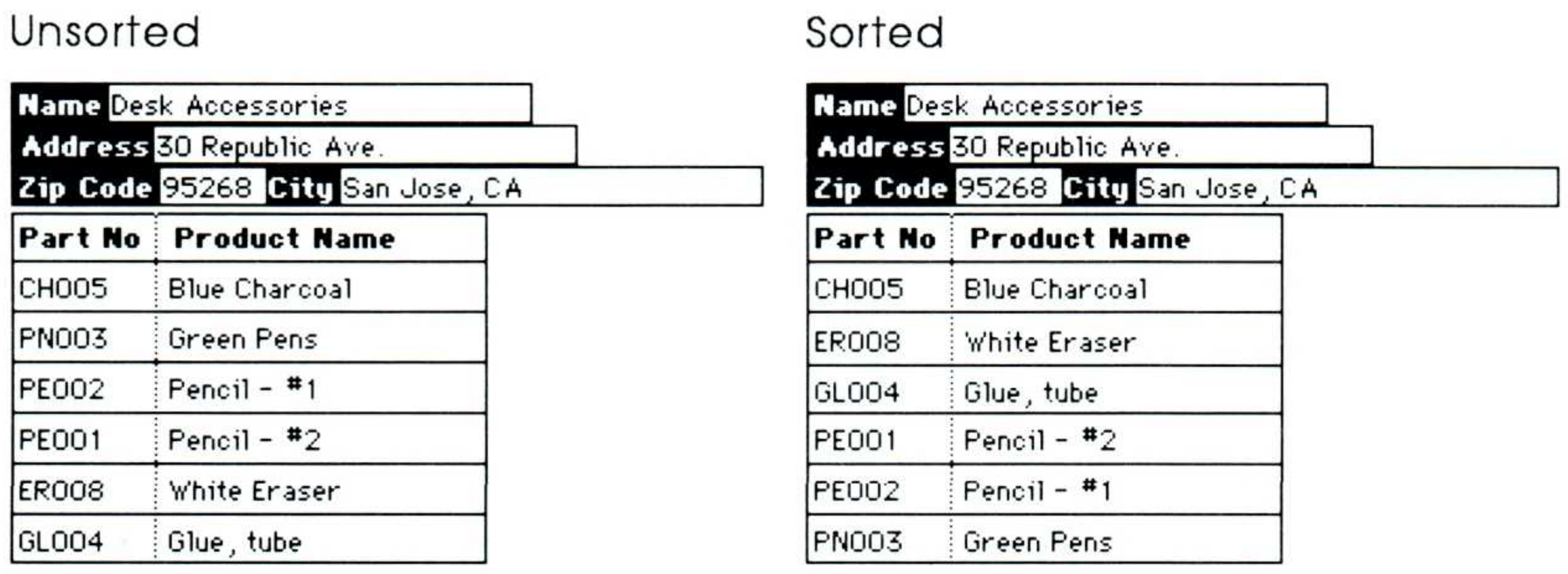
both subrecords are part of the subselection and the Blue Lighter subrecord becomes the current [Factories]Activities subrecord. The following statement

```
ALL SUBRECORDS([Factories]Activities'Production)
```

applies to the current [Factories]Activities subrecord, Blue Lighter. The three subrecords January, February and March are then the [Factories]Activities'Production subselection, while the subrecord for January becomes the current [Factories]Activities'Production subrecord.

In Figure A-23, the record on the left shows output when no layout procedure is present. By writing the following layout procedure for the parent layout, you get the display for the record shown on the right.

```
If (Before)
   ALL SUBRECORDS (Activities)
   SORT SUBSELECTION (Activities;Activities'Part No; >)
End if
```



Unsorted

Name	Desk Accessories
Address	30 Republic Ave.
Zip Code	95268 City San Jose, CA

Part No	Product Name
CH005	Blue Charcoal
PN003	Green Pens
PE002	Pencil - #1
PE001	Pencil - #2
ER008	White Eraser
GL004	Glue, tube

Sorted

Name	Desk Accessories
Address	30 Republic Ave.
Zip Code	95268 City San Jose, CA

Part No	Product Name
CH005	Blue Charcoal
ER008	White Eraser
GL004	Glue, tube
PE001	Pencil - #2
PE002	Pencil - #1
PN003	Green Pens

Figure A-23
Records displayed with and without a layout procedure

If invoking the SORT SUBSELECTION command when the record layout is already displayed, you must explicitly call the REDRAW command to force the subfile area to display subrecords in the appropriate sort order.

To put all product names in the Activities subselection of the Factories file current record in uppercase, use the following procedure as the layout procedure for Factories. Changes will be saved only when the user validates the entry.

```
If (After)
 ALL SUBRECORDS(Activities)
 APPLY TO SUBSELECTION(Activities;
   [Factories]Activities'Product Name:=Uppercase([Factories]Activities'Product Name))
End if
```

Using the database in Figure A-10, take a look at the output layout for the Products file shown in Figure A-24.

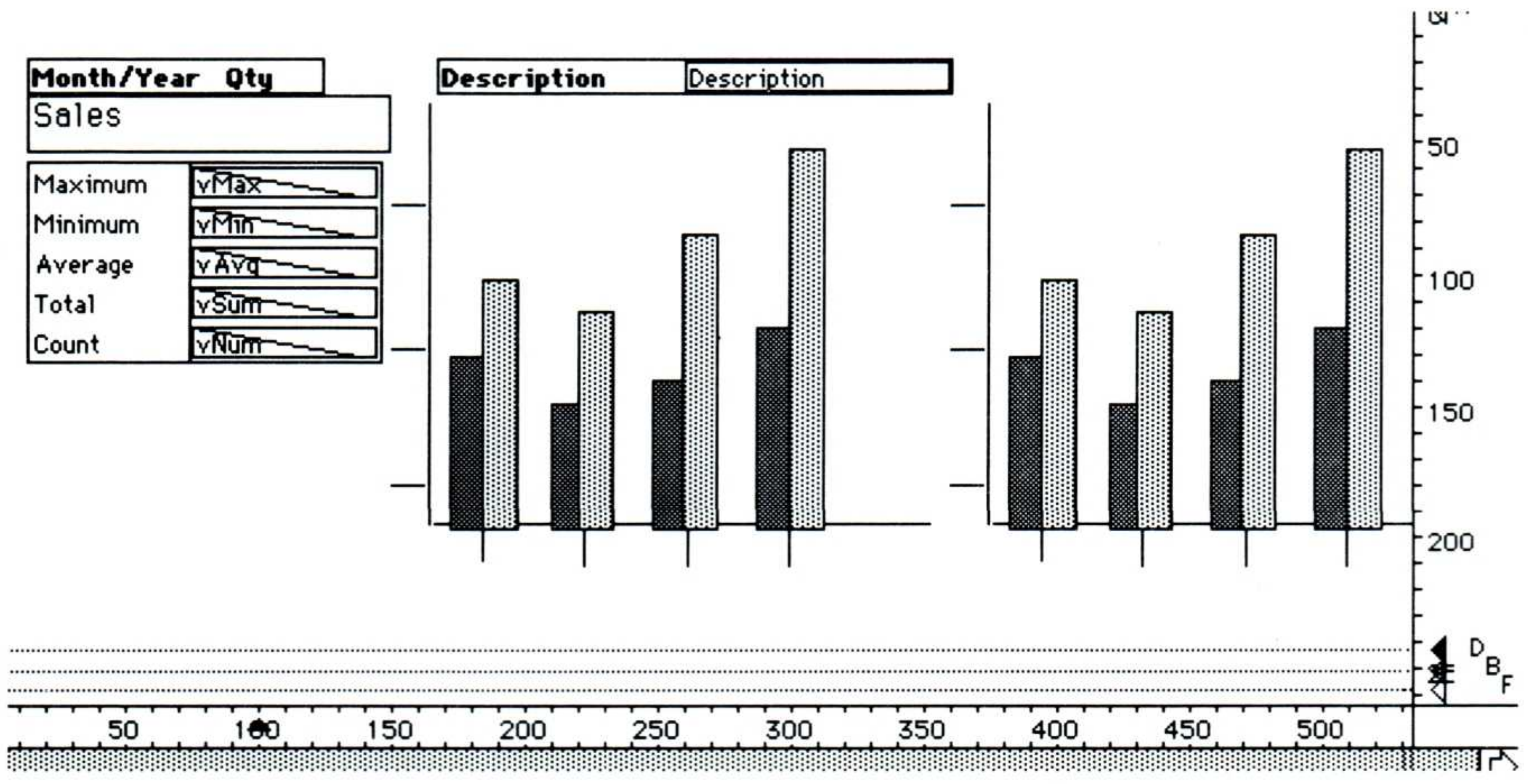


Figure A-24
Output layout for Products file

In the upper left corner of the layout, you can see the subfile named Sales. The vMax, vMin, vAvg, vSum, and vNum variables are located right below it. In these variables you place the maximum value, minimum value, average value, total value, and count for all the values contained in the Sales'Qty field for all the subrecords belonging to the current record. Both graph areas, located in the right part of the layout, contain a graph drawn on the basis of the values contained in subrecords, with the Sales'Year field as the x-axis and the Sales'Qty field as the y-axis. The left graph area (G1) displays bars. The right graph area (G2) displays shaded areas. The following layout procedure does the graphing:

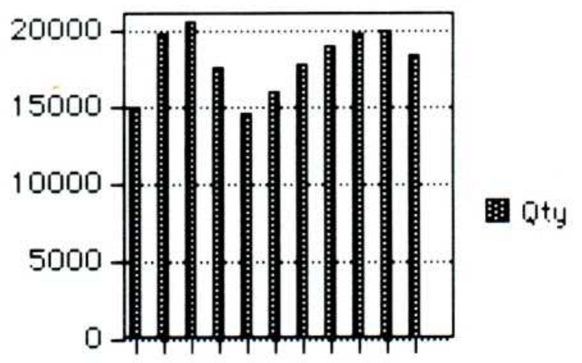
```
If(Before)
   ALL SUBRECORDS(Sales)
   vMax := Max(Sales'Qty)
   vMin := Min(Sales'Qty)
   vAvg := Average(Sales'Qty)
   vSum := Sum(Sales'Qty)
   vNum := Records in subselection(Sales'Qty)
   GRAPH(G1;1;Sales'Year;Sales'Qty)
   GRAPH(G2;4;Sales'Year;Sales'Qty)
End if
```

If the layout is used for printing, each record of the current selection will look like Figure A-25.

Month/Year	Qty
1 / 88	15000
10 / 88	19700
11 / 88	20500
2 / 88	17500
3 / 88	14500
4 / 88	16000
5 / 88	17800
6 / 88	19000
7 / 88	19800
8 / 88	20000
9 / 88	18400

Description Chalk

Maximum	20500
Minimum	14500
Average	18018.18181
Total	198200
Count	11



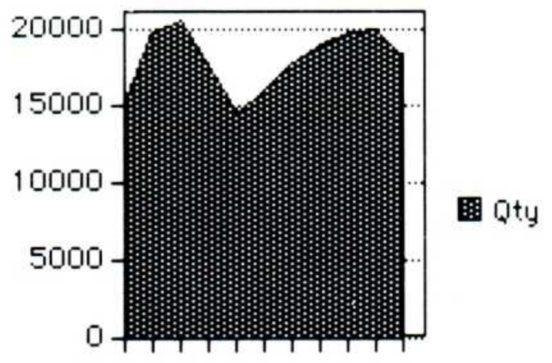


Figure A-25
Record printout

Graphs

This section gives additional information on graphs in layouts. You can draw graphs of field values by using the GRAPH FILE file command. In the same way, you can draw graphs of subfield values with the GRAPH command. In the first case, 4th Dimension generates the graph on the screen or on the printer. In the latter case, you generate the graph by creating a graph area in a layout and writing a layout procedure for the layout. You can then display or print the graph when displaying or printing the layout.

The GRAPH command can apply to variables. In this case, here is the proper syntax:

GRAPH (*var;posintexpr;strvarX;numvarY* {;*})

The *var* argument is a variable indicating the graph area in the layout. The *posintexpr* argument is a numeric expression that determines the graph type. The *strvarX* argument must be alphanumeric, because it labels the x-axis. The values of this table are used in the x-axis. You can have as many as eight *numvarY* arguments. These are then numeric values plotted against the y-axis.

Figure A-26 shows a file containing data about departments in a business.

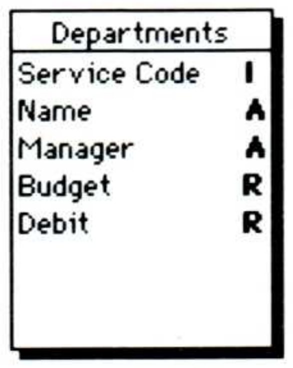

Departments
Service Code I
Name A
Manager A
Budget R
Debit R

Figure A-26
Departments file

To draw a graph showing the name of every department in the x-axis and the budget and debit of every department in the y-axis, you would write the following:

ALL RECORDS([Departments])
GRAPH FILE([Departments];1;[Departments]Name;[Departments]Budget;[Departments]Debit)

Now you want to draw a graph showing

□ the name of every department prefixed with its code in the x-axis

□ the budget and debit of every department

□ the difference between these two values in the y-axis

You create a layout like the one shown in Figure A-27.

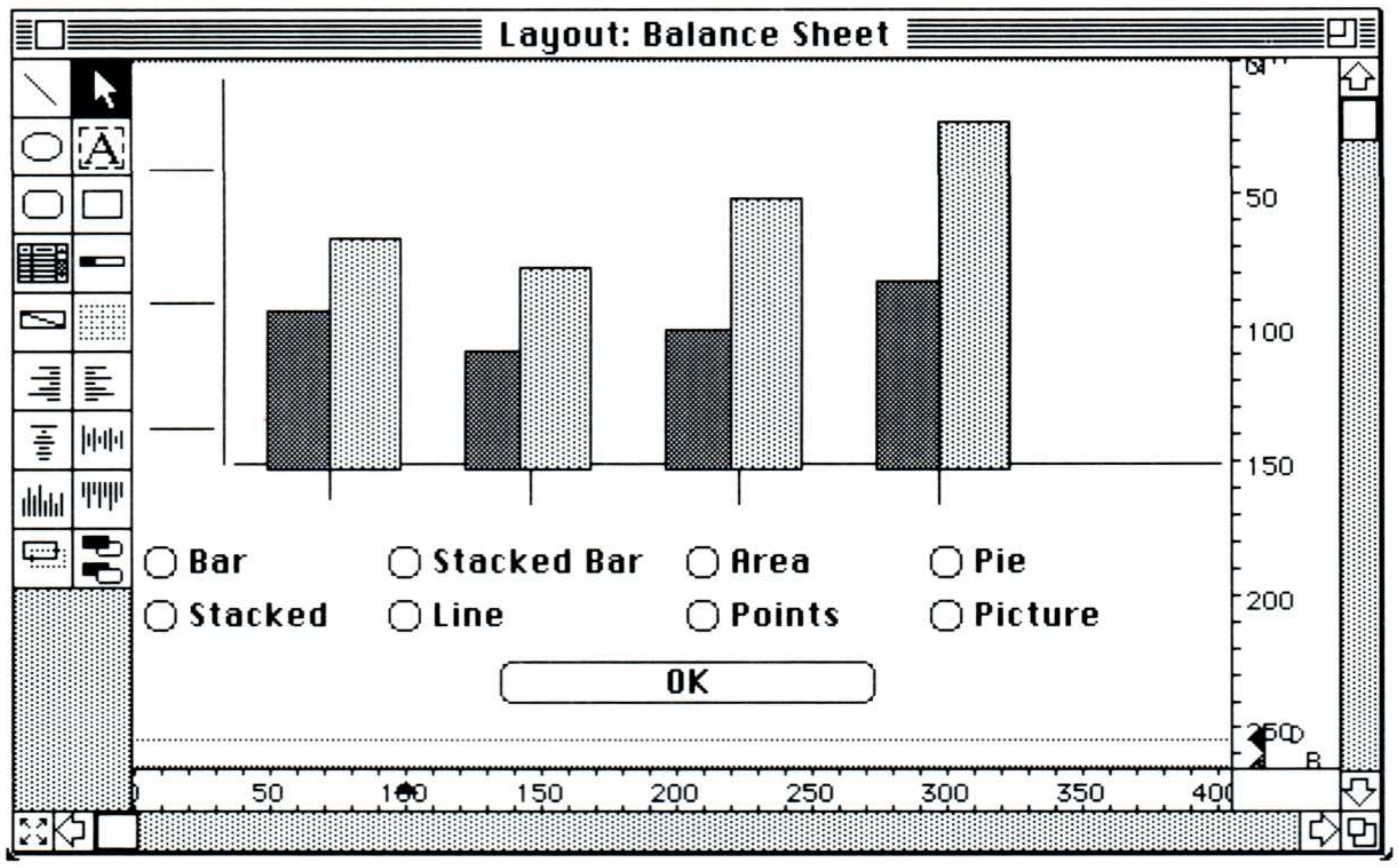


Figure A-27
Balance sheet graph layout

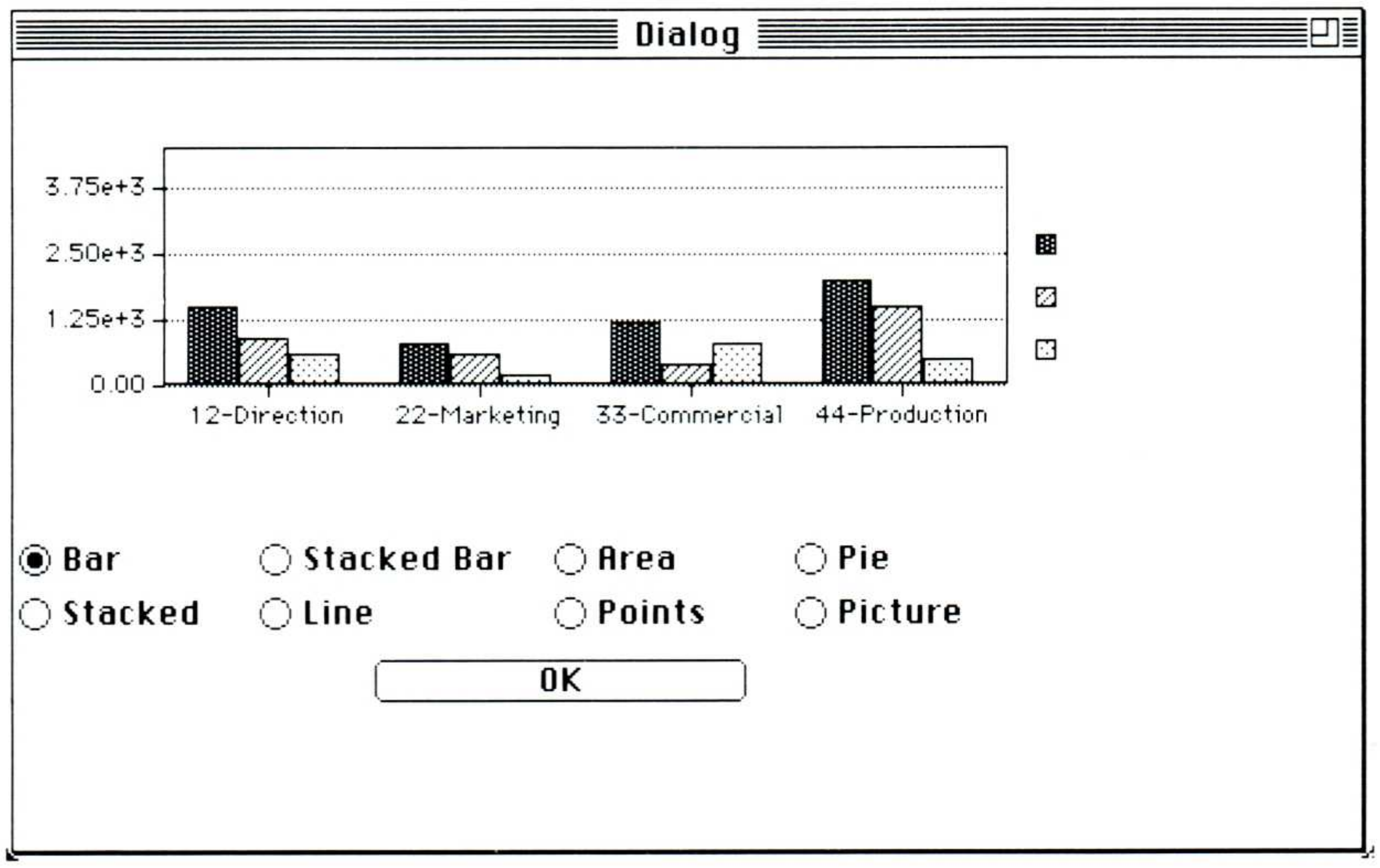
You must generate the graph using variables calculated from field data. This calls for the following procedure:

```
    `Balance sheet procedure :
DEFAULT FILE ([Departments])
    `Select all the records in the [Departments] file
ALL RECORDS
    `Create the variable tables.
    `The array named vLabel will contain the department codes and department names.
    `Array named vBudget will contain the budgets.
    `Array named vDebit will contain the debits.
    `Array named vDif will contain the difference between budgets and debits.
i := 0
While (Not (End selection))
    i := i + 1
    vLabel{i} := String ([Departments]Dept Code) + " - " + [Departments]Name)
    vBudget{i} := [Departments]Budget
    vDebit{i} := [Departments]Debit
    vDif{i} := [Departments]Budget - [Departments]Debit
    NEXT RECORD
End while
    `Determine the number of elements for each array.
vLabel0 := i
vBudget0 := i
vDebit0 := i
vDif0 := i
    `Send the "Balance sheet" dialog box to the screen to view the graph:
DIALOG ([Departments];"Balance sheet")
    `Delete variables from memory.
CLEAR VARIABLE ("v")
```

The graph is generated with the following layout procedure for the layout in Figure A-27:

```
If (Before)
 b1:=1
End if
If (Not(bOK=1))  `Skip when the user pressed the accept button
   `Compute the selected button
 $b:=(b1*1)+(b2*2)+(b3*3)+(b4*4)+(b5*5)+(b6*6)+(b7*7)+(b8*8)
   `Graph the function
   `MyMy1 is the name of the graph area variable in the layout.
 GRAPH(MyMy1;$b;vLabel;vBudget;vDebit;VDif)
End if
```

Figure A-28 shows the resulting graph.



Chosen setting: [✓ Bar]

Figure A-28
Graph of fields in Departments file

You can generate graphs by mixing subfields and arrays in the same GRAPH command. The general GRAPH syntax is

GRAPH (*var;posintexpr;strvarX* | *subfieldnameX;numvarY* | *subfieldnameY*{;*})

Links

This link example, based on the file structure shown in Figure A-29, shows how to use LOAD OLD LINKED RECORD and SAVE OLD LINKED RECORD when handling modifications to invoices. They have the effect of decrementing old accounts, so that the proper figures are maintained.

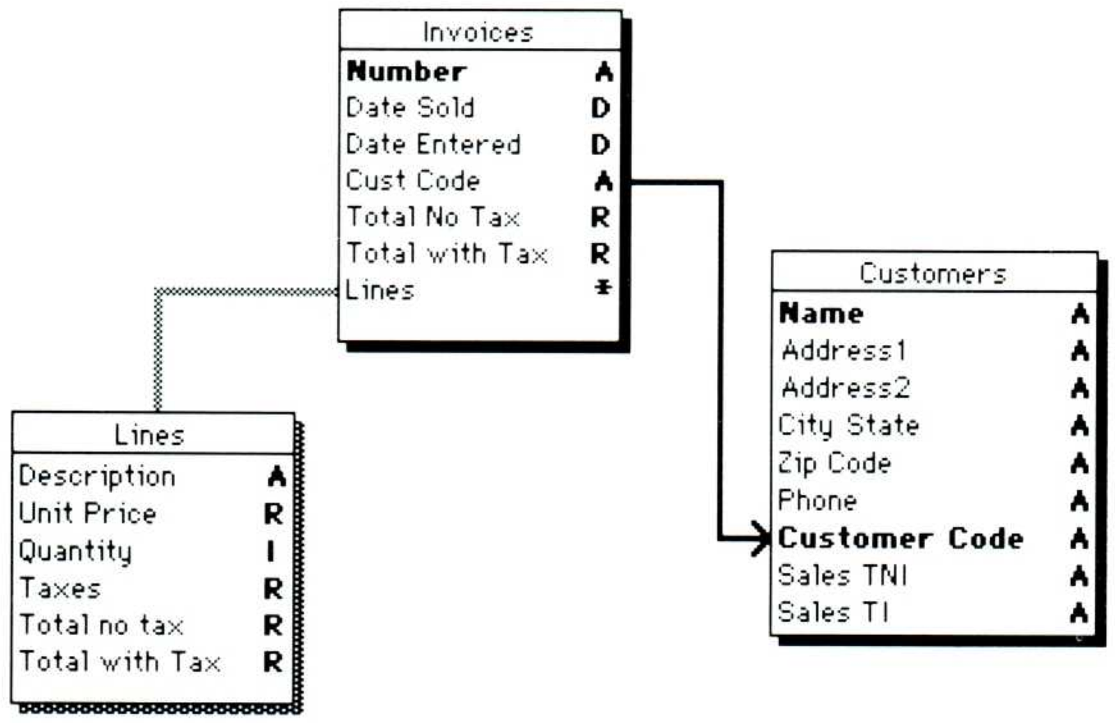


Figure A-29
Invoices database

Write the following the input layout procedure for the Invoices file:

```
 `Keep totals current (based on subrecords) at all times.
Total No Tax:=**Sum**(Lines'Total no tax)
Total with Tax:=**Sum**(Lines'Total with Tax)

 `Keep customer current
**LOAD LINKED RECORD**(Cust Code)

 `Update customer file when user validates a new record
**If** (**After**)
 [Customers]Outstanding:=[Customers]Outstanding+Total with Tax
 [Customers]Sales TNI:=[Customers]Sales TNI+Total No Tax
 [Customers]Sales TI:=[Customers]Sales TI+Total with Tax
 **SAVE LINKED RECORD**(Cust Code)
 **LOAD OLD LINKED RECORD**(Cust Code)
 [Customers]Outstanding:=[Customers]Outstanding-**Old**(Total with Tax)
 [Customers]Sales TNI:=[Customers]Sales TNI-**Old**(Total No Tax)
 [Customers]Sales TI:=[Customers]Sales TI-**Old**(Total with Tax)
 **SAVE OLD LINKED RECORD**(Cust Code)
**End if**
```

User interfacing

While in the Custom environment, you want to create a menu item to search for a given record within the Employees file of your database, so as to modify its contents. You write the ModifEmploy procedure and attach it to a menu item.

```
  `Displays dialog box "d.Search" asking user to enter employee names.  Searches for a given record and allows
  `user to modify it if found.  Continue asking for employees until [Cancel] buttton is pressed.

DEFAULT FILE([Employees])
$h:=Screen height/2  `Define center of screen.
$w:=Screen width/2

  `Select the "i.Entry" layout as the input layout.
INPUT LAYOUT("i.Entry")
  `We will stop only if the user selects the [Cancel] button in the dialog box.
Stop:=False
While (Not(Stop))
    `Display dialog box and get search info
  OPEN WINDOW($w-180;$h-70;$w+180;$h+70;1)
  DIALOG("d.Search")
  CLOSE WINDOW
    `Test the OK variable.  1 indicates <Enter> key or [OK] button pressed.
  If (OK#1)
   Stop:=True
  Else
   If ((vLastName="")|(vFirstName=""))
    ALERT("You must provide both first and last names.")
   Else
    SEARCH BY INDEX([Employees]LastName=vLastName;[Employees]FirstName=vFirstName)
    If (Records in selection#0)
       `Modify the record(s) selected
     OPEN WINDOW($w-200;$h-90;$w+200;$h+90;0;"Employee Entry")
     MODIFY RECORD
     CLOSE WINDOW
    Else
     ALERT("No record for "+vFirstName+" "+vLastName+" was found.")
    End if  `Records in selection#0
   End if  `names blank
  End if  `OK#1
End while  `not stop
```

Figure A-30 shows the Search dialog layout.

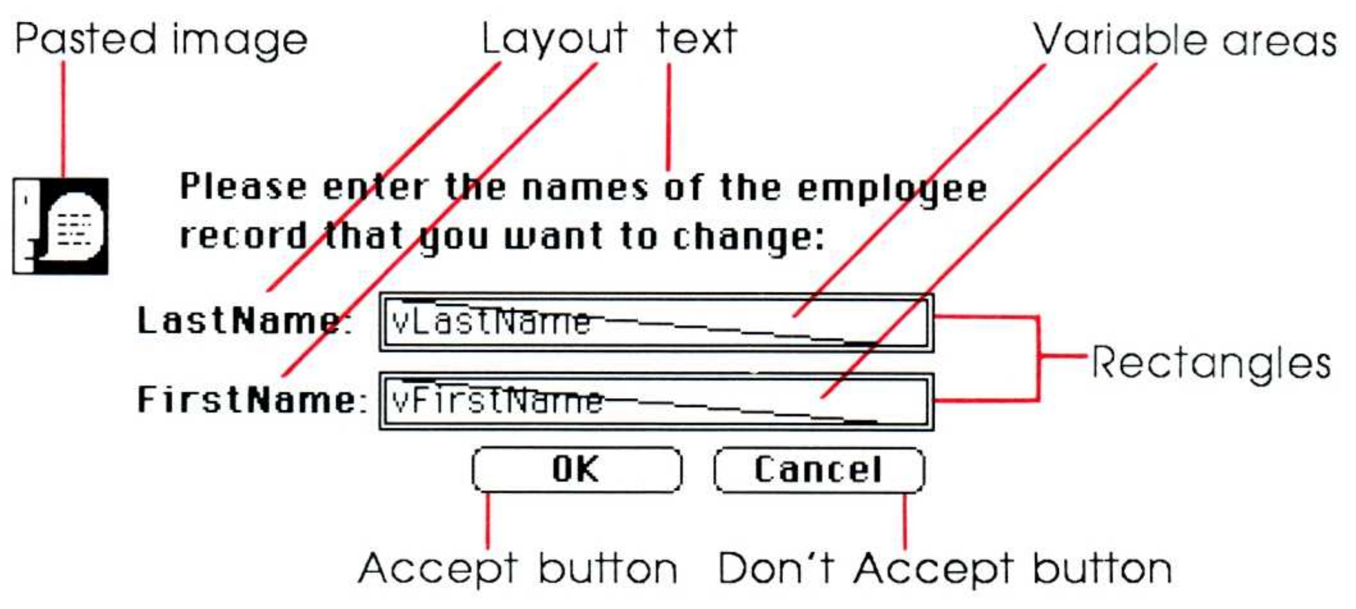


Figure A-30
Search dialog layout

Sending and receiving records and variables

This section contains procedures for sending and receiving records and variables. These include

- □ a procedure that adds variables and records to an invoice archive
- □ a procedure that retrieves variables and records from an invoice archive

Here is the procedure for sending variables and records to an invoice archive:

```
  `This procedure writes selected invoices to the serial port for pick-up by
  `"ReadArch" on a different machine.

  `Select the [Invoices] file as the default file.
DEFAULT FILE([Invoices])
  `Put up a dialog to ask the user the first and last days of the time period.
  `The dates are placed in the vBegin and vEnd variables.
vBegin:=!00/00/00!
vEnd:=!00/00/00!
DIALOG("d.ArchDial")
If (OK=1)
    `If the user validates the dialog, check the dates he/she entered.
  If (vBegin#!00/00/00!)
      `Check whether vBegin is a non-null date.
    If (vEnd#!00/00/00!)
        `Check whether vEnd is a non-null date.
      If (vBegin<=vEnd)
          `Is the first day of the period less than the last day of the period.
          `If all is OK, search by index the corresponding records.
        SEARCH BY INDEX([Invoices]Balance=0;[Invoices]Invoice date±vBegin;vEnd)
      If (Records in selection#0)
        `If records are found, inform the user and ask whether to continue.
       CONFIRM("OK to archive "+String(Records in selection)+" invoices?")
       If (OK=1)
         `If the user validate, ask for a name for the archival file.
         SET CHANNEL(12;"")
         If (OK=1)
              `If OK equals 1, the document has been created. Send a variable which contains the letters IA followed
              `by the current date, IA standing for Invoices Archived. These letters will tell exactly what the archival
              `file contains the next time the file is read.
          v:="IA"+String(Current date)
          SEND VARIABLE(v)
            `Send dates to save them in the archival file.
          SEND VARIABLE(vBegin)
          SEND VARIABLE(vEnd)
            `For each record in the selection, send the invoice
            `number copied to a variable, then the record itself.
          While (Not(End selection))
            v:=[Invoices]Number
            SEND VARIABLE(v)
            SEND RECORD
            NEXT RECORD
          End while
            `Send the zero value, to make sure, that when the
            `archival file is read, the end of file will be found.
          v:=0
          SEND VARIABLE(v)
            `Close the document.
          SET CHANNEL(11)
            `Delete the archived invoices from the file.
          MESSAGE("The archived invoices will be deleted")
          DELETE SELECTION
         End if
       End if
        `If the invoices cannot be archived, tell the user why.
     Else
      ALERT("There is no balanced invoice for that period.")
     End if
    Else
     ALERT("The end date for that period is prior to the begin date.")
    End if
   Else
    ALERT("You did not specify the end date for that period.")
   End if
  Else
   ALERT("You did not specify the begin date for that period.")
  End if
End if
```

Here is the procedure for reading variables and records from an invoice archive:

```
  `Read Archives procedure.
  `Ask the user to confirm the request.
CONFIRM("Do you want to read an invoices archival file?")
If (OK=1)
    `If the user validates the dialog, ask him/her to select the archival file.
  SET CHANNEL(10;"")
  If (OK=1)
      `If the user selected a document, read the first variable it contains.
    v:=""
    RECEIVE VARIABLE(v)
      `Check whether this is an invoices archival file.
    If (Substring(v;1;2)="IA")
        `If such is the case, retrieve the date on which the file was archived.
      vDate:=Date(Substring(v;3;Length(v))
        `Read the begin and end dates of the time period written to the document.
      RECEIVE VARIABLE(vBegin)
      RECEIVE VARIABLE(vEnd)
        `Here you can use a dialog to find out whether this is the right file.
        `You can also create a custom window to inform the user of the various
        `steps in the reading of the document.
      v:=1
        `With v initialized as 1, use a loop to go through the archival file.
        `You know the end of the file is reached when v is equal to zero.
      While (v#0)
        RECEIVE VARIABLE(v)
        If (v#0)
            `If v is not 0, there is a record in the document.
            `Create a new record . It becomes the current record.
          CREATE RECORD([Invoices])
            `The following command copies the values to the current record.
          RECEIVE RECORD([Invoices])
            `Once the values are copied, save the current file record.
          SAVE RECORD([Invoices])
        End if
      End while
    Else
      ALERT("The "+Document+" document is not an invoices archival file.")
    End if
  End if
    `Close the document.
  SET CHANNEL(11)
End if
```

Here is a procedure that prepares to receive the sent data.

```
  `The Install Com procedure is the procedure called from the menu.
  `If the variable is undefined, select the modem port as the default serial port
  `and set the port interface. Install the Call Detected procedure as a stop
  `routine for serial ports.

  `Assign a value to the zCom variable so it's no longer undefined and
  `place a check mark before the menu item to let the user know he/she will be
  `stopped if the serial port receives characters.
menu:=2  `Define the menu and item number that called this procedure
item:=1
If (Undefined(zCom))
 SET CHANNEL(1;94+3072+16384+8192)
 ON SERIAL PORT CALL("Call Detected")
 zCom:=1
 CHECK ITEM(Menu;Item;Char(18))
Else
   `The zCom variable is not undefined: remove the stop routine.
   `Clear the zCom variable to make it undefined again and
   `remove the check mark from the menu item.
 ON SERIAL PORT CALL("")
 CLEAR VARIABLE("zCom")
 CHECK ITEM(Menu;Item;"")
End if
```

Here is the procedure installed by ON SERIAL PORT CALL and used for detecting a call:

```
  `The Call Detected procedure is the stop routine installed with ON SERIAL PORT CALL
  `Ask the user whether to answer the call.
CONFIRM("Somebody is calling, do you want to answer?")
If (OK=1)
   `If this is the case, the user must answer using the following routines:
   `RECEIVE BUFFER, RECEIVE PACKET and SEND PACKET for an Ascii communication;
   `RECEIVE VARIABLE, SEND VARIABLE, RECEIVE RECORD and SEND RECORD for a
   `communication between two 4th Dimension programs.
 ReadArch
End if
```

Importing and exporting with subfiles

Figure A-31 shows a file Products (and its subfile Sales) and is the basis of this import and export example.

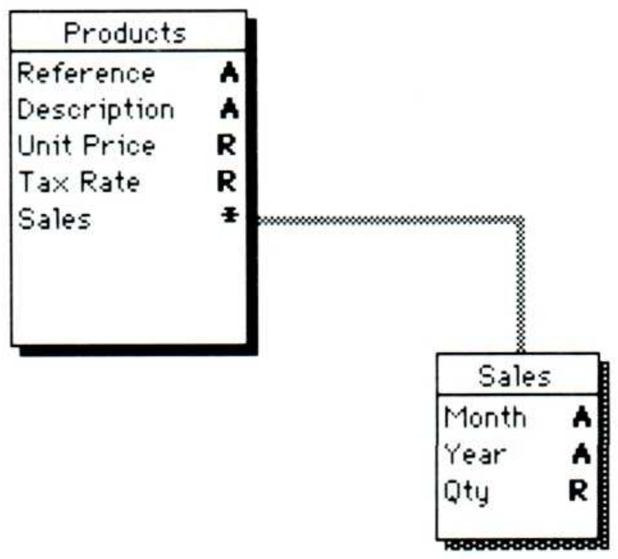
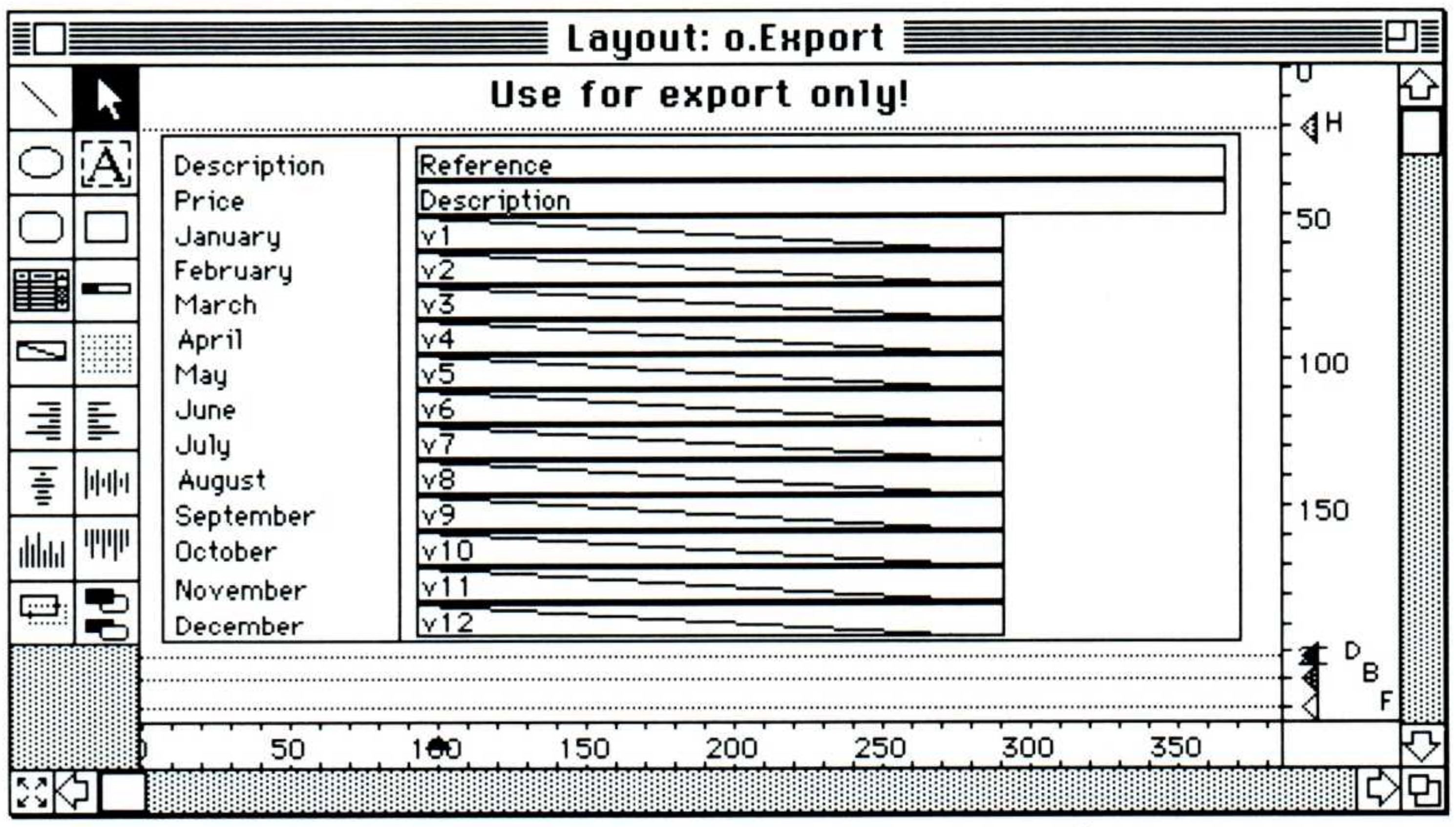


Figure A-31
Products file with Sales subfile

You want to exchange data between the Products file and another Macintosh application. You also want to exchange data contained in the [Products]Sales subfile. Specifying the subfile in an export layout doesn't solve the problem, because exporting ignores the subfile. Therefore, you must use variables.

You assume the file is operated on a one year basis and is reset at the end of each year. You know for sure that the number of subrecords in every record of the Products file cannot exceed 12. You create a data export layout in which you place 12 variables, in which you'll place the subrecords values with the help of layout formulas. Then you create an import data layout in which you place 12 variables which will receive the data read and from which you'll create subrecords.

Figure A-32 shows the layout named o.Export to be used when exporting data.


Layout: o.Export

Use for export only!

Description	Reference
Price	Description
January	v1
February	v2
March	v3
April	v4
May	v5
June	v6
July	v7
August	v8
September	v9
October	v10
November	v11
December	v12

Figure A-32
Export layout

You write the following export layout procedure:

```
If (Before)
   `The following code will execute once for every record in [Products]
   `before the data is written to the export document.
 ALL SUBRECORDS([Products]Sales)   `Select all the subrecords.
 SORT SUBSELECTION([Products]Sales;[Products]Sales'Year;>;[Products]Sales'Month;>)
   `Note this command also sets the current subrecord pointer to the first subrecord

   `Loop to assign the 12 layout variables:
 i:=1  `counter: increment i each time through loop.
 While (i<=12)  `for each variable in the layout
   If (End subselection([Products]Sales))
      `then have reached the end of the subselection, so assign 0 to variable
    v{i}:=0
   Else
      `Have not yet reached the last subrecord.
      `Assign the value from [Products]Sales'Qty into a variable.
    v{i}:=Sales'Qty
      `Move to current subrecord pointer to the next subrecord.
     NEXT SUBRECORD(Sales)
   End if   `end subselection(Sales)
     `increment counter
  i:=i+1
 End while   `i<12
End if   `Before
```

Figure A-33 shows the layout named “Import” to be used when importing data. It looks just like the layout used for exporting, but it has a very different layout procedure attached.

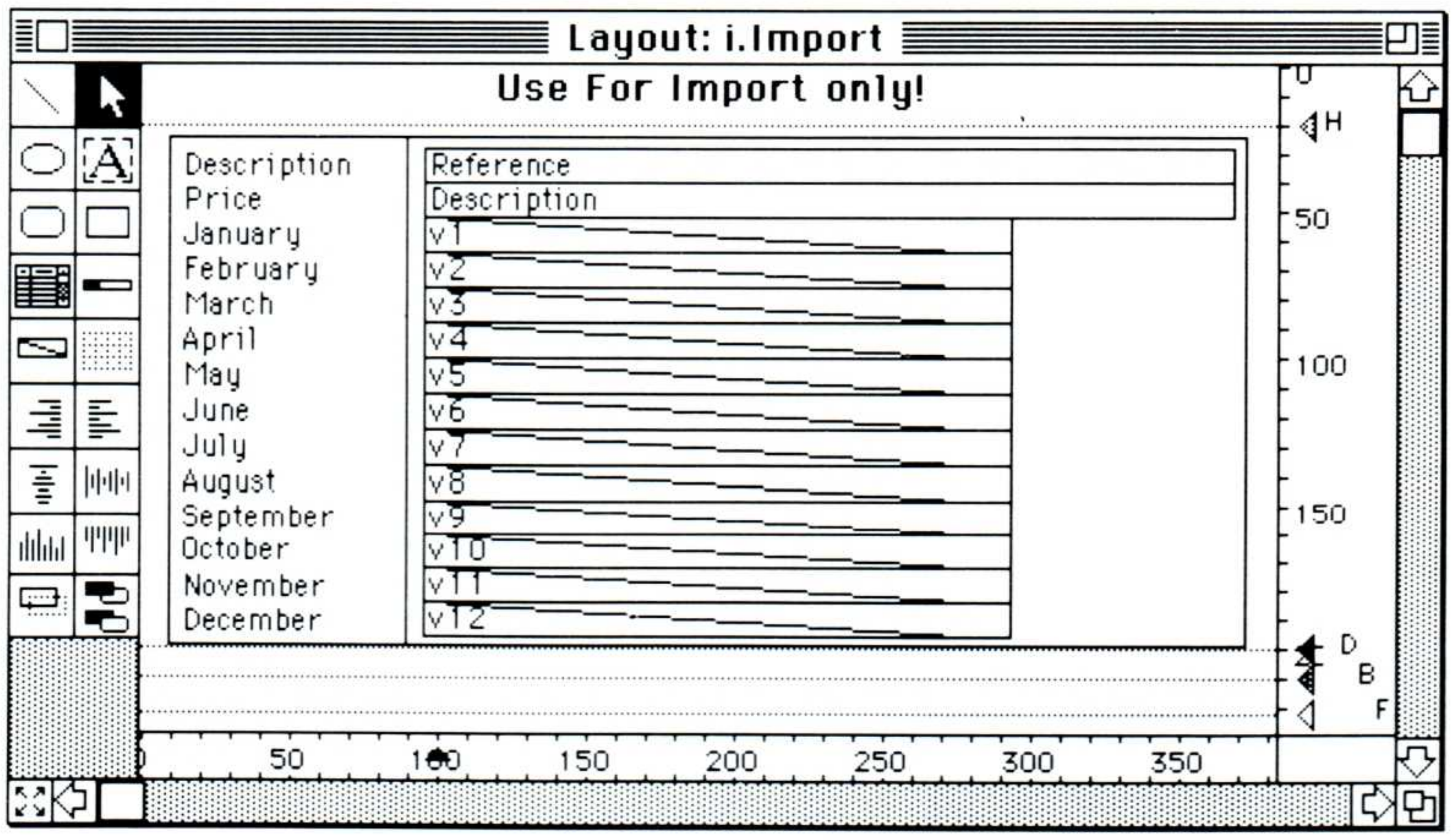


Figure A-33
Import layout

The following is the import layout procedure:

```
  `This layout procedure takes the 12 layout variables which were filled in by the import command and creates a
  `subrecord for each.

If (After)  `Only "After" phase is generated for import.
    `The following code will execute once for every record in [Products]
    `before the record is saved.
    `Loop to create the 12 subrecords:
  $i:=1  `counter: increment $i each time through loop.
  While ($i<=12)  `for each variable in the layout
    CREATE SUBRECORD([Products]Sales)
    [Products]Sales'Qty:=v{$i}  `Assign Quantity
    [Products]Sales'Month:=Month Name ($i)  `Assign month name
    [Products]Sales'Year:=String(Year of(Current date))  `Assign Year
    `increment counter for next subrecord.
    $i:=$i+1
  End while  `$i<12
End if  `After
```

Here is the Month Name function called by the procedure:

```
  `Global procedure: Month name
  `Given a month number ($1), return a name ($0)

$mnAll:="January  "+"February "+"March    "+"April    "+"May      "+"June     "+"July     "+"August   "
          +"September"+"October  "+"November "+"December "
$0:=Substring($mnAll;($1*9)-8;9)  `Grab one and only 1 month name
```

Using the EXECUTE command

EXECUTE can dramatically increase the efficiency of your programming efforts. EXECUTE executes its argument, a string. This section contains a few examples of what it can do.

A typical example is calling a procedure based on the value contained in a variable. The variable, *var,* can contain the values 1, 2, 3, or 4, and depending on the value you'll call the corresponding *Proc1, Proc2, Proc3,* or *Proc4* procedure. If you don't use EXECUTE, you have to write something like this:

```
Case of
    :(var = 1)
      Proc1
    :(var = 2)
      Proc2
    :(var = 3)
      Proc3
    :(var = 4)
      Proc4
End case
```

However, if you use EXECUTE, you write the one-line statement

```
EXECUTE ("Proc" + String (var))
```

In this case, the argument of EXECUTE concatenates the value of *var* to *Proc*. Then EXECUTE executes the resulting string as a 4th Dimension instruction.

Here is an example in which EXECUTE assigns field values to appropriate variables. The data base in question contains a file named *filename* with the following fields: *fieldname1, fieldname2, . . . , fieldname20*. The goal is to copy the values contained in these fields into the following variables: *var1, var2, . . . , var20*. One solution would be to write twenty lines that look like this:

```
var1 := [filename]fieldname1
var2 := [filename]fieldname2
.
.
.
var20 := [filename]fieldname20
```

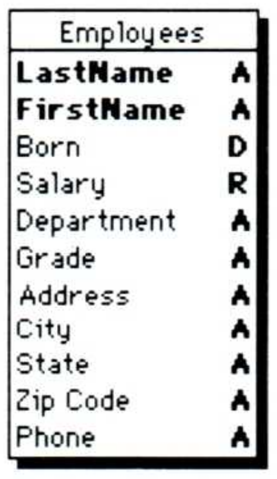
The EXECUTE routine lets you use a loop instead of writing numerous lines:

```
i := 1
While (i < 21)
    EXECUTE ("var{i} := [filename]fieldname" + String (i))
    i := i + 1
End while
```

For example, for i = 15, the following instruction is executed:

```
var{i} := [filename]fieldname15.
```

You can also use EXECUTE to read buttons in a dialog. For example, when printing or displaying the current selection of the Personnel file shown in Figure A-34, you present the user with the custom dialog shown in Figure A-35 instead of the standard window.



Employees	
LastName	A
FirstName	A
Born	D
Salary	R
Department	A
Grade	A
Address	A
City	A
State	A
Zip Code	A
Phone	A

Figure A-34
Personnel file

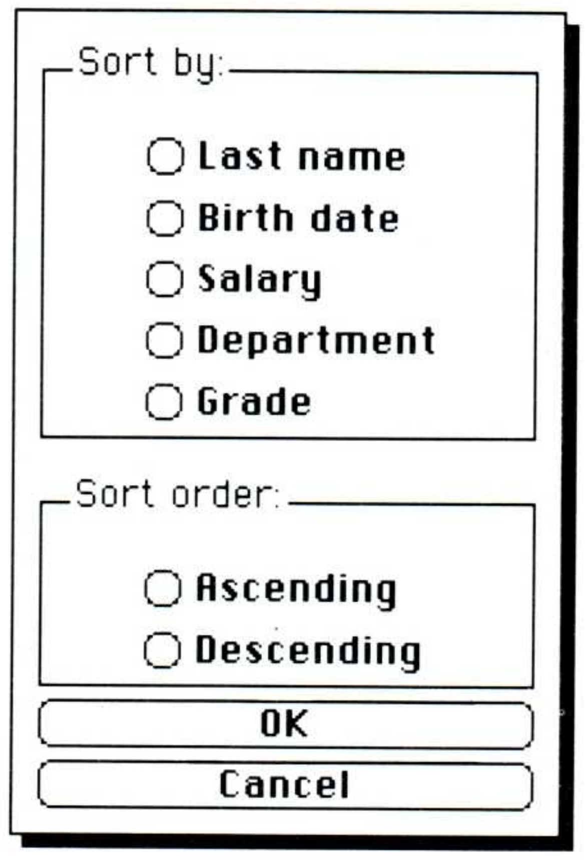


Figure A-35
Custom Sort dialog box

The programming problem is how to read the button selecting which field to sort on. You could write a procedure following this pattern:

```
  `Presents a dialog box and sorts the current selection of the employees file as directed.

  `Radio buttons rb1, rb2, ... rb5 are used to select the field to sort by.
  `Radio button Other1 indicates ascending
  `Radio button Other2 indicates descending

DIALOG([Employees];"d.Sort")
If (OK=1)
   `Assign n:= 1, 2, ... or 5.
   `At most one radio button can be on at a time.
 n:=(1*rb1)+(2*rb2)+(3*rb3)+(4*rb4)+(5*rb5)
   `determine the Order button number that was pressed (if any)
 Order:=(1*Order1)+(2*Order2)  `0, 1, or 2
   `Convert to ">" or "<" string.
 Order:=Substring(">><";Order+1;1)  `Ascending if none selected.
 ALERT(order)
 If ((n>0)&(n<6))
  fieldname1:="LastName"
  fieldname2:="Born"
  fieldname3:="Salary"
  fieldname4:="Department"
  fieldname5:="Grade"
  EXECUTE("SORT SELECTION([Employees]"+fieldname{n}+";"+Order+")")
 Else
  ALERT("Sort field not specified.")
 End if `(n>0)&(n<6)
End if  `OK=1
```

The expression

```
n := (1*rb1) + (2*rb2) + (3*rb3) + (4*rb4) + (5*rb5)
```

works because only one of the five radio buttons can return one. The others must return zero. Thus, zero buttons in the expression return zero and the pressed button returns its own value times one. When you add the button values together, you'll get the value of the pressed button only.

Appendix B

Using Trace

This appendix discusses the 4th Dimension interpreter and debugging techniques.

The 4th Dimension interpreter

The 4th Dimension interpreter is responsible for executing developer-written procedures (whether written as listings or as flowcharts). Execution begins in the Custom environment when the user makes a menu selection. In the User environment, you initiate execution by choosing Execute from the Special menu and then selecting a procedure. Procedure execution is also initiated anytime a layout with an associated procedure is displayed.

Any of these methods launches the 4th Dimension interpreter, which executes instructions by interpreting them. When a procedure is executed, here's what the interpreter does:

1. If the procedure is not already in memory, the interpreter reads the procedure from disk.
2. The interpreter then analyzes the instruction contained in the first line (listing) or in the first step or test (flowchart). If the syntax is correct, 4th Dimension executes the instruction. If not, it stops the execution process and displays an error message.
3. Every time an instruction is correctly executed, the interpreter goes to the next instruction and repeats the same process until it reaches the end of the procedure.

The word "interpreter" is used because whenever an instruction is correct, 4th Dimension translates or interprets what you wrote in the 4th Dimension high-level programming language into equivalent instructions that the Macintosh can execute.

❖ *Compilers:* The other way to execute programs is with a compiler. The programmer writes a program in a given programming language and then compiles the program into instructions the computer can understand. These instructions are then executed without accessing the original text.

The 4th Dimension interpreter offers the developer several advantages:

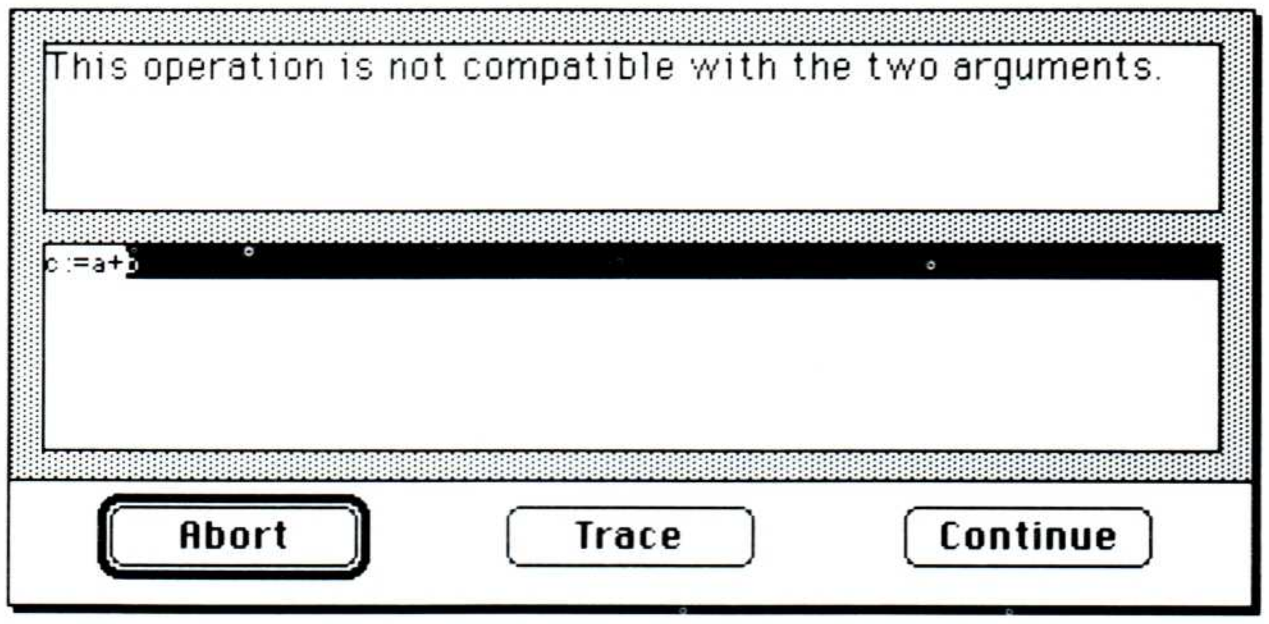
- □ You create a procedure and then execute it directly, without going through intermediary steps.
- □ When an error occurs during the execution process, 4th Dimension detects the faulty instruction and displays where in the procedure it occured.
- □ You can rapidly correct an error and check the accuracy of your modification by executing the procedure again.
- □ 4th Dimension enables you to follow the execution cycle step by step using Trace.

4th Dimension default setting for procedure execution is No Trace. However, you can switch to Trace anytime as explained later on in this appendix. When set on Trace, 4th Dimension stops after executing every line of the procedure, and displays a window in which you can

- □ view the procedure which is being executed, including the line which was executed last.
- □ view the contents of variables and fields.
- □ view the value of any expression.
- □ set breakpoints, so that the procedure executes normally until the breakpoint is reached. When 4th Dimension encounters a breakpoint, it stops before executing the line marked by the breakpoint and turns Trace on.

Working with Trace

When the interpreter detects an error or when you stop the execution with Option-Click, 4th Dimension displays the dialog box shown in Figure B-1.


This operation is not compatible with the two arguments.

```
c:=a+b
```

Figure B-1
Error dialog

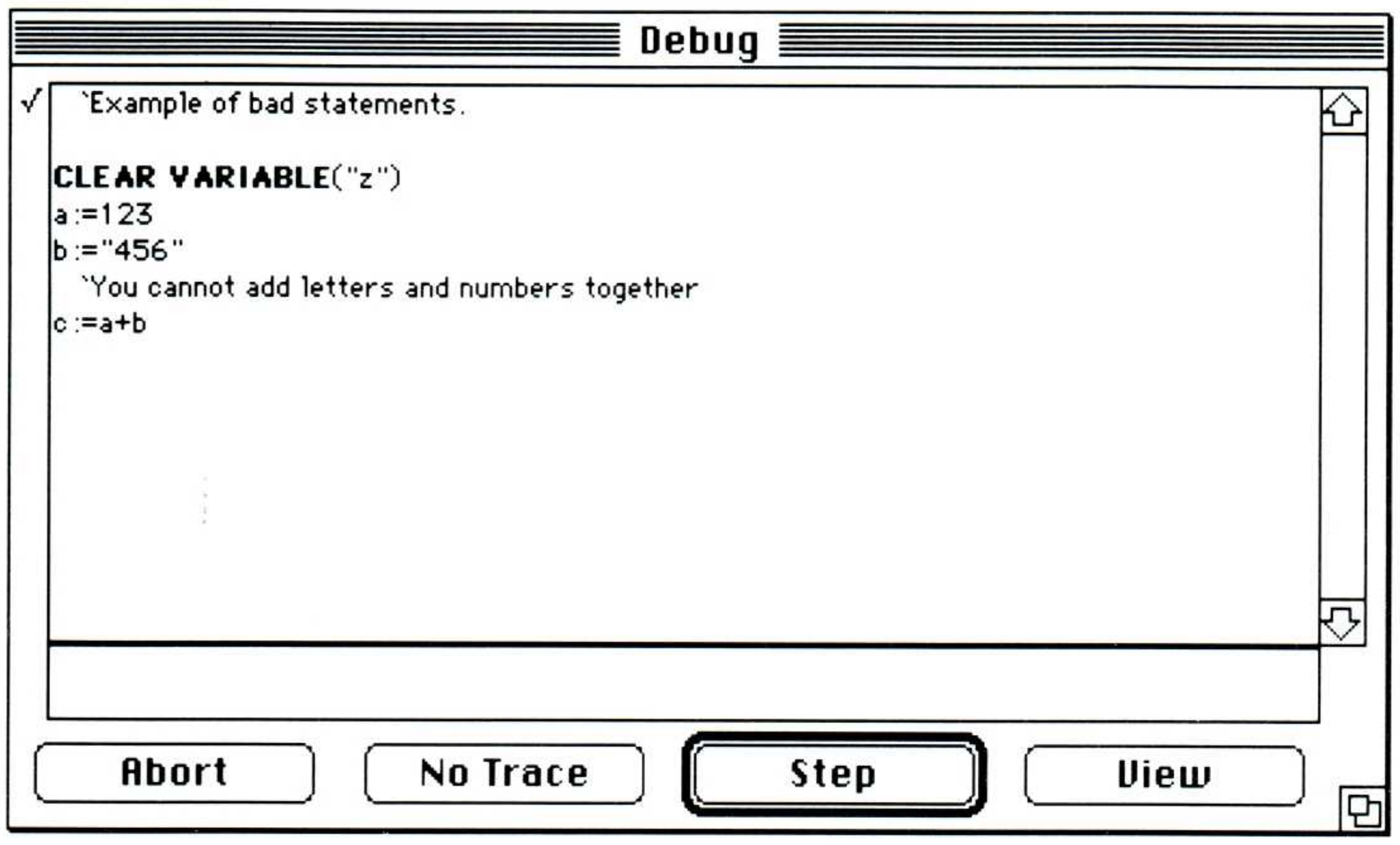
The top text area displays a message indicating the error type, for instance "Alphanumeric argument expected" or "Procedure stopped by user" if you pressed Option-Click. The bottom text area displays the faulty instruction, and highlights the point where the error originated or the execution was interrupted.

❖ *Password protection:* If the database is protected by a main password, you may not stop the execution of a procedure by pressing Option-Click, unless the database was opened using the main password.

The Continue button enables you to resume the execution process. If you click the Abort button, 4th Dimension stops executing the procedure. If you're dealing with a procedure related to a menu, you go back to the menu bar even if the procedure which was stopped is not the calling procedure. If you're executing a layout or file procedure, you remain in the layout but the current execution phase is terminated.

The error window will not be displayed if you installed a routine with ON ERR CALL. In this case, the event invokes your installed error handling procedure. Should an error occur or an interruption be forced in your installed routine, a 4th Dimension error window is displayed. This is because your error handling procedure cannot be invoked if the error originates in the procedure itself.

The Trace button lets you enter the Trace mode and causes 4th Dimension to display the Debug window shown in Figure B-2.



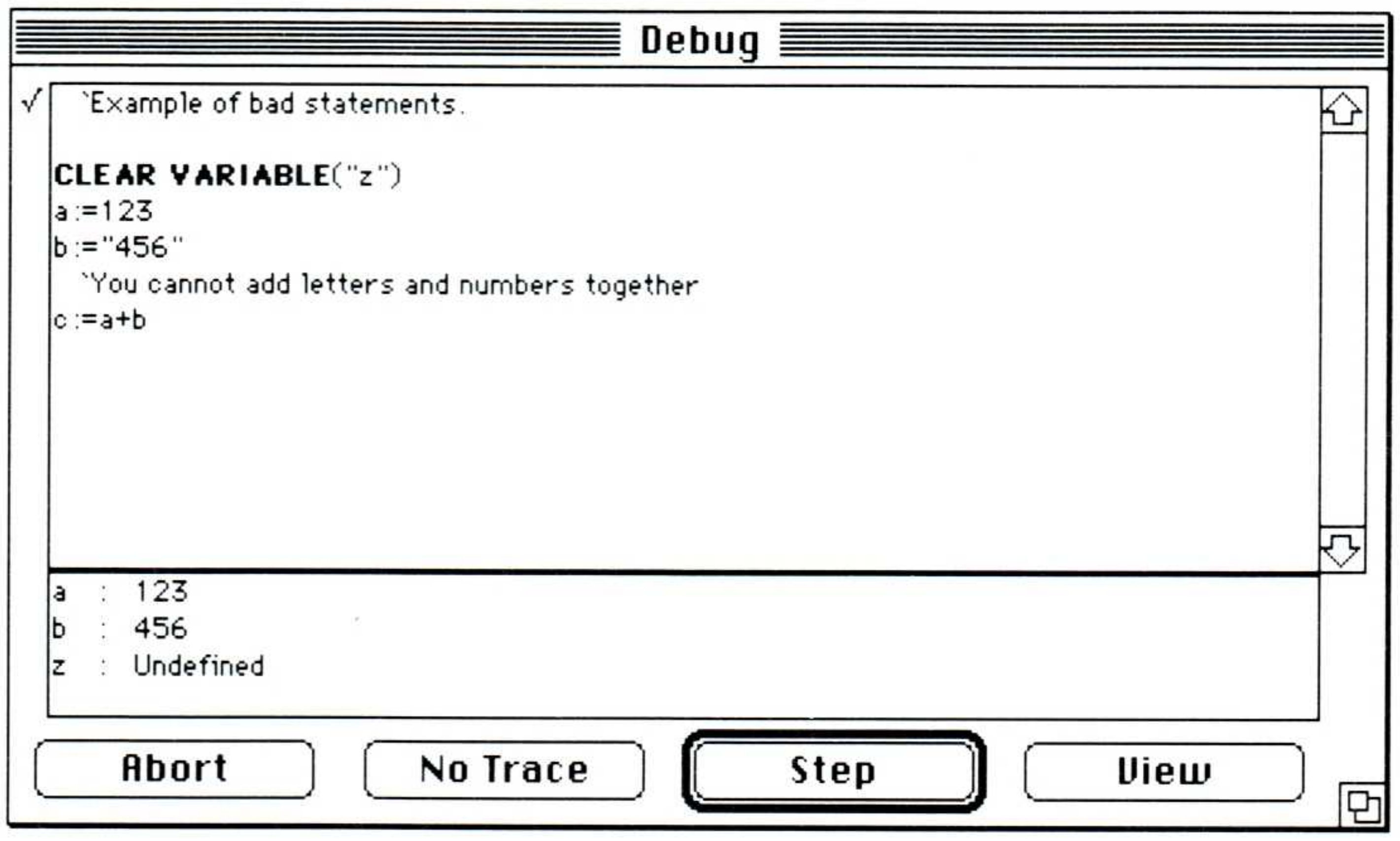
```
√  `Example of bad statements.

CLEAR VARIABLE("z")
a:=123
b:="456"
  `You cannot add letters and numbers together
c:=a+b
```

Figure B-2
Debug window

The Debug window is divided into two parts by a thick horizontal line. The top part of the window displays the instructions being executed. A check mark appears to the left of the line to be executed next.

❖ *Flowchart editor:* If you have written a procedure in the flowchart editor, the step or test that is to be executed next is highlighted.

Use the bottom part of the window to type in the names of variables, fields, system variables and expressions. 4th Dimension will display in that area the values contained or specified by what you typed. Figure B-3 shows the Debug window with values displayed.



```
√ `Example of bad statements.

CLEAR VARIABLE("z")
a:=123
b:="456"
  `You cannot add letters and numbers together
c:=a+b
```

```
a  :  123
b  :  456
z  :  Undefined
```

Figure B-3
Debug window with values displayed

❖ *Undefined:* The z variable is undefined. That is, it does not contain any value and its type is unknown.

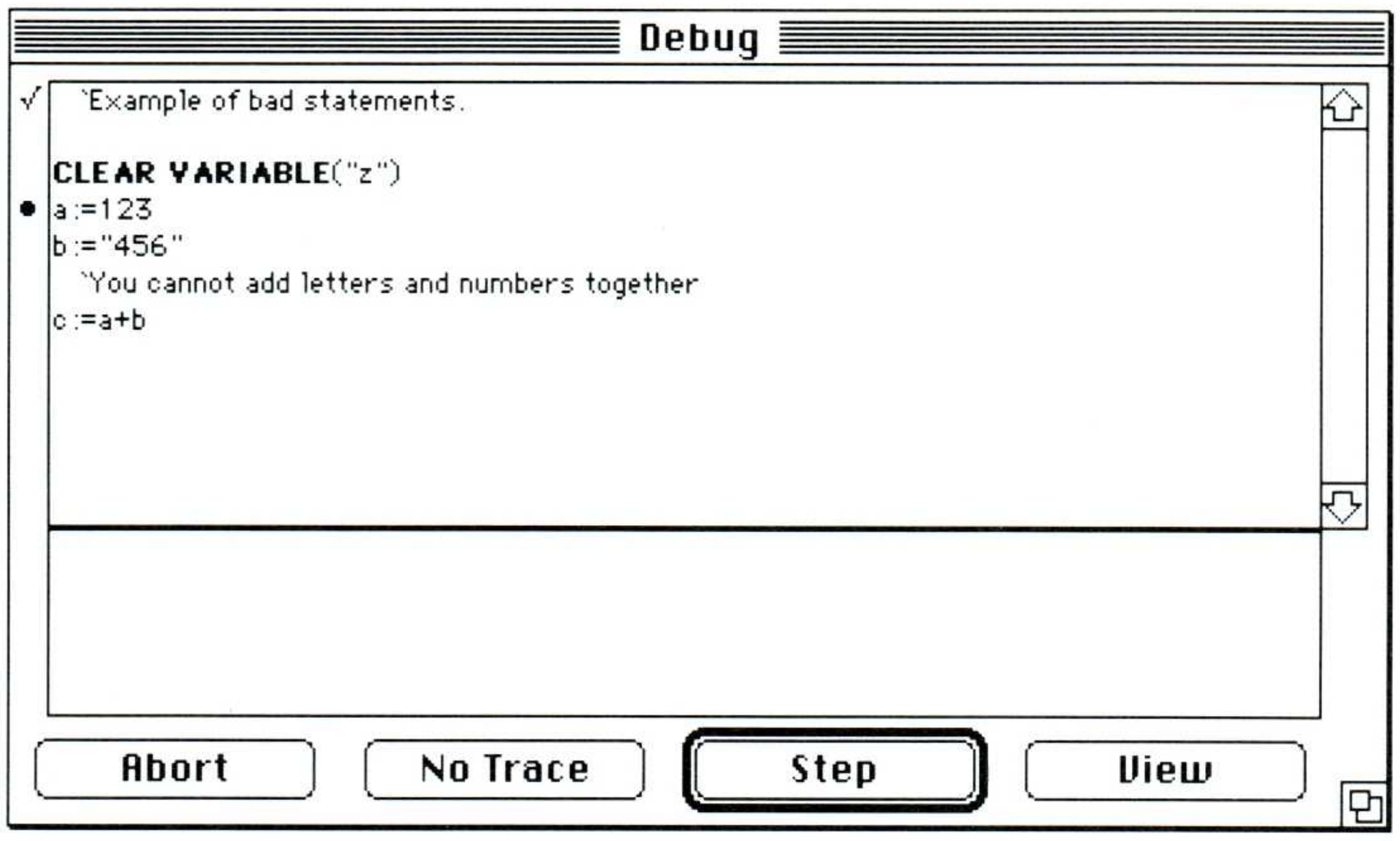
You may modify the size of the bottom part window by pressing the Option key while clicking inside the box; the horizontal dividing line will be placed on the point where you clicked.

The Debug window can also be displayed by invoking the TRACE command in a procedure. Once Trace is activated, 4th Dimension executes each statement step by step, and for every line, the Debug window is displayed so you can follow the execution in progress, the change in value for a variable, the calls to procedure, and so on.

The Abort button acts in the same way as in the error window. The No Trace button disables the Trace mode: 4th Dimension goes back to executing the procedure in the normal way. The Step button lets you resume execution while remaining in the Trace mode. The View button lets you view what happened on the screen. To go back to the Debug window, click the mouse button.

You may disable Trace by executing NO TRACE in a procedure. Once Trace is enabled, it can only be disabled by clicking the No Trace button in the Debug window or by executing a NO TRACE statement.

You can also set breakpoints. To do so, click on the left of a line in the Debug window: a bullet appears. To delete a breakpoint, click again on the breakpoint. Figure B-4 shows the Debug window with a breakpoint set.



```
√  `Example of bad statements.

   CLEAR VARIABLE("z")
●  a:=123
   b:="456"
     `You cannot add letters and numbers together
   c:=a+b
```

Figure B-4
Debug window with breakpoint set

When a breakpoint is set, 4th Dimension automatically enters the Trace mode whenever it encounters an instruction marked with a breakpoint, even if you clicked the No Trace button or invoked the NO TRACE routine in your formulas. The breakpoint is stored in memory. To delete a breakpoint, you must click on it again.

Use Trace when a procedure does unexpected things. That way you'll locate errors quickly, especially for undefined variables, loops or tests incorrectly structured, typing errors, syntax errors, and so on.

Appendix C

Miscellaneous

This appendix covers four topics of interest to 4th Dimension developers and users. These include

□ Apply Formula and APPLY TO SELECTION

□ correcting a database design error

□ working with a scrollable area

□ updating copies of a database

Applying formulas and APPLY TO SELECTION

The User environment's Apply Formula command and the APPLY TO SELECTION command are both powerful and flexible. Their power is due to the fact that they act on an entire selection of records. Their uses include

□ modifying data or its presentation

□ creating and assigning data

□ scanning string lengths

□ performing calculations

Modifying data or its presentation

Suppose you have a file, Persons, and realize the user has typed all the names in lowercase characters. Apply Formula can correct the problem. Select the entire file and apply the following formula:

```
[Persons]Name := Uppercase ([Persons]Name)
```

Creating and assigning data

In the same way, if you have a file containing a date and need a new field with the month of that date, create a new field [MyFile]Month to which you apply the following formula:

```
[MyFile]Month := Month of ([MyFile]MyDate)
```

Scanning string lengths

Indexes for Alpha fields use the specified field length when created. If you overestimate the number of characters for an indexed Alpha field, you may want to regain disk space by reducing the field's length. You can find the length of the longest string in an Alpha field using a procedure with Apply Formula or APPLY TO SELECTION. The SearchMax procedure does this:

```
`SearchMax procedure
If (Undefined (vMax))
        `If the vMax variable is undefined, initialize it.
    vMax := 0
End if
If (Length ([MyFile]MyField) > vMax)
        `If the length of the value contained in the field is greater than vMax,
        `place this length in the vMax variable.
    vMax := Length ([MyFile]MyField)
End if
```

MyField is an indexed alphanumeric field contained in your file. If you apply this procedure to all the records in the file, vMax will contain the maximum length of the entries in MyField. Choose Apply Formula from the Entry menu and specify the SearchMax procedure in the formula editor. To display vMax, execute the procedure

```
ALERT(String(vMax))
```

Suppose you have an indexed alphanumeric field with a specified length of 45 characters. The index will take up approximately 61000 bytes (≈59.5K) for 1000 records. After SearchMax has been applied, you realize that the longest entry contains 33 characters. You go back to the Design environment and change the length of the field to 35 characters. The index will then take up approximately 51000 bytes (≈50K). The operation freed 9.5K of disk space.

The APPLY TO SELECTION command is equivalent to Apply Formula. Executing the following two procedures has the same effect. In this case, the SearchLength procedure calls the SearchMax procedure through the APPLY TO SELECTION command.

```
`SearchMax procedure
If (Length ([MyFile]MyField) > vMax)
    vMax := Length ([MyFile]MyField)
        `place this length in the vMax variable.
End if
```

```
`SearchLength
DEFAULT FILE ([MyFile])
ALL RECORDS
vMax := 0
APPLY TO SELECTION (SearchMax)
ALERT ("The maximum length is " + String (vMax) + " characters.")
```

You can identify the SearchLength procedure with a custom menu item or execute it in the User environment by choosing Execute from the Special menu.

Performing calculations

You can create your own arithmetic functions (like Sum and Average) that work on screen rather than at print-time only (as 4th Dimension's built-in functions do). To compute totals without printing the selection, write two procedures—Compute and Calc. Compute calls Calc from an APPLY TO SELECTION statement.

```
    `Calc procedure
vSum := vSum + [MyFile]MyField
If ([MyFile]MyField > vMax)
    vMax := [MyFile]MyField
End if
If ([MyFile]MyField < vMin)
    vMin := [MyFile]MyField
End if
```

```
    `Compute procedure
DEFAULT FILE ([MyFile])
SEARCH
If (OK = 1)
    vSum := 0
    vMax := -1*(10^99) `A very small number
    vMin :=1*(10^99) `A very large number
    APPLY TO SELECTION (Calc)
    ALERT("Total: "+String(vSum))
    ALERT ("Average: " + String (vSum / Records in selection))
    ALERT ("Maximum: " + String (vMax))
    ALERT ("Minimum: " + String (vMin))
End if
```

Correcting a database design error

This section shows you how to change the structure of a 4th Dimension database without discarding or losing data. As an example, suppose you had originally created the database shown in Figure C-1.

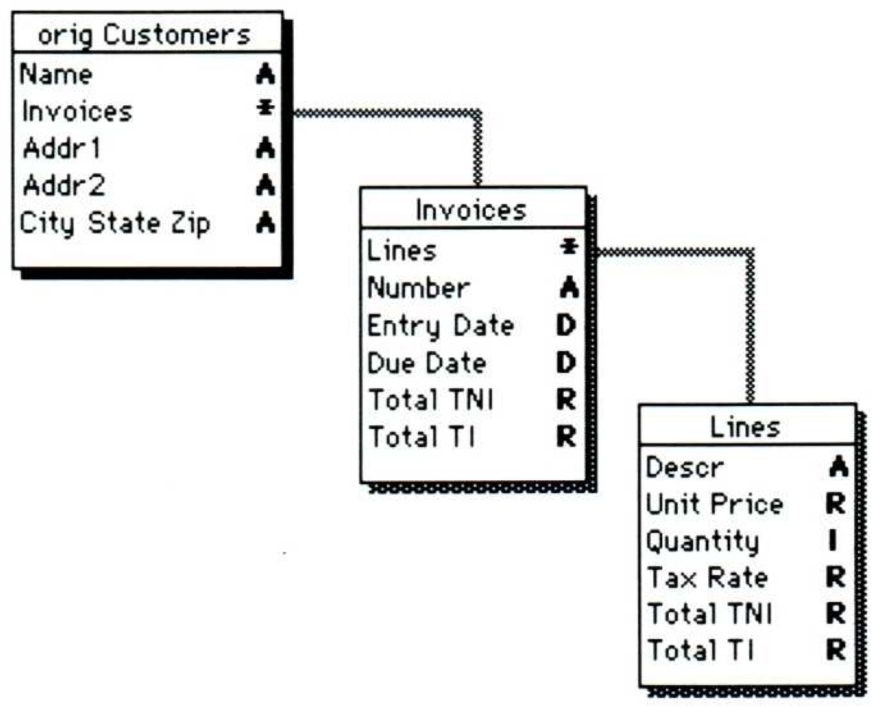


Figure C-1
Original database design

After working with it for awhile, you created a new design (shown in Figure C-2) that would be more efficient.

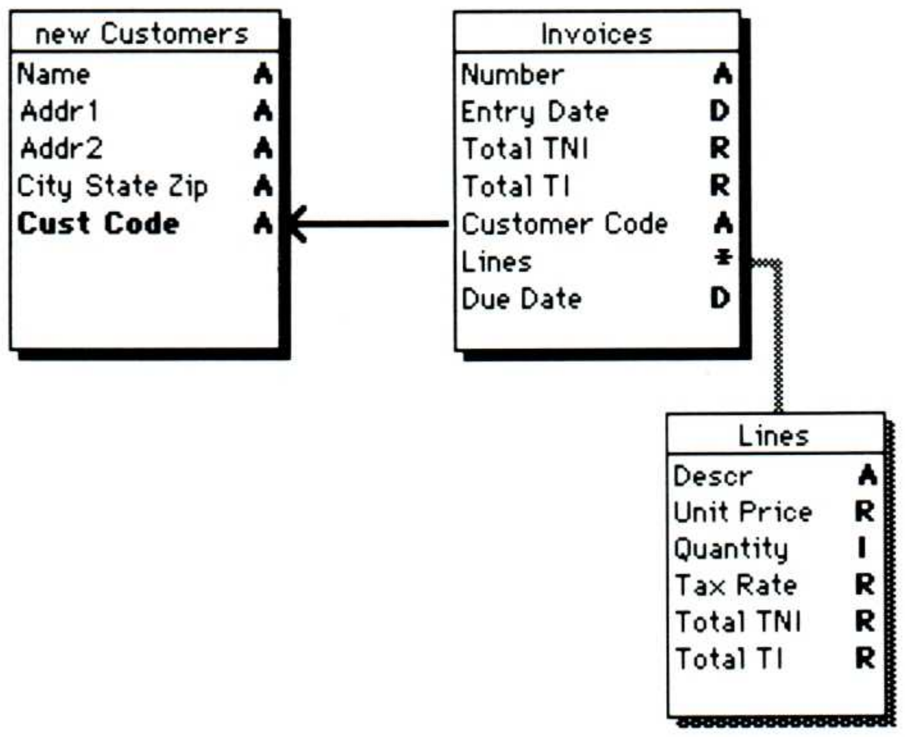


Figure C-2
Improved database design

The following steps show you how to modify the original structure and create a procedure to generate the improved structure from the old one, without having to enter the data again:

1. Back up the database.
2. Add the files from Figure C-2. At this point, the database will look like the one shown in Figure C-3.

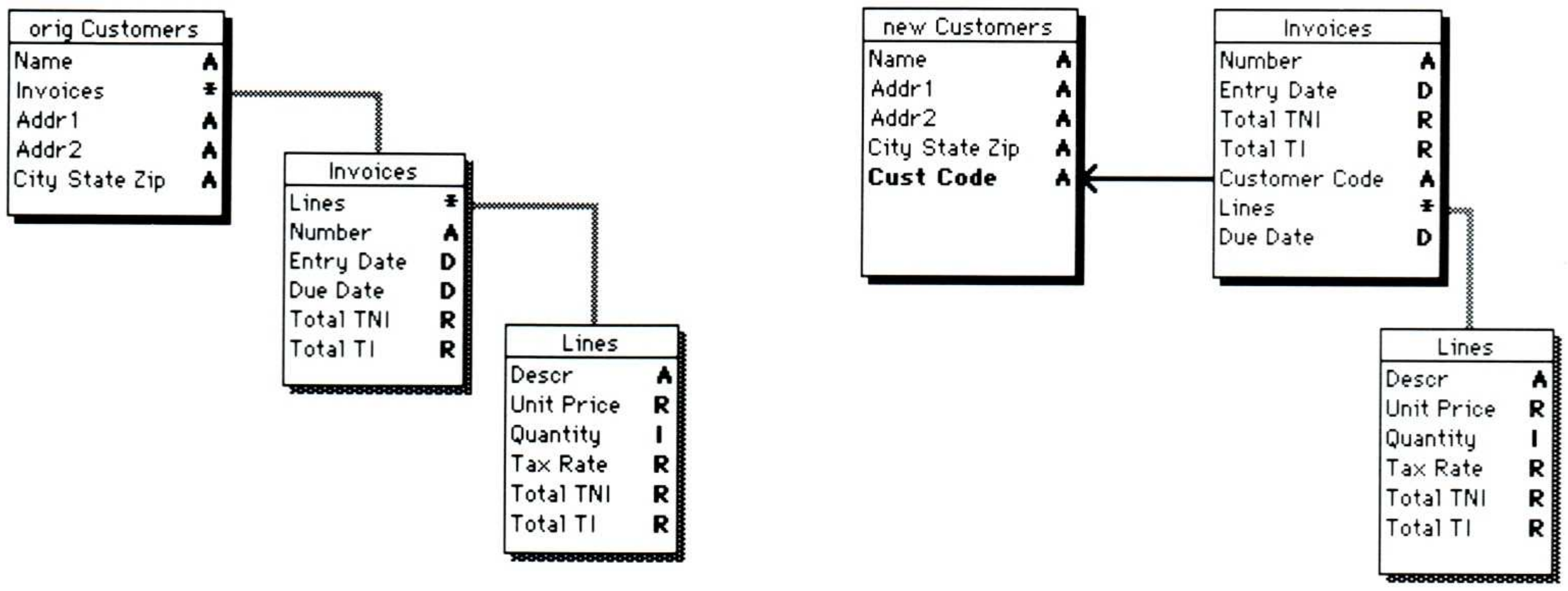


Figure C-3
Database with Invoices file added

3. Enter the Transfer procedure (given below) which will copy data between the files.
4. Execute the Transfer procedure.
5. Make a second backup of the database.
6. Delete the orig Customers file.
7. Modify layouts and procedures accordingly.
8. Make a third backup of the database.

Here is the Transfer procedure:

```
  `This procedure transfers data from a file/subfile/subsubfile structure to a file/subfile and link to other file
  `structure.

DEFAULT FILE([orig Customers])
  `Select all customers.
ALL RECORDS
  `Sort by the customer name.
SORT BY INDEX([orig Customers]Name;>)
vCode:="XXX"  `3 character start of customer code
vNum:=1  `Used for 3 digit number appended to customer code.
While (Not(End selection))  `==Customer==
 If (vcode#Substring([orig Customers]Name+"XXX";1;3))
   vCode:=Uppercase(Substring([orig Customers]Name+"XXX";1;3)
  vNum:=1  `Found differing first three letters, start count over.
 Else
    `Customer name starts with the same 3 letters, so we update
    `the three digit number appended to the customer code.
  vNum:=vNum+1
 End if
 CREATE RECORD([new Customers])
 [new Customers]Cust Code:=vCode+String(vNum;"000")
 [new Customers]Name:=[orig Customers]Name
 [new Customers]Addr1:=[orig Customers]Addr1
 [new Customers]Addr2:=[orig Customers]Addr2
 [new Customers]City State Zip:=[orig Customers]City State Zip
 SAVE RECORD([new Customers])
   `select all (old) customer invoices
 ALL SUBRECORDS([orig Customers]Invoices)
 While (Not(End subselection([orig Customers]Invoices)))  `==Invoice==
    `While there is still an invoice create record in the [Invoices] file
  CREATE RECORD([Invoices])
    `Copy the data to the invoice file:
  [Invoices]Number:=[orig Customers]Invoices'Number
  [Invoices]Entry Date:=[orig Customers]Invoices'Entry Date
  [Invoices]Due Date:=[orig Customers]Invoices'Due Date
  [Invoices]Total TNI:=[orig Customers]Invoices'Total TNI
  [Invoices]Total TI:=[orig Customers]Invoices'Total TI
  [Invoices]Customer Code:=[new Customers]Cust Code
    `Select all the invoice line items
    ALL SUBRECORDS([orig Customers]Invoices'Lines)
    While (Not(End subselection([orig Customers]Invoices'Lines)))  `==Line Item==
       `While there still is a line item
       `Create a line (subrecord) in the [Invoices] file.
      CREATE SUBRECORD([Invoices]Lines)
       `Copy the line data
      [Invoices]Lines'Descr:=[orig Customers]Invoices'Lines'Descr
      [Invoices]Lines'Unit Price:=[orig Customers]Invoices'Lines'Unit Price
      [Invoices]Lines'Quantity:=[orig Customers]Invoices'Lines'Quantity
      [Invoices]Lines'Tax Rate:=[orig Customers]Invoices'Lines'Tax Rate
      [Invoices]Lines'Total TNI:=[orig Customers]Invoices'Lines'Total TNI
      [Invoices]Lines'Total TI:=[orig Customers]Invoices'Lines'Total TI
       `go to next line item
      NEXT SUBRECORD([orig Customers]Invoices'Lines)
    End while  `==Line item==
     `Activate link going from [Invoices]Customer Code to [new Customers]Customer Code
    ACTIVATE LINK([Invoices]Customer Code)
     `Save the [Invoices] file record.
    SAVE RECORD([Invoices])
     `Go to the next invoice of this customer
    NEXT SUBRECORD([orig Customers]Invoices)
  End while  `==Invoice==
  NEXT RECORD
End while  `==Customer==
  `remove variables from memory.
CLEAR VARIABLE("v")
```

Working with a scrollable area

Suppose you want to manage documents containing large text fields. That is, you want to be able to enter and modify data and find information on a given topic. A search of the type

SEARCH ([Documents]Text = "@Value@")

takes too much time if you're dealing with large texts and if the file contains a lot of records. A better solution is to have a subfile which contains the keywords related to each text. Figure C-4 shows a documentation database containing such a subfile.

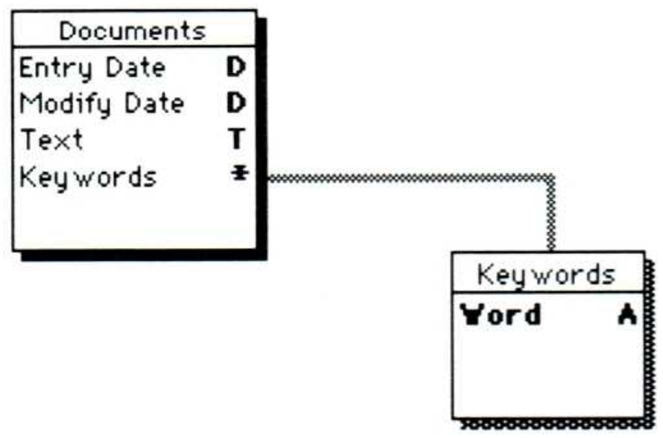


Figure C-4
Documents database

[Documents]Entry Date contains the date on which the record was created. [Documents]Modify Date contains the date on which the record was last modified. [Document]Text contains the entered text. [Documents]Keywords is the subfile and the [Documents]Keywords'Word subfield lets you save the keywords you create. Figure C-5 shows the input layout for the Documents database.

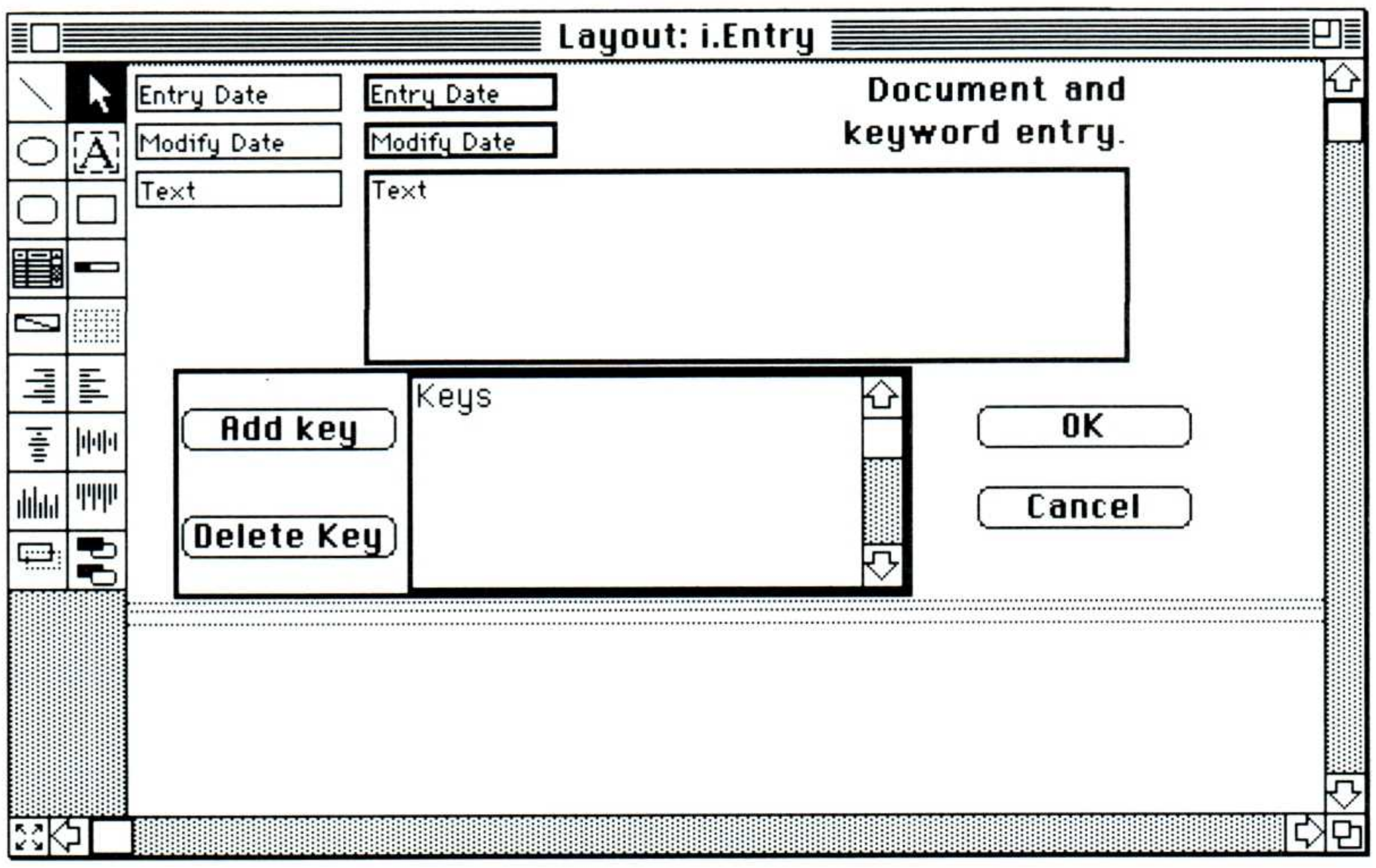


Figure C-5
Documents input layout

In addition to three fields, the layout also contains: a scrollable area named Keys, a plain Add button named bAdd, a plain Delete button named bDel, and because controls have been placed in the layout, an Accept button labeled OK and a Don't Accept button labeled Cancel.

Here's what the input layout procedure must do:

- □ If the user clicks the bAdd button, and the user has highlighted some text in the text field, add the text selection to the list of keywords in the Keys scrollable area.
- □ If the user selects a value in the keywords list contained in the Keys scrollable area, enable the bDel button. Otherwise, disable the bDel button.
- □ Every time an action takes place in the keywords list, automatically update the [Documents]Keywords substructure.

And here's the input layout procedure:

```
If (Before)
     `Before the layout is displayed , check if the entry date is equal to the null
     `date.  That is, is this record being created?
  If (Entry Date=!00/00/00!)
     Entry Date:=Current date   `If new, assign Current date to Entry Date.
  End if
  UpdateRec:=False   `Do not force an update unless a field is modified
     `  or a keywork is added/deleted
     `=== Build the keyword scrollable array ===
     `Select all the subrecords in the [Documents]Keywords subfile.
  ALL SUBRECORDS(Keywords)

  Keys0:=0   `Specify that the Keys list  contains no items... yet.
     `Go through subrecords and copy every keyword to the keys list.
  While (Not(End subselection(Keywords)))
     Keys0:=Keys0+1
     Keys{Keys0}:=Keywords'Word
     NEXT SUBRECORD(Keywords)
  End while

  Keys:=0  `Specify that there is no selected value in the Keys list.
  DISABLE BUTTON(bDel)   `Disable the bDel button.
End if   `Before

If (During)   `During data entry:

  Case of
     : (bAdd=1)   `If the user has clicked the bAdd button.
          `Put the position of the first and last characters of selected text in v1 and v2
       GET HIGHLIGHTED TEXT(Text;v1;v2)
       If (v1#v2)   `If the selection is not empty.
          v:=Substring(Text;v1;(v2-v1))   `Put the resulting selection in v.
          Keys0:=Keys0+1   `Specify that the list contains one more value.
          Keys{Keys0}:=v   `Place the selection in this new value.
             `Create a new subrecord in the [Documents]Keywords subfile.
          CREATE SUBRECORD(Keywords)
          Keywords'Word:=v   `Place the selection in this new subrecord.
          Keys:=0   `Specify that there is no selected value in the Keys list.
          UpdateRec:=True
       End if
```

```
        : (bDel=1)  `If the user clicked the bDel button.
              `Search the subrecord corresponding to the selected value in the Keys list.
          SEARCH SUBRECORDS([Documents]Keywords;[Documents]Keywords'Word=Keys{Keys})
              `Delete that subrecord from the [Documents]Keywords subfile.
          DELETE SUBRECORD([Documents]Keywords)
          i:=Keys  `Place in i the number of the value to be deleted from the keys list.
          While (i<Keys0)  `While i is less than the number of values in the Keys list.
              Keys{i}:=Keys{i+1}  `Copy the contents of the following value to the specified value.
              i:=i+1
          End while
          Keys0:=Keys0-1  `Specify the Keys list contains one value less.
          Keys:=0  `Specify there is no selected value in the Keys list.
          UpdateRec:=True
     End case

  If (Keys#0)
    ENABLE BUTTON(bDel)  `If  a selected value exists in the Keys list,
      `enable the bDel button.
  Else
    DISABLE BUTTON(bDel)  `Else disable the button.
  End if

End if  `During

If (After)
  Modify Date:=Current date  `Assign the Current date to Modify Date.
End if  `After

If (UpdateRec)
  UpdateRec:=False  `Just once is enough
  Entry Date:=Entry Date  `Modify the parent record to force a record save.
End if
```

Here is the global procedure for searching texts:

```
  `SearchText procedure
DEFAULT FILE([Documents])
  `Ask the user to specify the search keyword.
v:=Request("Specify the search keyword";"Macintosh")
If (OK=1)
    `If the user validates the request, search by index for records containing at
    `least one subrecord with the keyword contained in v.
  SEARCH BY INDEX([Documents]Keywords'Word=v)
  If (Records in selection#0)
    OUTPUT LAYOUT ("Output.Text")
    DISPLAY SELECTION(*)
  Else
    ALERT("There is no text on that topic.")
  End if  `Records in selection
End if  `OK=1
```

To search texts, call this procedure by choosing Execute Procedure from the Special menu in the User environment; or, you can set this up as a custom application, adding other features if you wish.

Updating copies of a database

If you make changes to a database, you can update copies of the database without having to use the Design environment and with no loss of data. Thus, you can easily add improvements to any copy of the database.

Here is what you can and cannot do.

- □ You can add, modify, and delete layouts, procedures, indexes, links. (If you modify the indexes or the links, the user must run the utilities after the update to recreate them.)
- □ You can add fields or change attributes of existing fields.
- □ You cannot delete fields.
- □ You cannot delete or add files.

Warning

Do not change the name of the database.

Here are the four steps for updating copies of a database:

1. Make a backup copy of the database.
2. Replace the files databasename.struct, databasename.res, and databasename.enum of the user's database with your own.
3. Leave the following files unchanged: databasename, databasename.data, databasename.index1, databasename.index2, and so on.
4. If you have changed any indexes or links, the user must run the 4th Dimension Tools and repair the indexes and links.

❖ *Note:* If the user has changed a Modifiable Standard Choices list, the changes will be lost, because you're replacing the databasename.enum file. The solution is to write down all user-entered values and re-enter them once the files have been replaced.

Appendix D

External Procedures

You can increase 4th Dimension's power and flexibility by writing external procedures. Using any high-level language like Pascal or C, or 68000 assembly language, you can create procedures that can be called from 4th Dimension. Once compiled and linked, external procedures behave just like built-in 4th Dimension commands.

You can pass parameters and also get data back from your procedures through parameters. External functions, however, are not supported. 4th Dimension calls external procedures, but external procedures cannot call 4th Dimension commands or the 4th Dimension program. The entire code for an external procedure (compiled and linked) cannot exceed one segment (32K).

Parameters

4th Dimension always places the address of the parameters on the stack and calls your external procedure. For this reason, in your procedure, all parameters must be variable parameters. In Pascal, in your procedure definition header, each parameter must have *var* before it. For example:

```
Procedure MyExternal (var theNum: integer);
```

You can pass the parameters listed in Table D-1 to external procedures.

Table D-1
Parameters for external procedures

Parameter	**Type declaration in MPW Pascal**
Integer	integer
Long integer	longint
Real Number	real
String	str255
Text	type
	TERec4D = record
	length: integer;
	text: CharsHandle;
	end;
Date type	type
	Date4D = record
	day: integer;
	month: integer;
	year: integer;
	end;
Picture type	PicHandle (See note below.)

❖ *PicHandle:* This is defined the same as the QuickDraw Picture definition, with a slight change: After the PicSize, PicFrame, and the picture data, there is an offset point (of type point) specifying the location of the picture on the background and then the mode used by 4th Dimension to display the picture.

Creating an external procedure

Here is an example of an external procedure written in MPW Pascal. The example procedure lets you load strings stored in the string list 'STR#' resource.

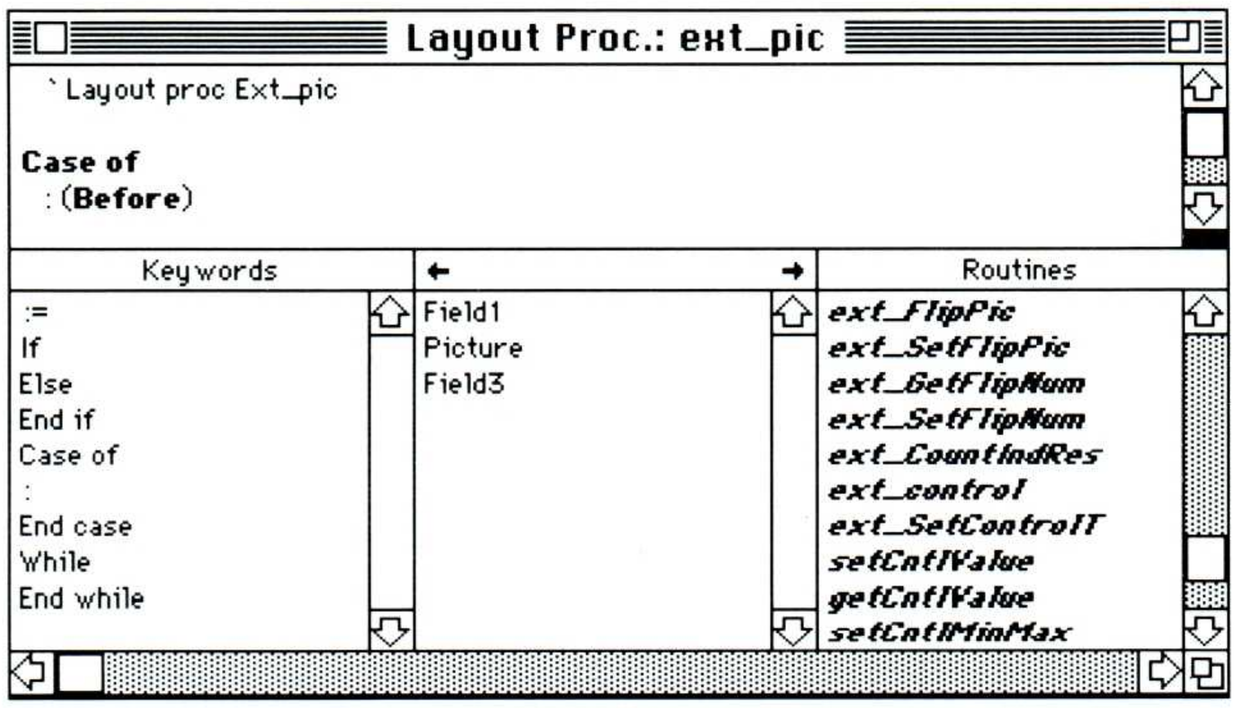
```
program Ext4D_GetIndString;

USES {$LOAD Mydumps}
Memtypes, Quickdraw, OSIntf, ToolIntf;
{$LOAD}
{ The LOAD command speeds up the compilation. }

{$R- }
{$D+ }
```

```
var
    DummyString: str255;
    DummyInt1, DummyInt2: integer;

procedure GetIndStr (var theStr: str255; var ResID, StrNum: integer);
{ All parameters must be var parameters. }

    begin
        GetIndString (theStr, ResID, StrNum);
        if ResError <> NoErr then theStr := 'Error';
    end;

begin
     GetIndStr (DummyString, DummyInt1, DummyInt2);
    { Some compilers do not compile procedures that are not invoked. }
end
```

The next step is to compile and link this program. After this, you need to install the procedure in 4th Dimension. This is done using the 4D Mover program. (See the *4th Dimension Utilities and Developer's Notes.*)

4D Mover copies the CODE segment 1 from your linked code, and pastes it into a 4DEX resource in 4th Dimension, or in the database.res file. If you have compiled and linked your program like an application, then the CODE begins with four bytes for the segment jump table. 4D Mover removes these bytes before pasting the code in the 4DEX resource. If your programming system allows procedure compilation, or if you have instructed the linker not to generate the segment offset, then you need to change ther paste conversion of the CODE resource in the 4D Mover. Check the 4D Mover documentation on how to do this.

After the procedure is installed, the procedure's name appears at the bottom of the list of routines in the procedure editor. External procedures appear in bold italic. Figure D-1 shows how these procedures look in the listing window.



```
  ` Layout proc Ext_pic

Case of
  : (Before)
```

Keywords	← →	Routines
:=	Field1	***ext_FlipPic***
If	Picture	***ext_SetFlipPic***
Else	Field3	***ext_GetFlipNum***
End if		***ext_SetFlipNum***
Case of		***ext_CountIndRes***
:		***ext_control***
End case		***ext_SetControlT***
While		***setCntlValue***
End while		***getCntlValue***
		setCntlMinMax

Figure D-1
Listing showing external procedures

To call the procedure, you simply type the name and pass the parameters needed. If there is a 'STR#' resource with ID = 11 containing 24 strings, you can get the third string in this list by the following 4th Dimension procedure.

```
Str1:=""
GetIndStr(Str1;11,3)
```

To create a 4th Dimension function that gets the string:

```
  ` 4th Dimension function ReadIndStr
$0:=""
GetIndStr($0;$1;$2)
```

To use this function, execute the following statement:

```
vString:= ReadIndStr(11;3)
```

Important details

If the external program has one main procedure and other procedures or functions that are called by the main procedure, you must structure the program as follows:

```
program test;

var dummyInt: integer;
    dummyStr: str255;

procedure UseMe (var theInt: integer); FORWARD;

procedure Main (var theStr: str255; theInt: integer);
    var i: integer;
        theSt: str255;

    begin
        .
        .
        .
        UseMe (i);
        .
        .
        .
    end;

procedure UseMe;
    begin
        .
        .
        .
    end;

begin
    Main (dummyStr, dummyInt);
end.
```

External areas

4th Dimension allows external areas in layouts and dialogs. Using the variables tool in the Layout window, you can draw an area and select External Area in the list of variable types. An external area is essentially a part of the window that is not handled by 4th Dimension. Whenever anything needs to be done in the area (like drawing, updating, or handling mouse events), 4th Dimension calls the external procedure associated with the area. 4th Dimension also passes parameters to the external procedure, which you can use to determine what action to take.

For example, if you create a variable called vNewEmp, assign it the type External Area, and relate the external procedure HandleExtNew to the variable (by typing it in the external procedure name), then the area you have created belongs to the procedure HandleExtNew. When an event related to this area occurs , 4th Dimension calls HandleExtNew to handle the area. There are five events that 4th Dimension passes to the external area procedure, defined as the following constants:

- InitEvt =16
- DeInitEvt =17
- CursorEvt =18
- MouseDown (predefined in the operating system)
- UpdateEvt (predefined in the operating system)

InitEvt, the layout is about to be displayed or printed. The external area procedure must create the data structures it needs in memory, initialize variables, etc.

DeInitEvt, the layout is going to be closed, or the printing of the layout is over. Clear structures created in memory, save documents, etc.

CursorEvt, the user has dragged the mouse over the external area. The procedure should do what it needs, such as change the cursor.

MouseDown, the user has clicked the mouse over the external area. Do the appropriate action, such as invert area or check Stilldown to see if the user is holding the mouse down.

UpdateEvt, the external area needs to be updated. If the external area draws a picture, then this is the time to actually draw the picture.

Every time one of the above event occurs, 4th Dimension calls the external area procedure with the following parameters:

- □ the event record
- □ the external area rectangle
- □ the name of the external area (variable name)
- □ a handle related to the area

The external area procedure must *always* be defined as follows:

```
procedure MyExtArea (var AreaEvent: EventRecord; var AreaRect: Rect; var AreaName: str255;
                               var AreaHndl: MyDataHndl);
```

Important

You must not choose any parameters in 4D Mover for external area procedures.

To interchange data between external procedures and 4th Dimension, use the last parameter (of type Handle). The external procedure name is this handle. In the types declaration, declare the last parameter as a handle to a pointer to any data structure (integer, real, string, array, a record, or any other structure). In the InitEvt part, create the handle using NewHandle (Sizeof (YourDataStructure)). In the DeInit part, dispose of this handle using DisposHandle.

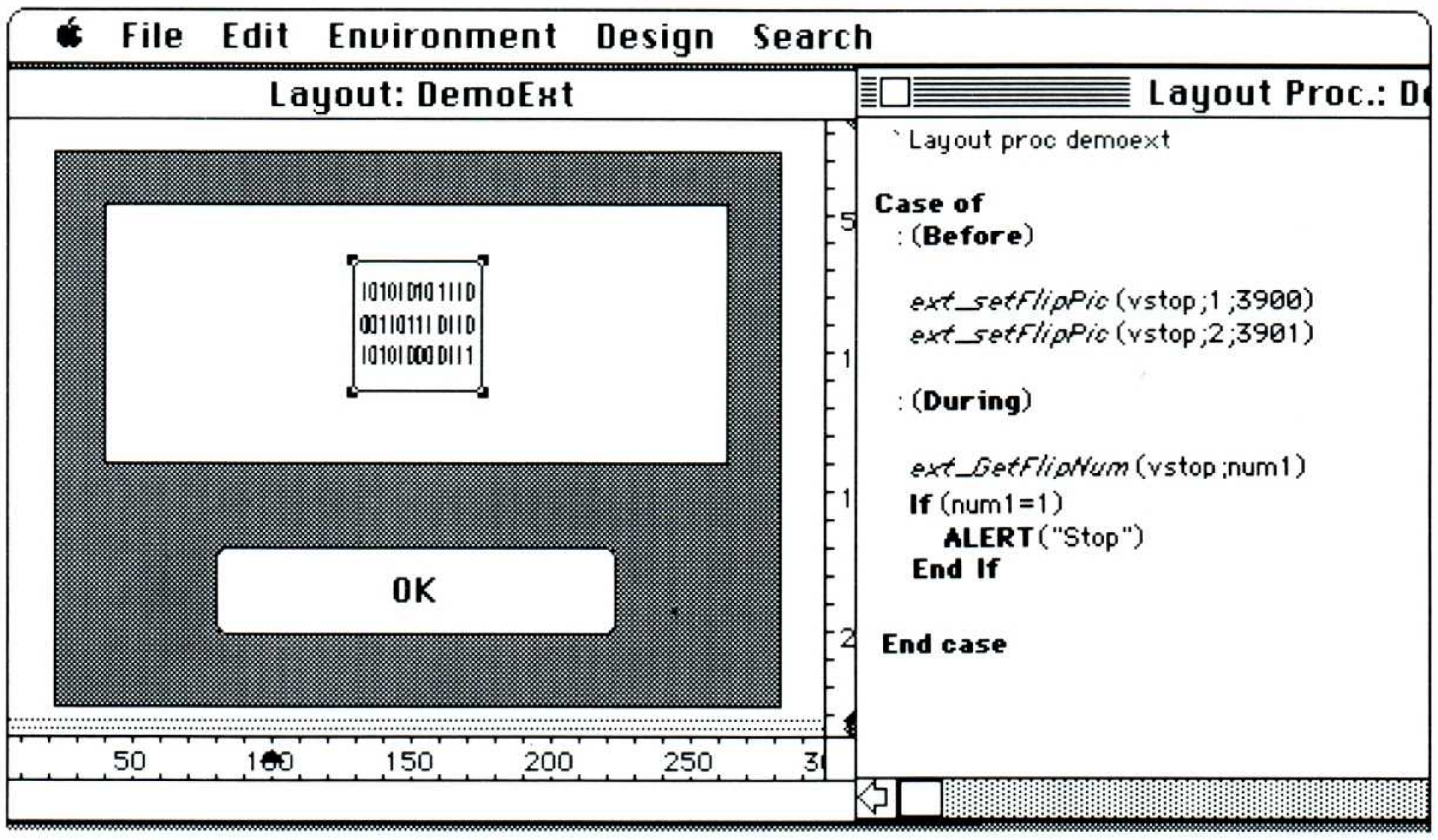
Now, as long as the layout is active, that is, in the Before and During phases, you can pass the handle to your structure by passing the name of the external area to any other external procedure.

If you had a number thenum in the record to which the handle is pointing, then when the user clicks, set thenum to 1. You can now, in the During phase, check to see if the area was clicked by calling an external procedure that takes the handle and integer as parameters, and assigns to the integer the value of thenum. In the 4th Dimension procedure, the integer parameter will contain 1 if the external area was clicked.

A graphic example

The following example shows how to use an external area that contains a picture. When the user clicks on the area, the picture is flipped with another. The pictures are 'PICT' resources in the database.res file, with resource ID equal to 3900 and 3901.

Figure D-2 shows the layout that you need to create to run the example.



```
 ` Layout proc demoext

Case of
  : (Before)

   ext_setFlipPic (vstop;1;3900)
   ext_setFlipPic (vstop;2;3901)

  : (During)

   ext_GetFlipNum (vstop;num1)
   If (num1=1)
      ALERT ("Stop")
   End If

 End case
```

Figure D-2
Layout for external procedure

Figure D-3 shows how the layout will look when it is run from either the User or the Custom environment.

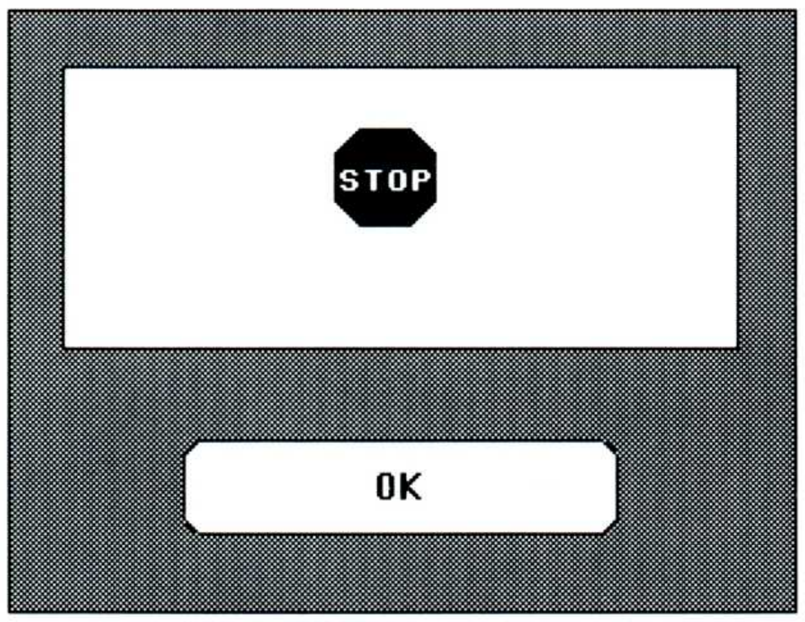


Figure D-3
A result of the external procedure

Figure D-2 shows the layout procedure you must write to call the external procedures. The example uses three procedures:

- □ ext_FlipPic, created in the FlipPic program.
- □ ext_SetFlipPic, created in the SetFlipPic program.
- □ ext_GetFlipNum, created in the FlipPicNum program.

Here is a description of the three procedures.

ext_FlipPic is the external area procedure.

ext_SetFlipPic (theHandle; theInt; theResID) is an external procedure that takes as parameters the external area handle, theInt (an integer having values 1 or 2 denoting the first picture or second) and theResID (the picture resource ID number). This procedure is called in the Before phase, once with theInt = 1, to set the picture that will be displayed first, and again with theInt = 2, with the picture that will be displayed when the mouse is down on the external area.

ext_GetFlipNum (theHandle; theInt) is an external procedure that takes as parameters the external area handle and theInt, which will contain 1 if the picture was clicked on, and 0 if it was not. The procedure resets the value to zero.

The listings for the three programs that create these procedures follow.

FlipPic

Here is the listing for FlipPic.

```
program FlipPic;

Uses
   {$LOAD HD:MPW:Pexamples:MQOTPLOAD}
   Memtypes,Quickdraw,OSIntf,ToolIntf,PackIntf;
   {$LOAD}

const
   InitEvt   = 16;
   DeInitEvt = 17;
   CursorEvt  = 18;

type
   MyData = record
               theNum: integer;
               thePic1: PicHandle; { what will be displayed }
               thePic2: PicHandle; { what will be displayed when mouse down }
            end;

   MyDataPtr = ^MyData;
   MyDataHndl = ^MyDataPtr;

{R-}
{D+}
```

```
var
    MyEvent: EventRecord;
    MyRect: Rect;
    MyName: str255;
    MyHndl: MyDataHndl;

procedure Ext_FlipPic (var AreaEvent: EventRecord; var AreaRect: Rect; var AreaName: str255;
                                var AreaHndl: MyDataHndl);
    var
        i: longint;
        Pt: point;
        hPic: PicHandle;
        rec: Rect;

    begin
        case AreaEvent.what of
            initEvt:
            begin
                AreaHndl := MyDataHndl (NewHandle (sizeof (MyData)));
                AreaHndl^^.theNum := 0;
            end;

            DeInitEvt:
            begin
                DisposHandle (handle (AreaHndl));
            end;

            CursorEvt:
            begin
                SetCursor (GetCursor (10)^^);
            end;

            Mousedown:
            begin
                SetRect (rec, AreaRect.left, AreaRect.top,
                    AreaHndl^^.thePic2^^.picframe.right - AreaHndl^^.thePic2^^.picframe.left + AreaRect.left,
                    AreaHndl^^.thePic2^^.picframe.bottom - AreaHndl^^.thePic2^^.picframe.top + AreaRect.top);
                DrawPicture (AreaHndl^^.thePic2, rec);
                AreaHndl^^.theNum := 1;
                while (StillDown) do { wait for the mouse to be released }
                begin
                end;
                SetRect (rec, AreaRect.left, AreaRect.top,
                    AreaHndl^^.thePic1^^.picframe.right - AreaHndl^^.thePic1^^.picframe.left + AreaRect.left,
                    AreaHndl^^.thePic1^^.picframe.bottom - AreaHndl^^.thePic1^^.picframe.top + AreaRect.top);
                DrawPicture (AreaHndl^^.thePic1, rec);
            end;
            UpdateEvt:
            begin
               SetRect (rec, AreaRect.left, AreaRect.top,
                   AreaHndl^^.thePic1^^.picframe.right - AreaHndl^^.thePic1^^.picframe.left + AreaRect.left,
                   AreaHndl^^.thePic1^^.picframe.bottom - AreaHndl^^.thePic1^^.picframe.top + AreaRect.top);
               DrawPicture (AreaHndl^^.thePic1, rec);
            end;
         end; {case}
   end;

begin
   Ext_FlipPic (MyEvent, MyRect, MyName, MyHndl);
end.
```

SetFlipPic

Here is the listing for SetFlipPic.

```
program SetFlipPic;

Uses
   {$LOAD HD:MPW:Pexamples:MQOTPLOAD}
   Memtypes,Quickdraw,OSIntf,ToolIntf,PackIntf;
   {$LOAD }

type
   MyData = record
               theNum: integer;
               thePic1: PicHandle;   { what will be displayed }
               thePic2: PicHandle;   { what will be displayed when mouse down }
            end;

   MyDataPtr = ^MyData;
   MyDataHndl = ^MyDataPtr;

{R-}
{D+}

var
   MyNum: integer;
   MyHndl: MyDataHndl;
```

```
procedure ext_SetFlipPic (var AreaHndl: MyDataHndl; var WhichOne: integer; var PicID: integer);
  begin
    if WhichOne=1 then
      AreaHndl^^.thePic1 := PicHandle (GetResource ('PICT',PicID))
    else
      if WhichOne=2 then
        AreaHndl^^.thePic2 := PicHandle (GetResource ('PICT',PicID));
  end;

begin {main}
    { Just to compile }
  ext_SetFlipPic (MyHndl, MyNum, MyNum);
end.
```

FlipPicNum

Here is the listing for FlipPicNum.

```
program FlipPicNum;

Uses
  {$LOAD HD:MPW:Pexamples:MQOTPLOAD}
  Memtypes,Quickdraw,OSIntf,ToolIntf,PackIntf;
  {$LOAD }

type
  MyData = record
              theNum: integer;
              thePic1: PicHandle;
              thePic2: PicHandle;
           end;

  MyDataPtr = ^MyData;
  MyDataHndl = ^MyDataPtr;

{R-}
{D+}

var
  MyNum: integer;
  MyHndl: MyDataHndl;

procedure ext_FlipPicNum (var AreaHndl: MyDataHndl; var theVar: integer);
  begin
    theVar := AreaHndl^^.theNum;
    AreaHndl^^.theNum := 0;
  end;

begin {main}
    { Just to compile }
  ext_FlipPicNum (MyHndl, MyNum);
end.
```

External areas and the execution cycle

The following is an overview of the execution cycle for an external area when it is displayed on the screen or printed.

1. The external area procedure is called with event.what = InitEvt.
2. Before is set to true and the layout procedure is called.
3. The external area procedure is called with event.what = UpdateEvt.
4. Wait for the next event.

   If the next event is a mouse down in the external area, then the external area procedure is called with event.what = MouseDown.

   If the next event is an update event in the external area, then the external area procedure is called with event.what = UpdateEvt.

   Otherwise, if the event is a null event and the mouse is over the external area, then the external area procedure is called with event.what = CursorEvt.
5. Normal processing of events is done by 4th Dimension.
6. If the layout is validated or cancelled, then the external area procedure is called with event.what = DeInitEvt.
7. If the layout was an input layout, and the user validated it, then After is set to true and the layout procedure is called.

Appendix E

System Variables and the System Set

This appendix discusses 4th Dimension's system variables and the system set, UserSet. 4th Dimension stores system variables in memory. These variables contain information on the execution process and user's actions.

The OK variable

OK is the most commonly used of the system variables. In general, OK contains a 1 when the user clicks an Accept button. It returns a 0 when the user clicks a Don't Accept button, when the user halts an activity like sorting or searching, and when an error occurs. OK contains a 2 when the user clicks the Delete button in the standard button palette. Table E-1 summarizes the affect that various 4th Dimension commands have on OK.

Table E-1
OK variable values

Command	Value of OK variable
ADD RECORD	1 if user validates; otherwise 0.
ADD SUBRECORD	1 if user validates; otherwise 0.
APPLY TO SELECTION	1 if operation completed; 0 if user clicks Cancel in progress window.
CONFIRM	1 if user clicks OK button; otherwise 0.
DIALOG	1 if user validates; otherwise 0.
EXPORT DIF	1 if operation completed; 0 if user clicks Cancel in progress window.
EXPORT SYLK	1 if operation completed; 0 if user clicks Cancel in progress window.

Table E-1 (continued)
OK variable values

Command	Value of OK variable
EXPORT TEXT	1 if operation completed; 0 if user clicks Cancel in progress window.
IMPORT DIF	1 if operation completed; 0 if user clicks Cancel in progress window.
IMPORT SYLK	1 if operation completed; 0 if user clicks Cancel in progress window.
IMPORT TEXT	1 if operation completed; 0 if user clicks Cancel in progress window.
MODIFY RECORD	1 if user validates; otherwise 0.
MODIFY SUBRECORD	1 if user validates; otherwise 0.
PRINT DIALOG	1 if printing completed; 0 if user clicks Cancel in either one of the two standard print dialog boxes or in the printing progress window.
PRINT LABEL	1 if printing completed; 0 if user clicks Cancel in either one of the two standard print dialog boxes or in the printing progress window.
PRINT SELECTION	1 if printing completed; 0 if user clicks Cancel in either one of the two standard print dialog boxes or in the printing progress window.
PRINT SETTINGS	1 if user clicks OK in both standard print dialog boxes; otherwise 0.
REPORT	1 if printing completed; 0 if user clicks Cancel in either one of the two standard print dialog boxes or in the printing progress window.
Request	1 if user clicks OK button; otherwise 0.
SEARCH	1 if operation completed; otherwise 0.
SEARCH BY INDEX	(In search window) 1 if user validates search; otherwise 0.
SEARCH SELECTION	1 if operation completed; otherwise 0.
SORT BY INDEX	1 if sorting completed; 0 if user clicks Cancel in progress window.
SORT SELECTION	1 if sorting completed; 0 if user clicks Cancel in standard sort window or in progress window.
SET CHANNEL	1 if the user clicks Open or Save; otherwise 0.

The Document variable

The Document variable contains the name of the Macintosh document that was opened last or created last with one of the following procedures:

EXPORT DIF, EXPORT SYLK, EXPORT TEXT, IMPORT DIF, IMPORT SYLK, IMPORT TEXT, LOAD SET, LOAD VARIABLE, SAVE SET, SAVE VARIABLE, SET CHANNEL, USE ASCII MAP.

The FldDelimit variable

The FldDelimit variable contains the ASCII code of the character which delimits fields. This character is used when exporting or importing data in Text format. 4th Dimension default FldDelimit variable is set to 9, the ASCII code for the TAB character.

The RcdDelimit variable

The RcdDelimit variable contains the ASCII code of the character which delimits records. This character is used when exporting or importing data in Text format. 4th Dimension default RcdDelimit variable is set to 13, the ASCII code for the carriage return character.

The Error variable

The Error variable works only in the stop procedure installed with ON ERR CALL. This variable contains the error code. Appendix H lists all 4th Dimension error codes and SANE NaN codes.

The Flush variable

The Flush variable controls what gets saved to disk and when.

Warning

Do not use this variable. You could cause a loss of data.

Event variables

Event variables record the results of keyboard and mouse events. They include

- □ MouseDown
- □ KeyCode
- □ Modifiers

The MouseDown, KeyCode, and Modifiers variables only work in procedures installed by the ON EVENT CALL command.

The MouseDown variable contains 1 if the mouse button was down when the event occured. Otherwise it contains 0.

This KeyCode variable contains the ASCII code of the character that was being typed if the event was a key stroke.

The Modifiers variable contains value indicating whether one or more of following keys was down when an event occured: Control, Option, Shift, Caps Lock. Here are two procedures that return these codes. The first procedure, ModLook, installs the ON EVENT CALL procedure, Event. Event, in turn, reports which modifier(s) were down.

❖ *Note:* When running this program, you must press the modifier key and a non-modifier key.

```
 `ModLook
 `Calls ON EVENT CALL to test Modifiers
OK:=1
ON EVENT CALL("Event")
i:=1
While (i<200)  `Something to do
 MESSAGE(String((Random/32767)-(Random/32767)))
 i:=i+1
End while
ON EVENT CALL("")
```

```
  `Event
  `Reads Modifiers system variable.
  `Doesn't bother with other keycodes.
If (Modifiers#0)
  m:=Int(Modifiers/256)
  Case of
    : (m=1)
      s:="Command"
    : (m=2)
      s:="Shift"
    : (m=3)
      s:="Shift-Command"
    : (m=4)
      s:="CapsLock"
    : (m=8)
      s:="Option"
    Else
      s:="This key combination"  `You can add the rest, through 15
  End case
  ALERT(s+" has a Modifiers value of "+String(Modifiers))
End if
```

The system set: UserSet

4th Dimension creates a set named UserSet when a user selects records in the User environment or when the user selects one or more records under the DISPLAY SELECTION or MODIFY SELECTION. The 4th Dimension set commands work with UserSet. Here is an example:

```
DEFAULT FILE([Accounts])
ALL RECORDS
DISPLAY SELECTION
CREATE EMPTY SET("MyPicks")
UNION("UserSet";"MyPicks";"MyPicks")
USE SET("MyPicks")
PRINT SELECTION
CLEAR SET("MyPicks")
```

UserSet is not available to the CLEAR SET command.

Appendix F

4th Dimension and Macintosh Codes

This appendix contains information on

- □ serial port codes for SET CHANNEL
- □ desktop document codes for SET CHANNEL
- □ Macintosh font codes
- □ Macintosh style codes
- □ Macintosh pathnames

Serial port codes

These codes apply to the SET CHANNEL procedure in its telecommunications syntax:

SET CHANNEL (*posintexpr1;posintexpr2*)

Table F-1 describes arguments for *posintexpr1,* and Table F-2 arguments for *posintexpr2.*

Table F-1
Serial port first argument settings

Object	Setting	Argument
Port	Printer	0
	Modem	1
Protocol	XON/XOFF printer	20
	XON/XOFF modem	21
	DTR printer	30
	DTR modem	31

Table F-2
Serial port second argument settings

Object	Setting	Argument
Speed (in baud)	300	380
	600	189
	1200	94
	1800	62
	2400	46
	3600	30
	4800	22
	7200	14
	9600	10
	19200	4
	57000	0
Data bits	8	3072
Stop bits	1	16384
	1.5	–32768
	2	–16384
Parity	No	0
	Odd	4096
	Even	12288

Desktop document codes

These codes apply to the SET CHANNEL procedure in its document syntax:

SET CHANNEL (*posintexpr*;*strexpr*)

Table F-3 describes arguments for the SET CHANNEL procedure in its document syntax. All commands apply only to text files.

Table F-3
SET CHANNEL document arguments

posintexpr	*strexpr*	Result
10	[filename]	Opens document named *filename*. If the file doesn't exist, these arguments create and open the file. If you don't include the name, the standard open file dialog appears.
11	None	Closes the current document.
12	None	Opens a text file.
13	None	Creates a new file. These arguments bring up the standard open file dialog so you can name the file. After naming, the file opens.

Macintosh font codes

The codes in this section are arguments for the FONT command. Its syntax is

FONT *(var;posintexpr| strexpr)*

You must give either *posintexpr* or *strexpr,* but not both.

Table F-4
FONT arguments

posintexpr	*strexpr*	*posintexpr*	*strexpr*
0	systemFont	9	toronto
1	applFont	11	cairo
2	newYork	12	losAngeles
3	geneva	20	times
4	monaco	21	helvetica
5	venice	22	courier
6	london	23	symbol
7	athens	24	taliesin
8	sanFran		

systemFont is Chicago, the font used by the system for drawing menu titles and commands in menus. *applFont* is the application font and the default font. For more information on fonts and font numbering, see *Inside Macintosh*.

Macintosh style codes

The codes in this section are arguments for the FONT STYLE command. Its syntax is

FONT STYLE *(var;posintexpr)*

Macintosh stores its style settings in a low-order byte. The bit(s) set to ON determine character style. By adding decimal equivalents together, you can create multi-style combinations, like ***bold italic.*** Plain text appears when all bits are off.

Table F-5
FONT STYLE arguments

Bit on	Decimal	Style	Bit on	Decimal	Style
0	1	Bold	4	16	Shadow
1	2	Italic	5	32	Condensed
2	4	Underline	6	64	Extended
3	8	Outline			

Macintosh pathnames

Table F-6 gives a brief summary of Macintosh pathnames. For more detailed discussion, see *Inside Macintosh.*

Table F-6
Macintosh pathnames

Syntax	**Example**	**Explanation**
strexpr	MyFile	*strexpr* with no colon means a filename in the current directory.
:*strexpr*	:MyFolder	*strexpr* with a preceding colon means a folder in the current directory.
strexpr1:*strexpr2*:*strexprn*	MyDisk:Tasks:ToDo:Today	*strexpr1* is a volume name. *strexpr2*: through *strexprn-1*: are folder names. *strexprn*: is a filename.

Appendix G

ASCII Codes

This appendix consists of two tables. Table G-1 presents the standard ASCII codes. Table G-2 presents the extended Macintosh character set for the Helvetica font.

Table G-1
Standard ASCII codes

Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex
NUL	0	0	0	sp	32	40	20	@	64	100	40	`	96	140	60
SOH	1	1	1	!	33	41	21	A	65	101	41	a	97	141	61
STX	2	2	2	"	34	42	22	B	66	102	42	b	98	142	62
ETX	3	3	3	#	35	43	23	C	67	103	43	c	99	143	63
EOT	4	4	4	$	36	44	24	D	68	104	44	d	100	144	64
ENQ	5	5	5	%	37	45	25	E	69	105	45	e	101	145	65
ACK	6	6	6	&	38	46	26	F	70	106	46	f	102	146	66
BEL	7	7	7	'	39	47	27	G	71	107	47	g	103	147	67
BS	8	10	8	(	40	50	28	H	72	110	48	h	104	150	68
HT	9	11	9	)	41	51	29	I	73	111	49	i	105	151	69
LF	10	12	A	*	42	52	2A	J	74	112	4A	j	106	152	6A
VT	11	13	B	+	43	53	2B	K	75	113	4B	k	107	153	6B
FF	12	14	C	,	44	54	2C	L	76	114	4C	l	108	154	6C
CR	13	15	D	-	45	55	2D	M	77	115	4D	m	109	155	6D
SO	14	16	E	.	46	56	2E	N	78	116	4E	n	110	156	6E
SI	15	17	F	/	47	57	2F	O	79	117	4F	o	111	157	6F
DLE	16	20	10	0	48	60	30	P	80	120	50	p	112	160	70
DC1	17	21	11	1	49	61	31	Q	81	121	51	q	113	161	71
DC2	18	22	12	2	50	62	32	R	82	122	52	r	114	162	72
DC3	19	23	13	3	51	63	33	S	83	123	53	s	115	163	73
DC4	20	24	14	4	52	64	34	T	84	124	54	t	116	164	74
NAK	21	25	15	5	53	65	35	U	85	125	55	u	117	165	75
SYN	22	26	16	6	54	66	36	V	86	126	56	v	118	166	76
ETB	23	27	17	7	55	67	37	W	87	127	57	w	119	167	77
CAN	24	30	18	8	56	70	38	X	88	130	58	x	120	170	78
EM	25	31	19	9	57	71	39	Y	89	131	59	y	121	171	79
SUB	26	32	1A	:	58	72	3A	Z	90	132	5A	z	122	172	7A
ESC	27	33	1B	;	59	73	3B	[	91	133	5B	{	123	173	7B
FS	28	34	1C	<	60	74	3C	\	92	134	5C	|	124	174	7C
GS	29	35	1D	=	61	75	3D	]	93	135	5D	}	125	175	7D
RS	30	36	1E	>	62	76	3E	^	94	136	5E	~	126	176	7E
US	31	37	1F	?	63	77	3F	_	95	137	5F	del	127	177	7F

Table G-2
Extended Macintosh character set (Helvetica)

Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex
Ä	128	200	80	†	160	240	A0	¿	192	300	C0	‡	224	340	E0
Å	129	201	81	°	161	241	A1	¡	193	301	C1	·	225	341	E1
Ç	130	202	82	¢	162	242	A2	¬	194	302	C2	‚	226	342	E2
É	131	203	83	£	163	243	A3	√	195	303	C3	„	227	343	E3
Ñ	132	204	84	§	164	244	A4	ƒ	196	304	C4	‰	228	344	E4
Ö	133	205	85	•	165	245	A5	≈	197	305	C5	Â	229	345	E5
Ü	134	206	86	¶	166	246	A6	∆	198	306	C6	Ê	230	346	E6
á	135	207	87	ß	167	247	A7	«	199	307	C7	Á	231	347	E7
à	136	210	88	®	168	250	A8	»	200	310	C8	Ë	232	350	E8
â	137	211	89	©	169	251	A9	…	201	311	C9	È	233	351	E9
ä	138	212	8A	™	170	252	AA		202	312	CA	Í	234	352	EA
ã	139	213	8B	´	171	253	AB	À	203	313	CB	Î	235	353	EB
å	140	214	8C	¨	172	254	AC	Ã	204	314	CC	Ï	236	354	EC
ç	141	215	8D	≠	173	255	AD	Õ	205	315	CD	Ì	237	355	ED
é	142	216	8E	Æ	174	256	AE	Œ	206	316	CE	Ó	238	356	EE
è	143	217	8F	Ø	175	257	AF	œ	207	317	CF	Ô	239	357	EF
ê	144	220	90	∞	176	260	B0	–	208	320	D0		240	360	F0
ë	145	221	91	±	177	261	B1	—	209	321	D1	Ò	241	361	F1
í	146	222	92	≤	178	262	B2	“	210	322	D2	Ú	242	362	F2
ì	147	223	93	≥	179	263	B3	”	211	323	D3	Û	243	363	F3
î	148	224	94	¥	180	264	B4	‘	212	324	D4	Ù	244	364	F4
ï	149	225	95	µ	181	265	B5	’	213	325	D5	ı	245	365	F5
ñ	150	226	96	∂	182	266	B6	÷	214	326	D6	ˆ	246	366	F6
ó	151	227	97	∑	183	267	B7	◊	215	327	D7	˜	247	367	F7
ò	152	230	98	∏	184	270	B8	ÿ	216	330	D8	¯	248	370	F8
ô	153	231	99	π	185	271	B9	Ÿ	217	331	D9	˘	249	371	F9
ö	154	232	9A	∫	186	272	BA	⁄	218	332	DA	˙	250	372	FA
õ	155	233	9B	ª	187	273	BB	¤	219	333	DB	˚	251	373	FB
ú	156	234	9C	º	188	274	BC	‹	220	334	DC	¸	252	374	FC
ù	157	235	9D	Ω	189	275	BD	›	221	335	DD	˝	253	375	FD
û	158	236	9E	æ	190	276	BE	ﬁ	222	336	DE	˛	254	376	FE
ü	159	237	9F	ø	191	277	BF	ﬂ	223	337	DF	ˇ	255	377	FF

Appendix H

Error Messages

This appendix contains two tables of error messages. Table H-1 lists 4th Dimension errors, and Table H-2 lists SANE's NaN errors. Any negative error numbers are Macintosh system errors. Consult *Inside Macintosh,* Volume 4.

4th Dimension error messages

Table H-1
4th Dimension error messages

Error number	Message
1	"(" expected.
2	A field name was expected.
3	This command may be executed only on a field in a subfile.
4	Parameters in the list must all be of the same type.
5	There is no file to apply this command to.
6	This command may be executed only on a field of type Subfile.
7	A numeric argument was expected.
8	An alphanumeric argument was expected.
9	The result of a conditional test was expected.
10	This command can't be applied to this field type.
11	This command can't be applied between two conditional tests.
12	This command can't be applied between two numeric arguments.
13	This command can't be applied between two alphanumeric arguments.
14	This command can't applied between two date arguments.
15	This operation is not compatible with the two arguments.

Table H-1 (continued)
4th Dimension error messages

Error number	**Message**
16	This field has no link.
17	A filename was expected.
18	The field types are incompatible.
19	This field is not indexed.
20	"=" was expected.
21	This procedure does not exist.
22	The fields must belong to the same file or subfile for a sort or graph.
23	"<" or ">" was expected.
24	";" was expected.
25	There are too many fields for a sort.
26	The field type must be Alpha, Date, or Numeric.
27	The field must be prefixed by its filename.
28	The field type must be Numeric.
29	The value must be 1 or 0.
30	A variable was expected.
31	There is no menu bar with this number.
32	A date was expected.
33	Unimplemented command or function.
34	Ledger not open.
35	The sets are from different files.
36	The file is bad.
37	":=" was expected.
39	The set does not exist.
40	This is a function, not a procedure.
41	A variable or field belonging to a subfile was expected.
42	The record can't be pushed onto the stack.
43	The function can't be found.
44	The procedure can't be found.
45	A field or variable was expected.
46	A numeric or an alphanumeric argument was expected.
47	The field type must be Alpha.
48	Syntax error.
49	This operator can't be used here.
50	These operators can't be used together.

NaN messages

NaN stands for *Not a Number.* It is a Standard Apple Numeric Environment (SANE) representation and appears when an operation produces a result that is beyond SANE's scope.

Table H-2
SANE NaN messages

NaN code	Reason
1	Invalid square root
2	Invalid addition
4	Invalid division
8	Invalid multiplication
9	Invalid remainder
17	Converting an invalid ASCII string
20	Converting a Comp type number to floating-point
21	Creating a NaN with a zero code
33	Invalid argument to a trig function
34	Invalid argument to an inverse trig function
36	Invalid argument to a log function
37	Invalid argument to an x^i or x^y function
38	Invalid argument to a financial function
255	Uninitialized storage

Index

ACTUS