MINISTRY OF AGRICULTURE AND FISHERIES

TE MANATU AHUWHENUA AHUMOANA

MAF

Infomate

Version 2.0

A data capture program for researchers.

Infomate

Version 2.0

By R.B. Jordan

of the

Engineering Development Group

RUAKURA

Contact Address: Engineering Development Group
MAF, Ruakura Agricultural Centre
Hamilton, New Zealand

Telephone: (0064) (07) 8385678
Facsimile: (0064) (07) 8385655

Copyright Notice

Disclaimer

Products and Trademarks mentioned

Infomate and Infomate Manual production

Infomate was written using Z88 BBC BASIC and it's Assembler, under the BAGZ Development system.

This manual was prepared on an Apple Macintosh computer using Word 4.0 and MacDraw II, and was printed on an Apple Laserwriter II.

INDEX

PREFACE

Rationale and history

Infomate has evolved from a series of programs that were developed at Ruakura during the 1980's which owe their existence to the advent of the personal computer. The strongest influence on Infomate was the Apple II computer for which a number of programs were developed to collect research data particularly at the Ruakura Agricultural Centre. In the last decade these programs have taught us a lot about data capture and the requirements that researchers place upon it. A number of points are worth collecting together on the philosophies required of data collection. These may appear as corollaries of Murphies Law, or as I prefer to call it "The Law of Cussedness"

1. If a program to perform a specific data capture task is perfect today, then by tomorrow the task will be so different that the program will no longer be perfect.

2. It is easy to write a program to do exactly what a researcher says he wants it to do. The difficulty is to write it to solve all of the "unimportant" problems that apparently never occur, and which will render the program requested as useless unless they are solved.

3. If an error situation can occur, it will, and the probability of correcting it is inversely proportional to the time before the error is detected.

4. The further that a person correcting errors is removed from the error creating situation, the lower the probability of the error being correctly corrected. Similarly the probability of correcting errors that are in fact not errors will increase when more people are involved.

5. People who say they collect data and people who actually collect data are often from mutually exclusive groups. A corollary of this is that people who know about data collection and its problems are those who actually do it, and not those who say they do.

6. It takes 95% of the effort to write a program to solve the problems that occur less than 5% of the time during data collection. The other 5% is easy and can be learnt by going to a 4 week high school programming course.

One could go on in this vein, but to summarise, it has been more than adequately shown that there is a need for a data collection program that puts all of the required power and flexibility of a portable computer system into the hands of the person who collects data. This is very different from putting a powerful and flexible computer in the hands of that data collecting person, because without that software the computer might as well not be there. The data collection system must be one that allows him or her to change their minds about what is to be collected, what form it will take, and in what order to collect it. In addition it has to be easy to use by people who are data literate but who may not be particularly computer literate. By this I mean people who know exactly what their data is to look like but, may never be able to write a program to collect it.

Probably the most significant data capture package to be used within MAF was the Apple II Recording System developed by Willy Booth. This was used all over New Zealand for capturing animal weights, produce weights and a variety of other miscellaneous pieces of data. Its strength was that it offered a full solution to many of MAF's smaller research station computer needs in terms of not only data capture, but data analysis as well. Unfortunately this program suite fell down at the upper end of the demands that were made of it because it was just not designed for such large tasks. Nevertheless it allowed a lot of groups to get into computer data capture and fault correction.

Other program suites developed, used the recording system as a basis for the data that they collected, the lamb data recording programs used at Rotomahana and Hopuhopu being a good example. Further programs were produced to capture data from kiwifruit graders, and from the Ruakura research abattoir. These all used some degree of user "programmability", a concept which has proven itself time and again as essential in a research environment.

When the Z88 arrived on the scene, Kevin O'Donnell recognised the potential of this lightweight computer with it's full sized keyboard and word processing software. It would be a good platform for an all-embracing data capture program, which coupled with optional weather-proof cases would handle within reason the full range of data capture tasks required by researchers, and handle it in the environment that the researcher often has to work. With this aim in mind Infomate evolved, and was released in July 1990 to about 40 researchers to evaluate and test. From this valuable phase came a large number of user inspired improvements which have now been incorporated into this version of the package.

The name Infomate is an amalgam of **Information** (knowledge communicated or gained through research - data that can be coded by a computer - what is told), or perhaps **inform** (to fill - inspire - communicate some knowledge) and **mate** (comrade - companion - workfellow - a second in command - habitual associate). These all have the connotations that the Infomate designers had of the package that has been developed.

It was anticipated that the Infomate package would find use in a wide range of data capture tasks, including:

1. Collection of live animal weights from scales and perform all of the validity checks that animal researchers expect during weight capture.
2. Recording produce weights and miscellaneous pieces of data of interest to horticulturists and agronomists and the like. Their requirements change often with the development of the research they perform.
3. Counting objects that are observed in random order such as is often performed with the aid of a microscope.

4. Recording the weight or size of objects such as fruit. The actual weights of a specific fruit has no importance, but the distribution of these sizes provides an important measure of overall fruit production.

5. Observing the timing of events and recording observations about the events such as is used in animal behaviour studies.

6. Recording the intricate data of the birth of young animals to show the relationship between mother and offspring as well as to start a new data set for the newly arrived individual.

7. Logging or recording electrical signals unattended. This allows indirectly the monitoring of such variables as temperatures, relative humidity and more, and is made possible on the Z88 using the ADAM interface.

It goes without saying that in addition to handling the tasks above, the package must also allow the operator to interact with the data that has been collected and correct or inspect that data as part of the research process, or just as part of the data verification process.

Now, to develop a program that has such a broad aim in its usages, provides a maximum of user friendliness to those who are perhaps not particularly computer literate, might appear an impossible task. We believe that Infomate approaches this aim. It is up to the user to decide this for himself. The reality of the matter is that Infomate version 1.0 did not reach that level, but showed the potential of the approach. Version 2.0 of Infomate comes a little closer to achieving that aim by adding many new features, tidying up and simplifying many old features, and further extending the whole concept without going outside the original aims.

Acknowledgements.

Obviously the development of a program such as Infomate hangs heavily on a few people, but it is the interest and enthusiasm of many others that make it all possible. In particular the unfailing enthusiasm of Kevin O'Donnell cannot go unmentioned. He has pushed hard the concept of the Z88, its HARDcase and Litecase housings and the all-purpose all-doing computer program Infomate. Without his efforts Infomate would not have been born. Many others have, perhaps unwittingly, supported this package. Willy Booth through his Apple Recording System provided a bench-mark to which all concepts could be compared. His influence is therefore strong. Many members of the Ruakura and other campuses, and of the Engineering Development Group in particular have put up or knocked down ideas or requirements that have affected the way that Infomate has evolved. Peter Schaare deserves special mention here. To all these people go my thanks. To Paul Gaastra who has looked after software for TIME and all of the external devices that connect to Infomate goes my special thanks. To the Ruakura artist Ken Young who created the drawings that do not use straight lines, and which make the pages of text a little less tedious, I give

special thanks. Finally I thank all of the researchers who nursed Infomate through its early days and patiently suffered the bugs and limitations of the early versions.

Other useful books

Obviously this Infomate documentation will never provide all that the Infomate user will need. Other books that should be used in addition to the manual provided with the computer are listed below. The new user should not be without

"Z88 Magic" by Vic and Gill Gerhardi and Andy Berry. Published by Kuma Computers Ltd.

This is probably the best Z88 reference available for the normal user, and provides lots of easily read examples of usage patterns.

If you are going to use the internal Z88 BASIC, the following book provides a full description of Richard Russell's particularly good rendition of BASIC and is indispensable

"The BBCBASIC(Z80) Reference Manual for the Z88" by Douglas J. Mounter. Published by M-TEC Computer Services (UK)

For details of the Z88 internals and machine code the book

"Z88 Developers' Notes" by John Harrison and Matthew Elton and Published by Cambridge Computers Ltd

is the essential aid to the serious developer but offers little for the non technical reader.

Finally to all users of the Z88, membership of the Z88 Users Club is strongly recommended. This provides access to regular newsletters, to an excellent program library, and in addition, to a large list of names of experienced Z88 users who have offered their services as free help to other Z88 users. Club membership forms are provided with this manual, or can be obtained directly from

Z88 Users Club c/o Roy Woodward
68 Wellington St Long Eaton,
Nottingham NG10 4NG United Kingdom.

Introduction

The Infomate documentation is provided in two parts. The first part is a more readable step by step introduction to Infomate and associated concepts that a person unfamiliar with both the Z88 and Infomate may work through and hopefully gain sufficient information to actually prepare and use Infomate with some proficiency. In the latter part of the manual is a reference section that lists all of the parts of Infomate in an order where it can be found for those who have developed the necessary skills and need to know the details of the structure or usage of a particular part.

THE TUTORIAL SECTION

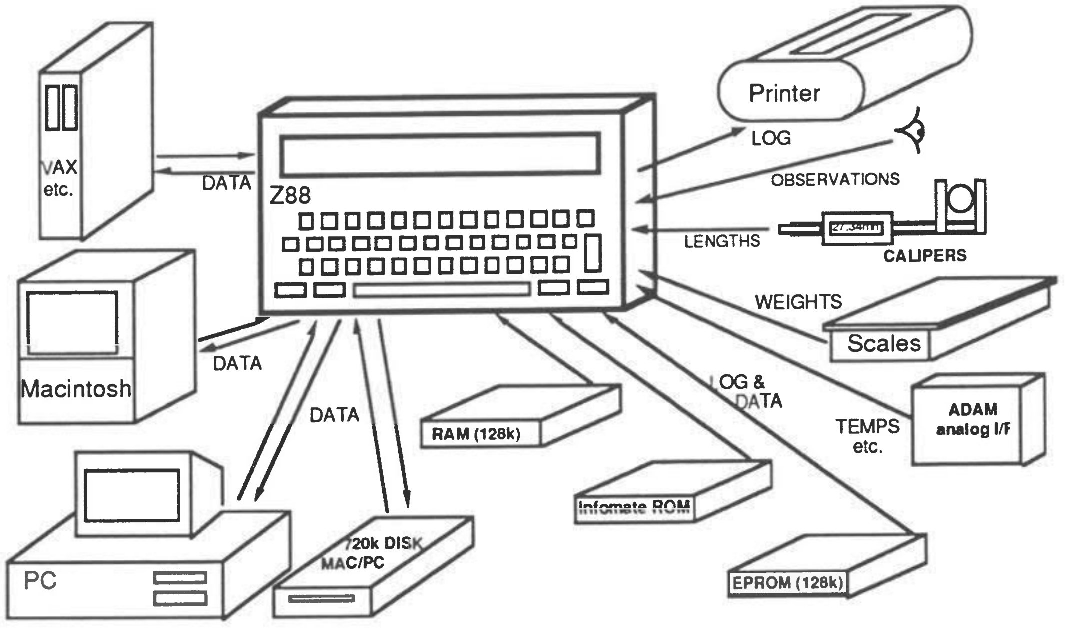
In this section of the manual the user will be taken on a rather meandering journey through Infomate and will learn, using hands on techniques, how to use the package to its fullest.

For **those of you who cannot wait** to give Infomate a trial run, move to the section below called **"the bits you need now"** to make sure that you have all of the required components. Then if necessary work through the list of instructions on how to install those bits. Finally work through the section "a quick example for those who cannot wait". If you have difficulty with this you may just have to work more fully through the manual.

For **those who can wait,** I recommend that you work through the manual step by step trying the examples provided. Most of them rely on what you have typed before and therefore it should not require a lot of typing to go from section to section. As indicated, this is a tutorial section and you will gain most from it by reading it while working on the Z88.

The Infomate Environment

Below is a schematic showing the Z88 and the Infomate ROM and all of the things that can sit around it. This can be used when reading the next section as a sort of map.



A note on dots and things

There is a potential confusion when writing some parts of this manual and it is worth noting early. Some of the expressions in Infomate contain a full stop, and sometimes these fall at the end of a sentence. When this happens the full stop at the end of the sentence can often be mistaken for a part of the language. To avoid this I often postpone the full stop a little like this↷. So if I was talking about the difference between "N2" and "N2." I might say something like - in an ID field never use N2. always use N2 . Read that last dot as a part of English, not of Infomate.

What you need

Before you can use Infomate you must collect together a few essential components. These can be divided into three groups; those that are essential immediately, those that you will probably need sometime, and those that you may need at some other time. Lets look at

The bits you need now:

One Z88; Infomate currently runs only on the Z88. The Z88 must contain **extra memory in slot 1,** and a **128k RAM is recommended. An Infomate ROM module is needed in** slot 2 or slot 3, but **slot 2 is preferred.** To complete this list you should also have **a Z88 manual and an Infomate manual.** Your Z88 will also need some **batteries.** The recommended types are the alkaline penlites (size AA) and you should have a **spare set of batteries** as well.

It is **strongly recommended** that you **use EPROMs to log important data** while you get familiar with the Z88 and build confidence with Infomate

Time is one other thing you will need. You must be prepared to invest a little time in learning Infomate and the Z88. **It is not difficult**, but there are a lot of options and you will need to sort out which ones are important to you. This will take a little time but you will end up with a new tool in your personal data capture repertoire, one which you may find to be surprisingly useful.

The bits you may need later:

A larger computer to send your data to. The Z88 should not be seen as a stand alone computer, and is probably not the computer to do your data analysis on. It performs best as a removable peripheral to a larger computer. In addition you will need **a cable to connect your Z88 to the larger computer**, because computers cannot speak to each other telepathically yet. The larger computer will probably need to have **a program to allow it to talk to the Z88**. If your bigger computer is a Macintosh then you will need Z88Link. If it is a PC or compatible, then you will need PCLink II or preferably RangerLink. If you talk to a VAX then you may need Kermit, or a version of Import/Export to be developed.

A mains adapter will allow you to run your Z88 from the mains power.

A few EPROM's will almost invariably be needed when you start to capture data so you can store your files or log your data into a non-volatile form of storage. They can be 32k byte, 128k byte or larger units depending how much data you will be collecting. If you use EPROMs you will need to erase them using **an EPROM eraser.**

Some optional bits that might be nice to have

A HARDcase or Litecase computer housing which will allow you to use the Z88 in more extreme environmental conditions. These are available in two forms; a fully waterproof shock resistant version complete with internal long life batteries, or a lightweight version designed for portable use in the less harsh environment preferred by researchers. The latter version comes with shoulder straps and can be operated with both arms free.

Electronic scales will allow you to collect weights automatically. You will also need **an interconnecting cable** for this task. The scales will have to be supported by Infomate. (See the list at the end of the manual).

Electronic Calipers and an interface to measure lengths, **or an analogue interface** to measure temperatures or voltages or ... wow, almost anything!

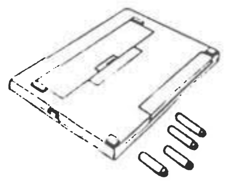
If you link your Z88 to a PC you might want a more powerful data transfer programs such as **RangerLink,** or perhaps the **Ranger Disk** drive unit which will allow you to create PC and Macintosh readable 3.5 inch floppy disks.

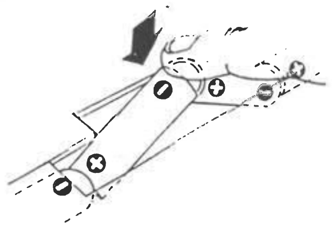
A Printer will allow you to see your data as it is produced by logging data to a printer rather than to an EPROM.

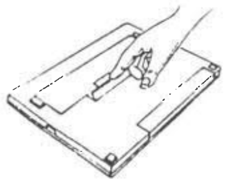
And where to put the bits.

Assuming that you have all the bits listed in the bits-you-need-now list above, how do you fit them all together? The instructions below assume that you have really started at first base and have a new Z88 in its box. If this is not the case, pick up the sequence at the appropriate point.

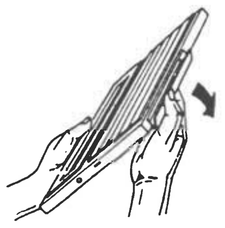
1. Remove the Z88 from its box and assemble around it all of the other bits required.
2. Turn the Z88 on its back and open the battery compartment and if required insert the four alkaline batteries, making sure that they are **all** inserted in the correct direction. The picture below will help with this.

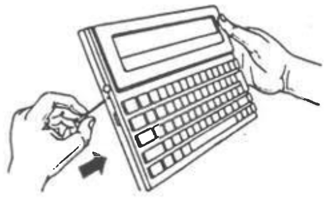

Use 4×MN1500, LR6 or equivalent Alkaline cells

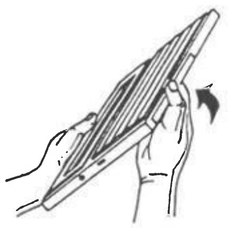

Load batteries


Replace battery cover

3. If the Infomate ROM or the 128k RAM card have not been installed, then with the Z88 switched on, and the display showing the INDEX, tilt the Z88 forward and open the flap at the front edge of the Z88 and insert the RAM into slot 1 and Infomate into slot 2. The picture below shows how to do this.


Open flap


Press reset


Close flap

Notes: If your Z88 already had cards in these slots then you may need to change them. We recommend a RAM card should go into slot 1, if your Z88 differs then you should change it. However if the RAM card installed already contains pending applications or data files then you will need to ◇ KILL the applications, and save the files onto EPROM or to another computer before proceeding. Next, perform a **soft reset** as described in the Z88 manual before restarting these instructions from the beginning.

If slot 2 holds another application ROM you can either remove it and put it in slot 3, or alternatively run Infomate from slot 3. If the application in slot 2 is PCLink or MacLink then it may be removed and stored because this is already installed in Infomate.

4. Reverse the Z88 and turn it on by pressing the two SHIFT keys. The computer should come up showing the Index screen on the display. Note Index at the top left. If it does not you may need to press the INDEX key.

5. Check the time and date by entering the Z88 clock using the command □T . If either needs changing, move to the SET command using ⇨ and press ENTER . You may then type in the correct time and date using the numeric keys and the arrows. An ENTER at the end will set these values into the Z88 clock. If you have difficulty setting the clock refer to the Z88 manual.

6. Move into the control panel using the command □S and make sure that the following are correctly set. Insert/Overtype is probably best set to Insert (move the cursor to this item and press **I** . Default device should be set to **:RAM.1** which can be achieved using the arrows and DEL to delete the 0 and then press **1** . You may also wish to set keyclick to **Yes**. When all options have been selected press ESC to get back to the main INDEX. Note that it is particularly important to have the default device set to :RAM.1 so that all of the Z88 applications that you use will store your files there.

The Z88 and Infomate are now ready for operation.

An overview of Infomate

Infomate consists of a number of specially written data capture modules which are collected together on the Z88. When you add to these the word-processor program Pipedream and the Clock that comes with the Z88, and a short list of commands that are provided by you the user you have a working Infomate system. It is this mix of parts that gives Infomate its strength. All are essential to the operation.

At first the need for you to write a set of instructions seems a bit like hard work, but that is probably Infomate's strongest feature. This set of instructions from you describes your requirements now. If in a weeks time your requirements change, then by just changing your set of instructions you can update Infomate to your **current** requirements.

So, to run Infomate you must set up this set of instructions. We call this a schema file and it is created using Pipedream. When the schema file is complete it is saved in RAMdisk and then Infomate is called up to compile it. This compiling step performs two tasks. First it checks the instructions you have provided for errors, and then it makes an abbreviated copy of the instructions for later use by Infomate.

When your schema file has been compiled you are nearly ready to roll. First, you will need to either open a data file if your data is to be placed in new file, or if you are updating an existing file, perhaps from a different computer you will need to make sure that the data file is in :RAM.1. You are now ready to run your Infomate schema file. This is the process that actually collects and checks your data and places it into the data file.

For extra security of that valuable data it is possible to automatically log the data to a printer as each record is captured, just in case something nasty happens to your data either by your hand or by accident to the computer itself. As an alternative to this you can log the data to a plug-in EPROM memory device which will "remember" your data even if the power is lost and the Z88 destroyed. It is **strongly recommended** that **you use** one of the two **log** possibilities when capturing real data, particularly if it is the sort of data that cannot be recaptured, or is difficult to recapture.

During your data collection you have immediate access to a number of other modules which allow you to change the way that the Z88 appears. For example, they allow you to display all messages in large characters, or set up a numeric keypad. These will be discussed in more detail later.

A quick example for those who can't wait

In this section a quick example of the use of Infomate is provided for those with a burning desire to get something, perhaps anything, going as quickly as possible. If you are game, follow the instructions below very carefully and you should succeed. Make a mistake and disaster (well it might appear that way) could strike. You may then have to slow down and start your journey into Infomate in a more leisurely way. Good luck!

For those who are not in such a hurry perhaps you could sensibly skip this section, and move on to the tutorials below, which will quite quickly get you operational, but by taking that path you will probably understand what is happening rather better. Right here goes for the game ones.

1. Go into the Index by pressing the INDEX key at the bottom left of the keyboard. Using the arrows, position the cursor over the Pipedream command in the list on the left and then press ENTER . This will start up a fresh copy of Pipedream for you to use. Type in the following text taking care **not to use** the TAB key and **not to create any blank lines.** Each line should be terminated by pressing the ENTER key, and you can type in as many spaces as you like between words to separate them and make the text easier to read. Use capital letters throughout by making sure the message **caps** is shown at the bottom right of the screen. If it is not, press CAPSLOCK . Right here goes:

```
TITLE  DEMO
FILE   WEIGHTS.DAT
ID
       FRUITNO N3
DATA
       WEIGHT  N4.1
SEQUENCE
       ENTER   FRUITNO  TYPE  IN  FRUIT  NUMBER
       MAKERECORD
       ENTER   WEIGHT   TYPE  IN  FRUIT  WEIGHT
END
```

2. When you have finished typing, check each line carefully and correct any mistakes. To delete characters move the cursor just past the character to be deleted and press DEL . To insert characters move to the insertion point and type the new text. Lines may be deleted by putting the cursor in the offending line and pressing ◇ DEL , and new lines may be created using ◇N . When all looks OK, save the file by pressing ◇FS (File Save) and then typing the name WEIGHTS.SCH followed by an ENTER . All of the other questions in the ◇FS command don't need to changed.

3. Now start up Infomate by pressing □ ZI . After a brief graphics display you will be presented with a menu of commands.

4. Select Compile Schema by pressing C and type in the name of your schema file WEIGHTS.SCH . If the compiler detects an error you might have to return to Pipedream using the command □P to correct the error line, resave the file, return to Infomate and recompile. When compilation is finished you will be returned to the Infomate menu.

5. Create a new data file by selecting the command New data file create with the N key.

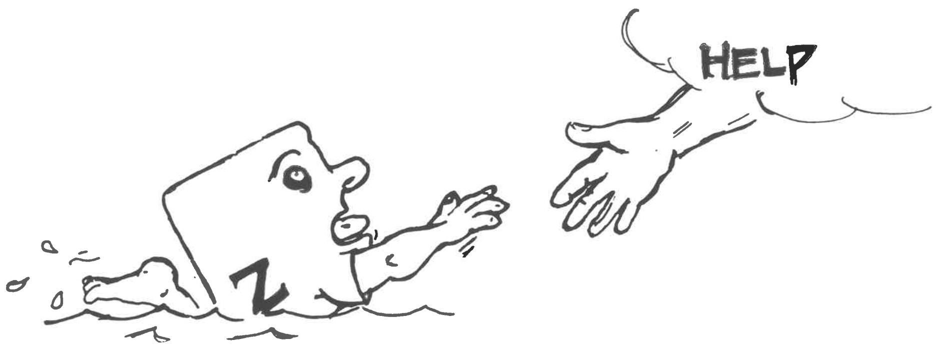
6. Run the schema by selecting Run schema with the R key. If all goes well you will be asked to TYPE IN FRUIT NUMBER as specified in the schema file above. Enter a number of up to three digits and press ENTER . You will next be asked to TYPE IN A WEIGHT which should be in the form 27.4 ENTER i.e. with one decimal place. When completed you will be returned to the question TYPE IN FRUIT NUMBER. This cycle may be repeated as often as desired.

7. To try some different things try entering the same fruit number twice. Press TAB to see earlier entries on the screen. Press ESC and select Mode control with the M key. Select Big characters by pressing B and then press I to install these changes. Running the schema now using the R command will show all of the messages in enlarged characters.

Well, that should be enough to wet your appetite to try Infomate further so proceed through the tutorial chapters now and learn how to create your own schema files, and how they work.

Help HELP

At all times while you are using Infomate there is a HELP facility sitting waiting to remind you of the details of Infomate. This does not provide a full manual for the system, but rather is an abbreviated set of reminders for you to help you remember what you read in the manuals.



To access help, press the HELP key while you are in Infomate and you will be presented with a menu of help topics to choose from. Move the cursor to the one that seems most appropriate to your current needs and then press ENTER . A page of help text will appear on the screen. To move around the help screens you can either use the ⇧ and ⇩ arrow keys on the keyboard, or press MENU to return to the help menu.

You can also move to Infomate help from the help for other applications by moving to the left using the ⇦ key and then moving up and down through the application help list. However, it is probably quicker to move to the application that you require help in and press HELP . For example if you were in Pipedream and wanted to find out the format of one of the Infomate schema lines press □ZI then HELP , move around in the Infomate help to locate the information you require then when you are ready press □P to return to Pipedream.

The Help structure and the ease with which you can move around it, is one of the strong features of the Z88 and it is unfortunate that more help is not provided by the people who developed the main Z88 applications.

Infomate provides some 25 screen pages of help so learn to use it. In fact it is recommended that you try the help out now by starting up Infomate using the □ZI command and then pressing the HELP key. Explore the help pages to gain a bit of a feel for the information that they contain.

If you ever need to know which version of Infomate you are currently using just get into Infomate and then press the HELP key twice. The screen will show the version number, serial number and creation date. This information should be recorded if you ever need to request information on problems from the Infomate authors.

THE SCHEMA FILE

Why have a Schema File?

At first it will appear to you that the need to have a schema file is rather an imposition on the potential Infomate user. However you will learn that it is this single aspect of Infomate that provides you with the flexibility that you really need. It contains all of the information about your data files and the method that **you** plan to use when collecting your data. Rather than arranging your data and its collection in a form that somebody else thinks is right for you, you can collect it and arrange it in the form that you want it. However, to gain this flexibility you will have to invest a little time learning how to use this powerful approach. Don't be daunted by this, it is possible to get away with knowledge of very simple schema files, or to just modify slightly some of the example schemas provided in this manual. For those who do feel daunted, please note that already many people who would never have written a computer program in their lives are becoming proficient in Infomate.

It is important to remember the difference between a schema file and a data file. The data file contains your data, while the schema file describes the format of your data and the way to collect it. In short the function of a schema file is to allow you to provide to Infomate an exact description of your requirements; firstly it defines the way that your data is to be stored in data files, and secondly the way that the data is to be collected. This latter part also allows you to define the checks that you want to make on your data and the shortcuts you wish to take when collecting it. A typical schema file appears in the Quick example section above. While far more elaborate schemas exist, this one holds all of the basic elements and can be expanded to handle more complex tasks. Without knowing anything about schemas you can probably already look at that schema and understand quite a lot of it. Right, lets learn about the bits that we don't know.

What is a Record.

While the word record can mean a lot of things, we do not mean here the type that are sometimes broken at the Olympic games, nor the sort that used to get played at AM radio stations before CD's were invented. For Infomate, a record is a single line of a computer file that holds all of the data related to a single observation-set or measurement. It is complete in that it contains not only the data itself, but sufficient information to **uniquely identify** that data. Examples of records are the tag number, tag colour, previous weight and current state of an animal, or the row number, vine number fruit count and incidence of scale insects on a kiwifruit vine. Each record contains a number of fields some of which identify the record and some which contain the data itself. In some cases fields in a record will be empty either waiting for data, or signalling that data is not available for that observation.

Here is an example of a single record of data with some descriptive detail around it:

A single record

524	YL	90	1699	1425	JER	37.4	40.3	c/r
ID	Tag	Year born	Dam ID	Sire ID	Breed	Mar. Wt.	Apr. Wt.	May Wt.
ID fields			Data fields					

At times it is difficult to define a record. Should it be all observations on a single fruit, or on a tray of fruit. This question deserves some thought because it affects how the schema file will be written. Generally if there is enough space, a record should contain all data relating to a single unit of observation. A tray of fruit being inspected for disease is probably a unit. So might be a cow and two calves at calving time.

Two further considerations for the record designer are firstly, the way that the data is to be analysed when it is eventually transferred to a larger computer. At that point you may require every fruit to be in a different record. At the other extreme you should consider the extent that it is necessary to identify each unit which may allow lots of similar measurements (e.g. the weights of 27 kiwifruit in a tray) to be placed in a single record.

When the contents of a record have been defined you will need to define which of the fields will be used to identify the data, and which are the data themselves. Often the demarcation between these will at first be obscure, but there should be a **minimal** set of fields that will always **uniquely** identify the data record being created. By uniquely we mean that no two records can have the same identification or ID. Examples of unique ID's might be for an animal weighing trial, the animal tag number plus the tag colour which is related to the animals year of birth. For a produce collection trial it may be a combination of row, treatment, and plant number. For an animal behaviour study it could be the date and the time that an observation is made. In some cases it may be just simply a record number. However commonly an ID will be a combination of more than one field.

Because Infomate requires every record to have a unique ID you should take care in defining these fields. There is a strong temptation to use too few ID fields and end up with records that try to have the same ID, or to have too many and end up with data in the identification area. There is a strong differentiation between DATA and ID in Infomate.

Infomate now becomes your tool to create and fill the data records in a file in the way that you want to fill them. It will allow you to add records to an existing file or create a new file. You can also modify just a few of the fields in each record of a file, and even have a number of schema files set up to modify or add data to different parts of a single file on different occasions. All this is possible without the need for computer programmers; you take over that role. You should note that Infomate is not a data base package and does not allow you to move data from record to record, nor to collect data in more than one record at a time. In practice this is not a limitation for most data capture tasks.

Giving the fields names

It makes sense to give fields a name that describes their function. Thus an animal ID tag number could be called "Tag_Number", an orchard row number could be called "Row_No". In both of these examples the double quote (") characters are not part of the name. Note the convention of using capital letters to start a word in a name, and how the two words are separated by an underscore "_" character. From this you may infer that a field name like "Row Number" with a space in the middle is not valid. However the use of upper and lower case is completely up to you. ROWNO, rownumber, Row_no, and ROW_no are all perfectly valid names. But be careful, because Row_No is a different name to Row_NO. While it is not essential to follow the above convention, it is probably one of the tidiest.

You should also note that field names must be different from each other. If you have two fields called weight only the first one will ever receive any data.

What type of field is that?

Fields may come in two types. Number fields and letter fields. Examples of number field values might be "12.7" for a weight number field, or "27" for a fruit count. Note again that the quotes (") are not part of the field value. Letter fields examples are "RED" for a fruit colour, or "This is comment number 7" for a letter field containing a comment. The way that we inform Infomate of the type of field we are using is to use one of the letters N or L which is short for Number or Letter respectively.

Further information is needed next to describe the size of the field. Thus, N3 is a number field that contains 3 characters, and L12 is a 12 character letter field.

Number fields may be even more complicated. They may contain a decimal point in some circumstances. We tell Infomate this by putting a decimal point after the field size character and then add the number of decimal places we wish Infomate to record after the point. Thus numbers like 12.7 or 99.1 are described as N4.1 fields. This means that the **Number** field has **4 characters**, there **is a decimal point** , and there is **1 digit after the decimal point**. Whew! If a decimal point is not required and the numbers are to have 4 digits then just use a field description like N4 .

Note: for obvious reasons number fields as specified above cannot contain more than 9 characters. This is not unreasonable as the accuracy of the arithmetic on the Z88 is only about 9 digits anyway. Letter fields must not be greater than 24 characters in length for reasons that made the programming of Infomate a little easier.

Some further limitations on ID fields

There are a few limitations on ID fields that do not apply to data fields. ID Number fields must contain only the digits 0 to 9 and the space character, and cannot contain decimals. ID Letter fields must contain only the letters of the alphabet plus the slash (/) and blank character, and no distinction is made between upper and lower case letters; ab123 and AB123 are considered to be the same.

In addition, the total number of characters in all ID fields is limited to 6 letters or 9 digits. Where there is a mixture of letters and numbers, the following limits apply.

Numbers	0	1	2	3	4	5	6	7	8	9
Letters	6	5	5	4	3	2	2	1	0	0

Thus, if there were 3 letters in an ID then the maximum number-field-digits allowed would be 4, no matter what the organisation of the fields was. The reason for this limitation is that Infomate maintains an abbreviated copy in memory of all ID's used so that it can quickly find a particular ID. If you have designed your data and ID fields correctly this should not be a limitation, as it allows you to uniquely identify one billion records, and that is a lot of asparagus.

Putting the file definition together.

In the last few sections we have described what fields are and how they are described. Lets see how they are put together in a way that Infomate can understand. This is most conveniently done using the simplest possible example. Consider a data capture requirement that involves recording fruit weights of a few hundred individual fruit which are typically 60 grams each. The ID and DATA regions could look like this:

```
ID
     Fruit_No      N3
DATA
     Weight        N4.1
```

Which in English can be read as a single ID field called Fruit_No which is a 3 character number and a single data field called Weight which is a 4 character number with 1 decimal place recorded. Now, we must add to that two further lines to describe the purpose of the schema file (i.e. a TITLE line), and the name of the data file into which we intend to place the data that we collect (a FILE Line). The TITLE line should contain enough information to inform you what this schema file does, and in particular how it differs from all other schemas like it. It should also contain author, date, and details of any modifications.

Put this together and the schema file will look like this:

```
TITLE       Fruit Wt Collector 30/7 RBJ
FILE        WEIGHTS.DAT
ID
     Fruit_No      N3
DATA
     Weight        N4.1
```

If there were to be more ID or DATA fields then these would be added into the position required. Now all of that above just says what the file is to look like and does not say how to collect the data. That comes next!

Defining the SEQUENCE

While there is a lot of flexibility needed to allow **you** to create a file specification to suit your current needs, even greater flexibility is needed to allow you to define how the data is to be collected. You need to be able to control the order that data is collected, and the messages and checks that are to be placed in the collection sequence. This is done using a SEQUENCE which in reality is a computer program that tells Infomate what **you** want to do.

The sequence follows immediately after the DATA definitions and is started with the line

```
SEQUENCE
```

and strange as it may seem, ends with the line

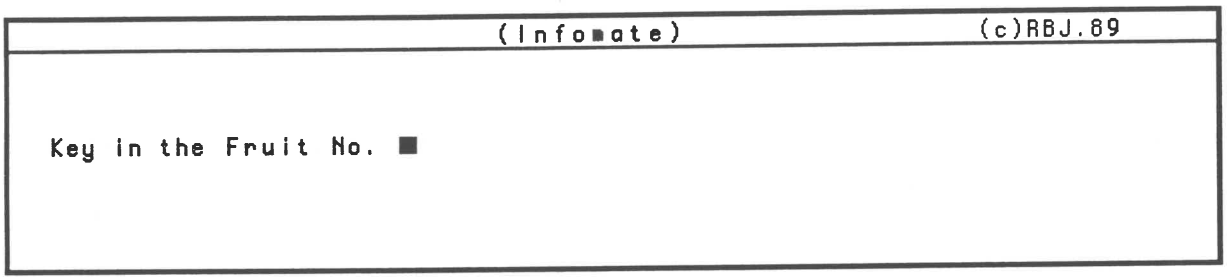
```
END
```

Within these lines you place your sequence or program. Lets look first at a short sequence to go with the ID and DATA definitions above. It has three tasks. First it must fetch from the keyboard the number of the fruit that is to be weighed i.e. its ID. Secondly it must check the ID for validity and make space for the record on the file using a MAKERECORD, and finally it must fetch the fruit weight from the keyboard.

The sequence to perform that might look like this.

```
SEQUENCE
     ENTER     Fruit_No       Key in the Fruit No.
     MAKERECORD
     ENTER     Weight         Key in Fruit weight
END
```

The only part of the above that may not be completely obvious to you are the messages at the right of the two lines starting with ENTER. These are the messages that will be displayed on the screen to inform the person weighing the fruit what they are required to do. They would see on the screen something like this:



```
                         (Infomate)                        (c)RBJ.89

 Key in the Fruit No. ■
```

So let's look now at the completed Schema file as we have created it so far. It contains some general information about the file followed by a full definition of the file content format and winds up with a description of the way the data is to be collected. So here it is:

```
TITLE   Fruit Wt Collector 30/7 RBJ
FILE    WEIGHTS.DAT
ID
        Fruit_No  N3
DATA
        Weight    N4.1
SEQUENCE
        ENTER     Fruit_No  Key in the Fruit No.
        MAKERECORD
        ENTER     Weight    Key in Fruit weight
END
```

Note how space characters have been inserted to make the file more readable. Infomate in fact only needs one space between each part of the line but is quite happy to accept as many as you like. Also note that all of the special Infomate words like FILE, DATA and ENTER are typed using capital letters only.

Putting Comments into a Schema file.

When you create a schema file it is often nice to be able to put some marginal comments into the file to remind you about what you are doing or what might need checking in later versions. Infomate allows you to do this by placing the text of the comment at the end of any schema line and preceding it with a backslash (\) character. Thus you might like to comment the above file like this

```
TITLE   Fruit Wt Collector 30/7 RBJ
FILE    WEIGHTS.DAT             \Change later to rowno.DAT
ID
        Fruit_No N3             \ Only 500 fruit per file
DATA
        Weight   N4.1           \That allows wts to 99.9k
SEQUENCE
        ENTER    Fruit_No   Key in the Fruit No.
        MAKERECORD
        ENTER    Weight     Key in Fruit weight
END
```

Now when that is fed into the compiler later all those comments will be ignored, but they remain there for your purposes at all times when you look at them with Pipedream. Note that although it is possible to start a comment at the left hand margin of a line, and thus have no Infomate statement in that line, this is not recommended as it will confuse Infomate whenever it needs to tell you the number of a line that contains an error. If you must put in a block of comments, use dummy LABEL statements at the left so that Infomate can keep track of your line numbers like this:

```
LABEL   \a block of comments
LABEL   \can go like this
LABEL   \and Infomate won't mind.
```

Throughout this manual we have not used comments extensively because most of the schema files examples are described in some detail in the text. We recommend that you make a habit of putting comments in your schemas; it will help you understand them later when you modify them.

Well, the above is really only a paper schema; it doesn't yet exist in a computer - that transition is very close now as you are now ready to enter it into the computer.

Using Pipedream to Create a Schema file.

Pipedream is a simple word processor and is ideal for handling schema files. The techniques you learn in this section will be sufficient to allow you to create simple text documents such as letters, and of course schema files.

One at first confusing aspect of the Z88 is that you can have many documents or applications "open" or pending at one time. As you get used to this you will learn to use the power that the concept provides. For example you can be typing a letter to Mum and by pressing two keys start typing a schema file. At any stage you can move back to the letter by pressing the same two keys. Nothing of the earlier work is lost, not even if you stop typing in the middle of a word. This could be useful to let you tell your Mum about your cunning schema file.

If you want a new version of Pipedream to be set up for you, then select the Index by pressing the $\boxed{\text{INDEX}}$ key. Now move the cursor to the Pipedream line on the left and press $\boxed{\text{ENTER}}$. A new version of Pipedream will be created for you. If on the other hand you wish to move to an existing Pipedream module or application as it is called, just press □P . If more than one Pipedream application is active it may be necessary to press □P more than once to get to the one that you want. This technique for moving from one Z88 application to another can be used at any time. Immediately after loading Pipedream press □F to get into the Filer and check that it is displaying the device for the RAMdisk :RAM.1 . If it is not then you have not set the control panel correctly so first change the device to :RAM.1 using a ◇SV command and finally change the panel's default device to **:RAM.1** using □S .

So, you are in Pipedream and are ready to type. Type in the schema that is shown above. Use the space key freely to layout your schema and make it easy to read, but **do not** create blank lines or use the $\boxed{\text{TAB}}$ key to space your text. In Pipedream the $\boxed{\text{TAB}}$ key means something entirely different and should be avoided like the plague. If you do accidentally hit $\boxed{\text{TAB}}$ then clear all of the text you have typed into the column that you have tabbed to and press ◇$\boxed{\text{TAB}}$ to move back to the start of the column A. Knowledge of a few more keys to control Pipedream will be handy here. Some keys, or key combinations are shown in the table on the next page. If you want to know more, read the sections on Pipedream in one of the books recommended at the start of this manual.

Keys	Action performed.
DEL	Deletes the character before the cursor.
SHIFT DEL	Deletes the character under the cursor.
◇D	Deletes from the cursor to the end of the line.
◇ DEL	Deletes entire line and moves following lines up.
◇N	Creates a new blank line at the cursor and moves the current line down.
◇S	Changes the case of the character under the cursor. i.e. capitals become small letters and vice versa.
◇ESL	Moves all of the text from the cursor onwards into a new line below the current line. i.e. edit split line.
◇EJL	Merges the current line and the line below into a single line. i.e. edit join lines.
◇⇦	Moves to the beginning of the line.
◇⇨	Moves to the end of the line.
◇⇧	Moves to the first line.
◇⇩	Moves to the last line.
SHIFT ⇨	Moves one word to the right.
SHIFT ⇦	Moves one word to the left.
◇T	Deletes one word.

With the cursor arrow keys, and the set of commands listed above you can perform any editing required in Pipedream. There are other commands to allow you to move blocks of text around and these are described in the Z88 manuals.

When you have typed in the schema you will need to save it in RAMdisk. This can be done by typing ◇FS (i.e. File Save), and then typing the name you wish to give the schema. It is suggested that all schema files end with the file extension letters .SCH . For example the above schema might be called WEIGHTS.SCH . Whether you use upper or lower case for your file names is entirely up to you as the Z88 doesn't really know the difference in file names. However you should probably be consistent so make a decision early.

Take care with this schema file name as it is very tempting to give it the same name as the file you have named in the FILE line of the schema. This is wrong as that name refers to the **DATA** file and not the **SCHEMA** file. The difference is significant! One holds your data (the .DAT file) and the other your instructions (the .SCH file).

During the development and testing of your schema file, it may be necessary to make small changes and resave your file a number of times. To make this process easier, it is suggested that you perform a ◇FC command to specify the Current File name to Pipedream. By typing your schema file name into this command, subsequent operations to save the file using ◇FS will have the current name displayed and you can "accept" that name by pressing the ENTER key.

If you want to send a schema file to another computer for printing etc. save it from Pipedream using the 'plain text ' option. This will eliminate all of the Pipedream formatting characters from the start and end. It is not necessary to save the file for Infomate in this way as Infomate has been 'trained' to ignore these characters.

And so ends a short lesson in Pipedream. Not a marvellous word processor or editor by any means, but I think you will find it to be entirely adequate as your proficiency improves.

USING INFOMATE

Moving round the Z88 environment

Having just created a schema file using Pipedream you now need to pass that schema to Infomate itself and use it to collect data. But first lets recap a little on the way of moving around the Z88 environment. You learnt in the last section to move to Pipedream using the □P command. Well there are abbreviated commands to move to any of the Z88 applications or popdowns. One of the popdowns that you will need quite often to check if is time for coffee is the Clock. This can be accessed by pressing □T (for Time). When you have finished with the clock, just press $\boxed{\text{ESC}}$ and you will be returned to whatever you were doing. This clock function is called a popdown which can be overlaid on top of your current task whenever you need it. It will disappear when you press $\boxed{\text{ESC}}$, and return you to your original state. An application like Pipedream will not disappear when you press $\boxed{\text{ESC}}$, it is a little more permanent. Infomate is an application too, but unlike Pipedream you can only have one copy of Infomate in memory at a time. In later versions of Infomate it may be possible to have more than one but that is another story. To start up Infomate, or to return to it later you need only press the key sequence □ZI which is an abbreviation for Z88 Infomate. If you forget this you could also do the same thing by pressing $\boxed{\text{INDEX}}$ and then selecting Infomate from the right hand column if it has been started, or the left hand column if this is the first time.

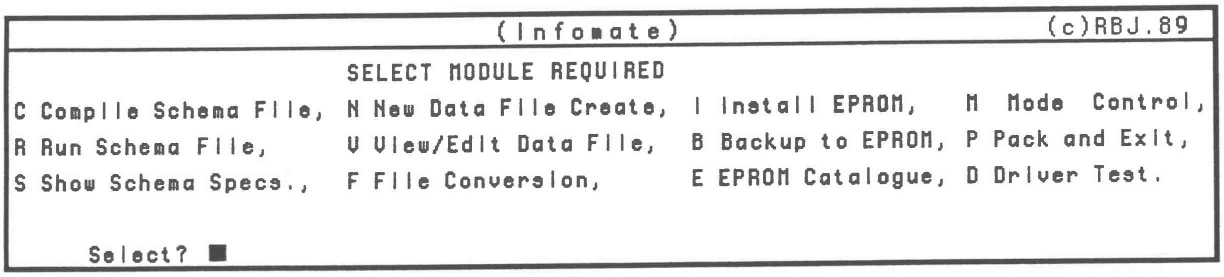
The most common application switches that you will make when using Infomate to create and test a schema file are those between Pipedream and Infomate. These are performed using □P and □ZI respectively. While you could go to the Index each time, try using the abbreviated commands. Try now - start up Infomate using □ZI and swap to and from Pipedream with □P and □ZI a few times. If in fact you have more than one Pipedream application pending at the time you press the □P command you may end up in the wrong one. To fix this just press □P again. The cycle will then become □ZI □P □P etc.

Compiling a schema

So you have a schema file, you can use Pipedream, you can get into Infomate -what next.

Your schema file is an "almost-English-language" statement of your requirements, and it must be compiled by Infomate before it can be used. Compilation can be thought of as a translation process that is carried out by a computer to turn text that you can understand into numbers that the computer can understand. For example Infomate does not really care whether you call your weight Fruit_weight , Weight , or in fact just Wt . It just sees it as the first data field. However if you called it Weight in the DATA definition region and WT in the SEQUENCE region then it would become understandably confused. From this we can infer that the compiling step also checks for some logical errors.

When you go into Infomate you are presented with a menu of commands. **This is the Infomate Menu** and will be referred to often from now on.



```
                              (Infomate)                              (c)RBJ.89
                        SELECT MODULE REQUIRED
C Compile Schema File,  N New Data File Create,  I Install EPROM,     M Mode Control,
R Run Schema File,      V View/Edit Data File,   B Backup to EPROM,   P Pack and Exit,
S Show Schema Specs.,   F File Conversion,       E EPROM Catalogue,   D Driver Test.

    Select? ■
```

All of these are activated by pressing the first letter. Thus to View/Edit your data file press V . If you have not compiled your schema you will probably get a message telling you that there is "No Data File Specified".

Whenever you need to return to the menu above you can do so by just pressing the $\boxed{\text{ESC}}$ key, but be sure that Infomate has completed its current task. If for example you were capturing data and the last value you typed in was still shown on the screen, then if you pressed the $\boxed{\text{ESC}}$ key at this point you would lose the data before it was saved in the internal memory of the Z88 or was logged on the printer or EPROM.

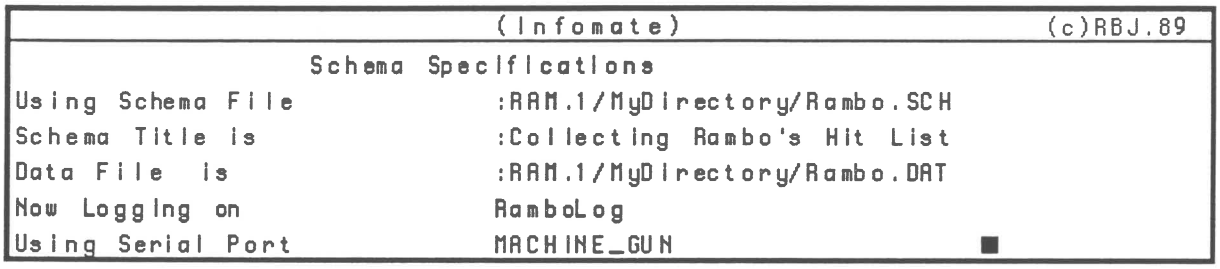
Right, select "Compile schema file" by pressing C and if all goes well you will see your file whiz past and you will be returned to the Infomate menu again. If you want to stop it going so fast just hold down both the $\boxed{\text{SHIFT}}$ and ◇ keys simultaneously after you press the C . You could try that if it compiled OK on the first pass by repeating the Compile step. You will note when you inspect the compiling file in this way that all of the extra spaces and comments that you typed have been eliminated by Infomate. Frankly My Dear, Infomate doesn't give a (hoot) about these.

If you were not so lucky to compile your schema first time (and most normal people aren't so lucky), then note down the line number of the fault from the Infomate error message. Next look at the error message and try to determine what you have done wrong. When the problem has been located, switch back to Pipedream using □P and make any corrections required. Save the new file from Pipedream using ◇FS (which will overwrite the original file with the corrected version), and then switch back to Infomate using □ZI. You may now need to press a key to terminate the error message and you can try compiling again. It is not uncommon when compiling a new schema to have to swap back and forth between Infomate and Pipedream a number of times. Eventually you should get a compiled version of your schema and can proceed.

For those who did manage to compile first time, it is probably worth going back into Pipedream and introducing a deliberate error to learn the method of swapping back and forth and using the applications in parallel. So change the spelling of one of the words in the schema file, save and recompile. Note that Infomate does not check the spelling of the message part of the line!

Which Schema file is this anyway?

If you use the S command **"Show Schema Specs"** from the main menu you will be presented with a brief summary of your schema file. This includes the name of the schema file and it's associated data file, and details of the TITLE, and any PORTIS and LOG statements. This is a useful way of checking that the current schema is the one you really want, and shows why you should keep your TITLE line as meaningful and as up to date as possible. An example of the Show Schema Specs. screen is shown below.

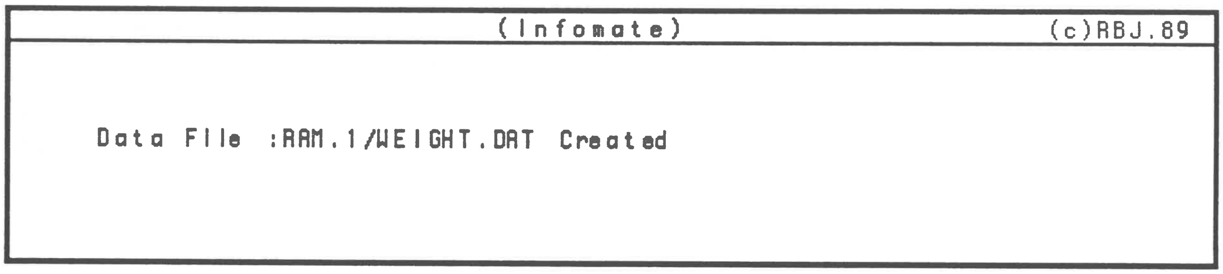


```
                          (Infomate)                     (c)RBJ.89
                 Schema Specifications
Using Schema File          :RAM.1/MyDirectory/Rambo.SCH
Schema Title is            :Collecting Rambo's Hit List
Data File  is              :RAM.1/MyDirectory/Rambo.DAT
Now Logging on             RamboLog
Using Serial Port          MACHINE_GUN
```

Well you are getting very close to running the schema file now, but first one important step.

Creating a data file

Even though your compiled schema has all of the information required to create the data file in which your data will appear, it doesn't know when to create the file. You can create a file at any stage using the **N** command **"New Data File Create"** from the Infomate menu. If there is not already a file in :RAM.1 with the name specified in your schema file, then a file will be created, the message



```
                          (Infomate)                     (c)RBJ.89


   Data File :RAM.1/WEIGHT.DAT Created
```

will be displayed briefly and you will be returned to the menu once more.

If when you select "New Data File Create" there is already a file with that name on :RAM.1 then you will be asked to rename the old file. This is part of **the Infomate philosophy** that **at all times**

Infomate will attempt to protect any data that has been collected. Thus it would be rude of Infomate to just assume that the old file is not wanted and overwrite it.

You can use this to advantage if you want to collect data into a number of different files, one file per row of fruit trees for example. Create a file for the first row, and fill it with data from that row. Next create a new file and rename the first row's file as ROW1.DAT. You can now start filling the row 2 file. This has particular advantages when you are logging data to an EPROM which is described in a later chapter.

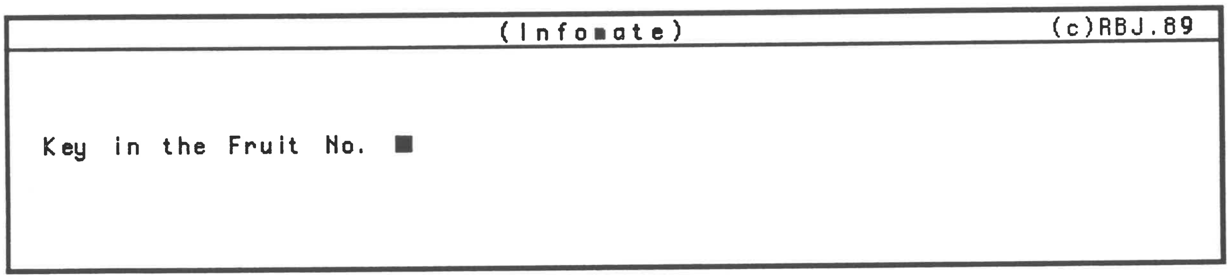
So with a compiled schema and a data file in which to place your data there is nothing to stop us?

Running a schema

The process of stepping through the sequence part of the schema file is described as **running** the schema or running the sequence. It is, in computer parlance, the execution phase of the process. When you select **R** to **"Run Schema File"** from the Infomate menu, Infomate will first of all load in the ID fields of all of the data that has been collected into the data file to date (if any) and then start executing the sequence. So lets look back on the sequence we have been working through above.

```
SEQUENCE
     ENTER     Fruit_No        Key in the Fruit No.
     MAKERECORD
     ENTER     Weight          Key in Fruit weight
END
```

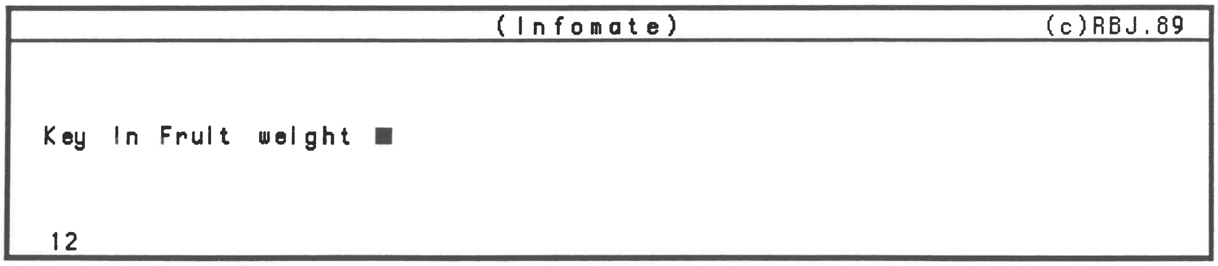
When you press the R to run the schema you will see



```
                          (Infomate)                    (c)RBJ.89

 Key in the Fruit No. ■
```

which is Infomate executing the first line of the sequence.

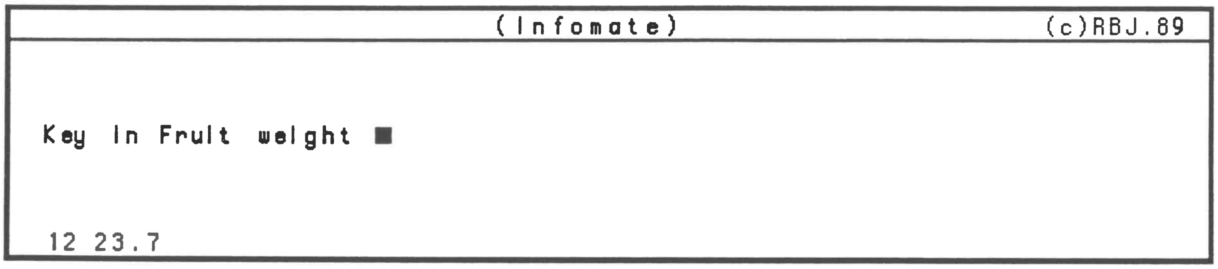
Type in a number for the fruit number, say 12 and you will see



```
(Infomate)                    (c)RBJ.89

Key in Fruit weight ■

12
```

Note that the Fruit number has appeared at the bottom of the screen. As each ID or DATA field is filled you will see their contents appear at the bottom of the screen.

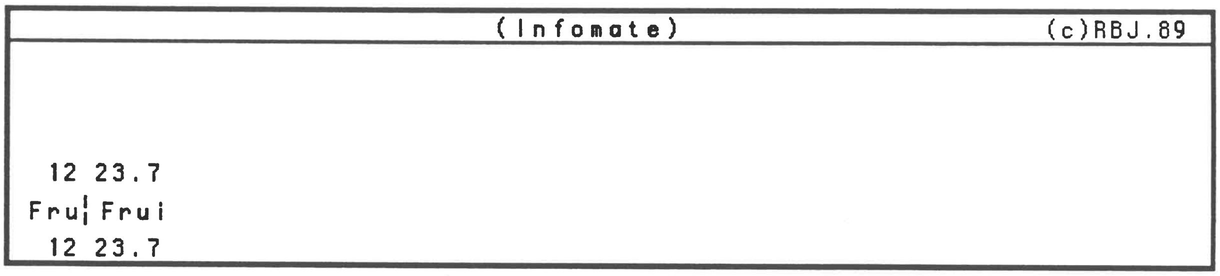
Try typing a weight now such as 23.7 and the screen will briefly appear as



```
(Infomate)                    (c)RBJ.89

Key in Fruit weight ■

12 23.7
```

before it reverts to the first question again. In that short period the data has been stored in the file WEIGHT.DAT in :RAM.1 . The sequence may now be repeated as often as required.

If you want to see previous records entered press the $\boxed{TAB}$ key and the screen will change to



```
(Infomate)                    (c)RBJ.89


 12 23.7
Fru¦Frui
 12 23.7
```

Now that last display is showing at the bottom the current record, which in this case has just been completed, the names of the fields (or at least as much of them as will fit), and the previously saved record above that. If you had saved more than 4 records the screen would show only the most recent four data records. You can revert to the data capture phase at any time by pressing $\boxed{TAB}$ again.

The vertical line between the two field names shows where the ID and DATA regions of the record are separated. At any stage you can return to the main Infomate menu by pressing the ESC key.

So that the field names above are more readable you may wish to change Fruit_No. in the schema to Num_Fruit , and Fruit_Weight to Weight. Try this as an exercise, taking care to change the names both in the field definition areas as well as in the sequence. This will make the display of field names appear as

```
                Num¦ Weig
```

which is far more readable. So the names of the fields take on a new importance here.

Oops I made a mistake

When you are working your way through a sequence you may find that you typed something incorrectly on the previous ENTER statement. To go back to that statement and correct that error all you need to do is press the ⇧ key and you will be 'backstepped' to the previous question. The value you typed will be presented to you as an option and you can edit that data using the ⇦ and ⇨ keys to get you to the right place, and the DEL key to erase your mistakes. When you have edited the line to the state that you want it to be, press the ⇩ key to move back through your sequence to the place that you left. This ⇧ and ⇩ operation is available to you at all times and will step to any ENTER, DISPLAY, REPEAT or GETPORT statement. Remember when you are designing your schema though that it steps only back and forward, and will not follow a complicated chain of GOTO statements.

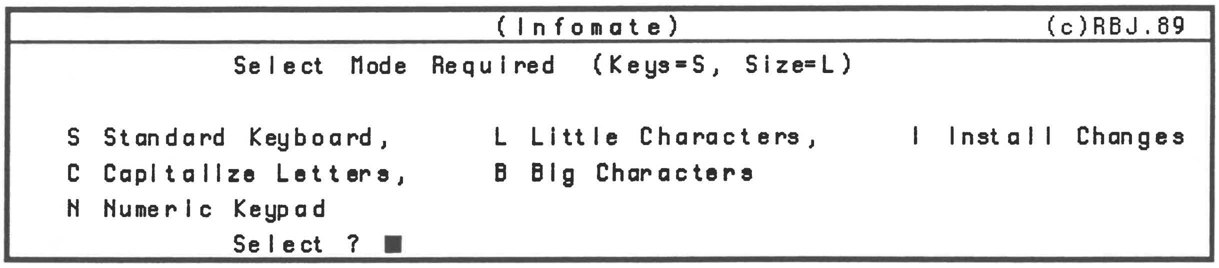
Unfortunately once you have reached the END statement of your schema, it is not possible to step back in the current record because the current record has just become the previous record. Sorry!

Play a little further with the sequence created in the last section now, by using this new backstepping concept. Try to enter fruit IDs that already exist, try creating new files, and then you might like to try changing your *modus operandi*.

Modes of operation

Infomate has two sets of operation modes that the operator can select at any time. To change modes, return to the Infomate menu using the ESC key and press **M** for the **"Mode control"** command.

The screen will appear like this.

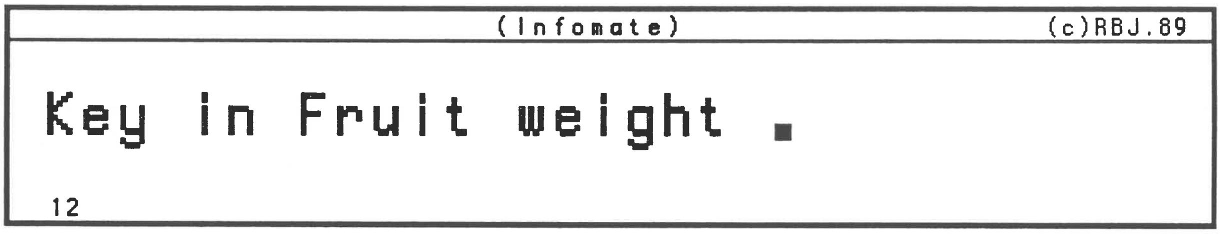


```
                        (Infomate)                              (c)RBJ.89
              Select Mode Required  (Keys=S, Size=L)

  S Standard Keyboard,       L Little Characters,       I Install Changes
  C Capitalize Letters,      B Big Characters
  N Numeric Keypad
              Select ? ■
```

You will note that at the top of the screen the current settings of the two modes are displayed. In the case shown the Keyboard is set to Standard and the character size is set to Little. To explore these try pressing the letters of the different modes and note how they change in the line at the top. When you have set the modes as you want them to be press the I key or the ENTER key to Install them. If you have changed your mind (or chickened out) just press the ESC key and all of the changes you made will be ignored and you will be returned to the main menu. The above menu is a standard Infomate menu style which is also used in the file conversion menu described later. The modes that you are setting here will now be described in more detail.

Big and small letters

The examples given above for running the schema files were shown in Little character mode. If Big characters were selected then the screen would look like this (sort of).



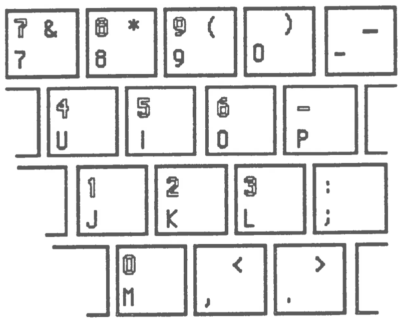
```
                        (Infomate)                              (c)RBJ.89

 Key in Fruit weight ■

 12
```

Obviously in this mode there is a limit to the length of the message, and the display is marginally slower. In some circumstances these two deficiencies are well worth the added advantage for short sighted people, or those who work in situations where light or position are less than optimum.

Capitalize letters

This mode will return all letters as capitals, regardless of the state of the **caps** indicator at the lower right of the screen. This prevents accidental bumping of the $\boxed{\text{CAPSLOCK}}$ key causing changes to the data collected. In this mode the numeric keypad is disabled.

The numeric keypad



One of the major criticisms placed on the Z88 keyboard by experienced typists is the lack of a numeric keypad for data entry. A partial solution to this problem has been provided by Infomate. The HARDcase and Litecase Z88 housings are provided with a keyboard overlay that shows alternative functions for the keys UIOJKLM and P. These are labelled with the numbers 4561230 and - respectively and provide a key layout as shown to the right. This is fine as long as you can let the computer know which of the two possibilities that you intend when you press a particular key. This is achieved as follows, and it is suggested that you try these techniques on the Z88 as you read.

Infomate is powered up in Reverse Caps mode which means that the $\boxed{\text{SHIFT}}$ key always swaps the function of the **caps** indicator on the screen. i.e.. with **caps** displayed pressing c gives a capital "C" while $\boxed{\text{SHIFT}}$ c gives a little "c". When **caps** is off the reverse is true and the $\boxed{\text{SHIFT}}$ key performs a more normal function i.e. the key c gives little "c" and $\boxed{\text{SHIFT}}$ c gives capital "C".

Now if you select Numeric keypad mode, the **caps** indicator becomes a Number/Letter indicator. If it is shown, you get letters (in fact you get capital letters). If it is off or not showing you get numbers, all according to the key layout above. You now have two ways to operate the numeric keypad. Either press $\boxed{\text{CAPSLOCK}}$ to get the **caps** indicator showing for letters or not showing for numbers, or for a quick and temporary change just hold down the shift key while pressing the number on the numeric keypad..

A typical scenario is this: all data fields to be collected are numbers except a single character colour code. Set the Z88 to Reverse Caps off mode by erasing the **caps** indicator using $\boxed{\text{CAPSLOCK}}$. This enables the numeric mode. Select numeric keypad in the mode menu. Now all keys pressed in the MJKLUIOP area will be numbers. If you need a letter just hold down the $\boxed{\text{SHIFT}}$ key and press the number. In fact you only need to hold down SHIFT for MJKLUIOP.

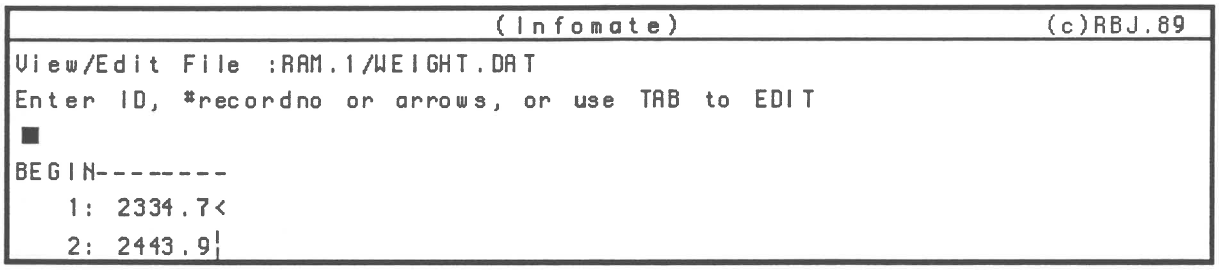
Summary of the Numeric keypad Mode: if the word **caps** (in lower case) is showing then UIO... are letters. If the word **caps** is not showing then UIO etc are numbers. In either mode, the $\boxed{\text{SHIFT}}$ key always alters the function of the UIO keys as indicated by the **caps** display. To get out of ReverseCaps mode press ◇ $\boxed{\text{CAPSLOCK}}$, and to return to it press □ $\boxed{\text{CAPSLOCK}}$.

Viewing a data file

OK, so we collected some data and we saw a pretty display that matched the values we typed, but how do we really know that the numbers have actually gone into the file and are not just being held by a clever display program within Infomate? A valid question!

By returning to the main Infomate menu using the ESC key you will see in the second column a command operated by the **V** key called **"View/Edit File".** This module is for viewing your data file and should allay your fears a little.

Press the **V** key to select View/Edit. The display will turn into something like this.



```
                              (Infomate)                          (c)RBJ.89
View/Edit File :RAM.1/WEIGHT.DAT
Enter ID, *recordno or arrows, or use TAB to EDIT
■
BEGIN--------
   1: 2334.7<
   2: 2443.9¦
```

There is quite a lot to understand there. Lets start at the top line. This shows that you are in the View/Edit module and that your data file is called WEIGHT.DAT . The second and third lines are your instructions, and a place to type your requests; we will return to them shortly. The last three lines are the contents of the front of your file. The word BEGIN is not actually in your file but serves as a marker to show where the start is. The numbers at the left (1 and 2) are record numbers and are also not in your file but serve as markers for your file viewing and editing. The actual data in the file is from the colon (:) character to the < or ¦ characters. This data shows a fruit numbered 23 with a weight of 34.7 and a fruit numbered 24 with a fruit weight of 43.9 . The fact that there is no space between the fields seems a bit strange, but that is what the schema file specifies, so that is what you get. The < character points to the line to be edited and we will return to that also.

Moving through the file

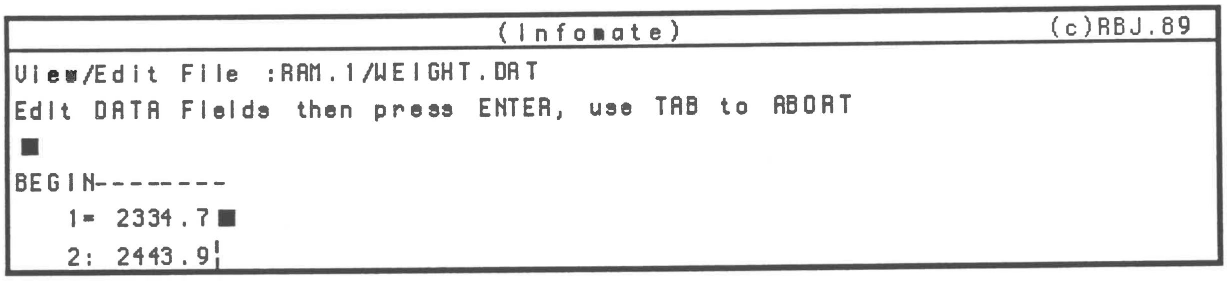
To move around your file use the arrow keys ⇧ and ⇩ . If they are a bit slow to move to the point that you want to go to, then hold down the SHIFT key while using the arrows and you will move through your records 3 at a time. By holding down ◇ or □ the arrow keys will work even faster; in fact at 10 and 50 records for each press of the arrows. You might like to think of the SHIFT , ◇, and □ keys as accelerators or turbo-boosters? These are different usages of these keys from other Z88 applications, so be warned that they behave differently in Pipedream.

There are other ways of moving through your data that are even better if you know a little about your data. For example if you want to find ID number 27 then just type 27 ENTER and you will be there. This works for all ID's if you type them as they appear in the ID area, even if they are composed of a number of fields. For example if you had an ID comprising a single digit year, a three digit tag number and a single letter tag colour then typing 51234G would search through the data for the year 5 green tag number 1234 animal. Note also that provided there is no ambiguity then 62R will find year 6 tag 2 colour Red.

Sometimes you do not know the ID of the record you are looking for but know its position in the file. To access that record precede its record number with the # key. Thus #54 will move to record number 54 in the file and display its contents. #9999 will always find the last record.

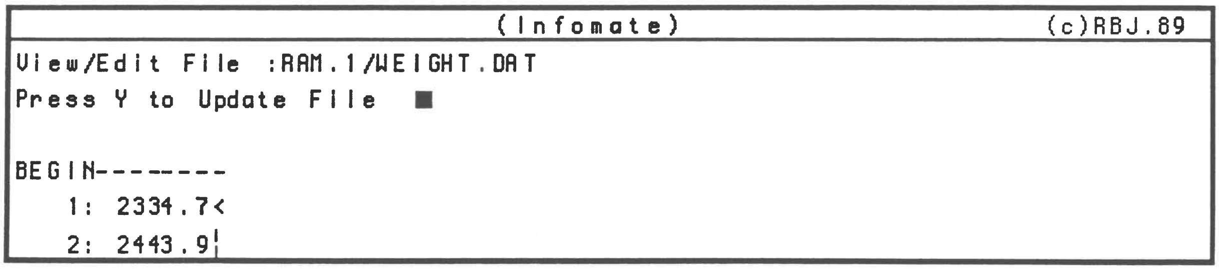
Changing your files contents

What if there is a mistake in my data? Well that is the second function of the View/Edit module. When you have located the record that is to be edited and have it positioned at the < mark on the display, press the TAB key.


```
                         (Infomate)                              (c)RBJ.89
View/Edit File :RAM.1/WEIGHT.DAT
Edit DATA Fields then press ENTER, use TAB to ABORT
■
BEGIN--------
   1= 2334.7■
   2: 2443.9¦
```

You are now in the edit mode and can modify the data in the record. Note that I said **data** here. At this stage it is not possible to modify the ID part of the file for a number of reasons which include partly laziness on the part of the author. So move the cursor to the part of the record that you wish to modify and you can then overtype the new information into the record. At this stage you are not modifying the data file itself, but a copy of that particular line. **Take care when editing as you have the potential at your finger tips to stuff up real data.** In fact if the data is that useful you should only modify a copy of the data rather than the data itself. You can make a copy using the **Filer** if you read the Z88 manuals.

When the record looks like you want it to look press the ENTER key and you will be asked to "Press Y to Update file" like this:



```
                          (Infomate)                          (c)RBJ.89
View/Edit File :RAM.1/WEIGHT.DAT
Press Y to Update File  ■

BEGIN--------
   1: 2334.7<
   2: 2443.9¦
```

If for any reason you are not happy with your editing do not press Y and the data in the file will remain intact. Conversely pressing **Y, will update** and permanently modify the data file. Two further keys will help you to move through your data fields when in the editing mode. Pressing ◇⇦ will move to the start of the first data field of the record, and ◇⇨ to the end of the last data field.

Pack and Exit

While using Infomate you have seen the ease with which you can swap back and forth between Infomate and Pipedream. If your Z88 is full you may get a 'No Room' error when you try this. If this happens then each time you move from Infomate to another application you will have to use the **P** command **"Pack and Exit"**. This command will squeeze Infomate down as small as possible before it transfers to the Z88 Index. Even though Infomate is squeezed, it will remember all of the important things that you have told it, including the values of all of the fields in the last record, the SCRATCHPAD field values, the details of your schema file and lots more

You should also use the Pack and Exit command if you are ever in the state of not needing Infomate for a while but do not want to kill it just yet.

EPROM AND PRINTER USAGE

Let us deviate a little from all of this schema file stuff and discuss storage of data files and schema files on EPROM, and using printers to print data as it is collected. EPROM's are a feature of the Z88 that are found on few computers of its type. They are a robust, low cost and convenient unit in which to store Z88 files in a form that is safe from battery failure and reasonably safe from serious hardware failure as well. In computer jargon EPROM's are described as being **non-volatile** which means their data does not evaporate when the power goes off as it does in RAM memory. All important data should be backed up into EPROM.

There are a number of ways of using EPROM, each with their particular merit. These are described in the following sections.

EPROM Erasure and erasing policy

So if EPROM's are so darn tough do you just throw them away when you have finished? In fact that is not the case. EPROM's may be erased by exposing them to ultraviolet light for about thirty minutes using special EPROM erasing units. These units erase everything so you cannot just delete a single file. This calls for a little organisation in your data handling methods. One technique is to have sets of three EPROM's which you cycle through in a systematic way. This calls for good bookkeeping. At any point in time you will have a current EPROM holding the latest copy of your files. A second EPROM will hold the previous version in case anything goes wrong with the current one. A third one which held an even older copy is erased ready for use next time. At each stage of the process write on the EPROM label, the status of the data it contains and the date and perhaps the time.

Now this technique really only works with schema files and other files that do not change very much. Data files that may hold new data each time should be held until the data they contain has been transferred to a larger computer, **and** has been backed up there as well. But it all really comes down to how important your data is. It never ceases to amaze me how lackadaisical some people can be with important data. In fact it often amazes me how lackadaisical I can be with my data! So be warned and start a good backup system early.

File backup using the Filer

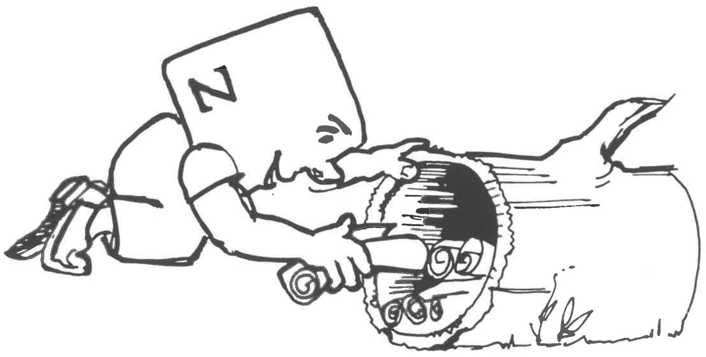
The most obvious use of EPROM is for direct backup of data or schema files. At convenient points in the data collection process, the Z88 Filer can be called and the data can be shuffled off to EPROM. If there is an earlier version of this file on the EPROM there already, then that will be marked as deleted, but will not in fact have any of its contents modified in any way at all. The new version is stored at the end of the list of files already on the EPROM and life goes on. In an emergency it is possible to recover earlier versions of EPROM files and procedures to do this are provided on the disk accompanying Infomate. The BBC BASIC program RECOVER.BAS, and its associated documentation RECOVER.DOC which can be inspected using Pipedream will allow the EPROM to be catalogued, and all versions of files on it brought back to life.

To use the Filer to back up a file to EPROM, move to the Filer using □F . Mark the files that you wish to move to EPROM by positioning the cursor on them and pressing the TAB key. Next move the cursor to the left hand column item "SAVE TO EPROM" and press ENTER . after lots of flashing the files will be on EPROM and may be inspected there using the Filer's "CATALOGUE EPROM" command. This is the normal technique to save a **schema** file on EPROM.

Installing an EPROM

If you are using Infomate and you sneak an EPROM into the front of the Z88 without telling anybody, then the Z88 will not know it is there if it is a blank EPROM. To tell the Z88 what you are doing it is easiest to insert the EPROM while you are in the Filer or Index modules. Alternatively you can use Infomate's Install EPROM command to automatically move to the Filer and register the EPROM

Logging data to EPROM



Somebody once said "wouldn't it be nice if we could save our data to EPROM after capturing each record. Wouldn't we feel so secure!". So after much grunting and sweating by the author, the possibility of logging data to EPROM was introduced. This has now become the recommended way of automatically providing a backup copy of your data. By logging to EPROM each record is saved in a form that is immune from power failure or computer malfunction, as and when it is created with little or no overhead on the data capture process. It is also more efficient on EPROM space than taking a copy of the entire file and placing it on EPROM at regular intervals, which requires the same data to be repeatedly placed on EPROM.

The concept of a log file is a slight fiddle in terms of doing things the Z88 way, but seems to work well. Normal files are stored on EPROM in a single operation which records first how long they are and then stores the data itself. This allows the operating system to later climb through all of the files on EPROM and locate each file, and the end of the files. Infomate EPROM log files work differently.

Firstly when you open an EPROM log file you do not know how big it will eventually be, so you cannot store its length. This is solved by inserting a very large length into the length place in a way that can later be modified, and then just writing the data into the file as it comes along. This has the effect of creating (temporarily) a very large EPROM file, as long as the EPROM itself in fact, thus reserving all of the remaining EPROM space for your log data. If at any time you wish to store other files on the EPROM then you must ask Infomate to close the file. When this is done using the Catalogue EPROM command, the file length is correctly recorded and the EPROM becomes completely normal again. Because Filer does not know about log files, any attempt to save data to EPROM when a log file is open will cause the Filer to say "Cannot satisfy request" which really means "I'm so confused I just don't know what to say". Unfortunately Filer also uses this message to mean "I've tried quite hard to write some data on a part of this EPROM and I've failed". This would happen if an EPROM were not erased correctly for example.

So if that is the theory of the matter, how do you actually log data on an EPROM. Its quite easy! Add one line to your schema file (that holder of all your hopes and desires) and Bob's your aunty. With the schema we have been playing with up until now, the change appears in the line 3 like this.

```
TITLE       Fruit Wt Collector 30/7 RBJ
FILE        WEIGHTS.DAT
LOG         WEIGHTS                                   \⇦ There it is
ID
     Fruit_No      N3
DATA
     Weight        N4.1
SEQUENCE
     ENTER         Fruit_No       Key in the Fruit No.
     MAKERECORD
     ENTER         Weight         Key in Fruit weight
END
```

Now you might notice that the LOG statement does not have an extension on the LOG file name; this is added on automatically by Infomate as the log file is opened and closed. The extensions follow the pattern of adding ".LG0" to the first log file created, ".LG1" to the second and so on up to ".LGZ". You can only use the schema file above if you have an EPROM in place. Try it.

Data File backup using Infomate

If you are not logging to EPROM, your data files may need to be copied onto EPROM more often than schema files. So a special command is incorporated into Infomate to perform this task directly. In fact the command just takes the Z88 through the same steps that you would use if you used the Filer. This will be obvious when you run the "Backup to EPROM" module in Infomate. One additional feature of the Infomate version of this command is that Infomate will first check if there is enough space on EPROM before proceeding. If there is not then Infomate will give up in a tidy way. Note that if you are using log files (see below) you will need to close that log file before you can save your data to RAMdisk. See the notes in the EPROM cataloguing section below.

Cataloguing the EPROM and things

While the filer provides an EPROM Catalogue command, this does not provide much information. Deleted files do not appear, it doesn't tell you the file lengths, it doesn't tell you how much space is left, and it does not handle log files. So we wrote a new one called "EPROM Catalogue" in the Infomate menu. It is not fast, but it tells you all you ever wanted to know about your EPROM files. The Infomate EPROM catalogue has one additional feature, it allows you to close any log files that are open. These files do not have to be created by that particular Infomate so you could in fact close an (Info)mate's log files by putting their EPROM into your Z88!

Recovering the log

When the EPROM log file has been closed by the catalogue command in Infomate, it becomes a **fully** compatible Z88 file and may be loaded back into RAMdisk using Filer commands. Its data will be in chronological order of its construction and will usually be an exact replica of the data file. Any difference will only be in the order of the data in the file. Because the log can be closed and reopened a number of times your data may be fragmented over a number of log files which can be merged together using Pipedream using the ◇FL command with the insert-at-cursor option. Note that when loading your data files into Pipedream and resaving them, the size of every record must match the schema file definition. If any single record has one more or one less character than this, then Infomate will become horribly confused. Further data on this point will be described later in the miscellaneous topics chapter.

Logging to a Printer

Rather than logging data to EPROM, it is sometimes nice to log your data to a printer. This gives you a nice piece of paper which even if it gets wet and muddy gives you that ring of confidence feeling as you drive back from your day of data capture. Providing you have a suitable printer, logging data to it is easy. Insert the statement

```
LOG  PRINTER
```

after the FILE statement in your schema and the data will be logged to the printer in the same way that it was logged to EPROM above. Note that as with all Infomate reserved words use PRINTER (in capitals) not Printer. The latter will save data to a log file on EPROM called Printer.LG0 which is probably not what you intended.

If your printer needs special care in the baud rate or parity department, this can be handled by setting the printer to the default settings for the Z88 (9600 baud, No Parity, No Xon/Xoff flow control), by setting the appropriate panel settings using □ S , or directly using Infomate. For example

```
LOG  PRINTER[1200EN]
```

will run your printer at 1200 baud with Even Parity and No Xon/Xoff flow control. Note there is no gap between the characters R and [. Details of the available options here are provided under PORTIS in the Reference section.

Unfortunately as the Z88 has only one serial port, it is not possible to log to the printer, and capture data directly from scales and other equipment connected to the serial port. If this conflict arises try logging your data to EPROM which is much tidier.

UPDATING AN EXISTING DATA FILE

All of the discussion to date has been about creating new data files. In many situations new data is to be added to an existing file rather than into completely new records each time. Two examples of this situation are provided below.

Getting your data bit by bit

In some data collection tasks it is not possible to collect all of the measurements relating to a single record at the same time. A typical example would be in the collection of herbage drymatters which use the green weight of the grass collected in the field, the weight of a sample of that grass before it is oven dried, and a post drying weight. All of these figures go into the melting pot to produce a value for dry matter per hectare for each plot. Can Infomate handle such a task? It most certainly can. The following example assumes that all data is to be entered into the Z88 by hand, and that a log is to be kept so that all stages of the data capture are secure.

Task 1 is to collect green weights. It is performed by the following schema file. Note that space has been provided for the pre and post oven data even though this schema does not collect that data.

So here goes. This file should be saved as HERBGRN.SCH .

```
TITLE        Drymatter Suite - Green Capture RBJ 31/7
FILE         HERBIE.DAT
LOG          HERBIE
ID
        Plot_No    N2
DATA
        Gn_Wt     N5.1
        Sm_Wt     N5.1
        Dy_Wt     N5.1
        Yield     N4.1
SEQUENCE
         ENTER    Plot_No   Keyin Plot Number
         MAKERECORD
         ENTER    Gn_Wt      Keyin Green Weight(kgs)
END
```

Well there isn't very much there that has not been done before. There are more data fields, and some aren't mentioned again after the definition stage. But it will allow you to fetch green weights.

So having done that lets add some pre oven sample weights using the following schema that we will call HERBSAM.SCH .

```
TITLE        Drymatter Suite - Sample Capture RBJ 31/7
FILE         HERBIE.DAT
LOG          HERBIE
ID
        Plot_No    N2
DATA
        Gn_Wt     N5.1
        Sm_Wt     N5.1
        Dy_Wt     N5.1
        Yield     N4.1
SEQUENCE
         ENTER    Plot_No   Keyin Plot Number
         FINDRECORD
         ENTER    Sm_Wt      Keyin Sample Weight(gms)
END
```

And that is almost the same as the first schema except that it fetches the sample weight field rather than that for green weight. **Also** there is a slight difference in the third to last line. That statement

FINDRECORD looks through the file that already exists and locates the record that has the same Plot_No . The new data is added to the file and it is stored in RAMdisk again.

You can probably guess what the third schema looks like. Let us call this one HERBDRY.SCH .

```
TITLE          Drymatter Suite - Post Oven Capture RBJ 31/7
FILE           HERBIE.DAT
LOG            HERBIE
ID
        Plot_No    N2
DATA
        Gn_Wt      N5.1
        Sm_Wt      N5.1
        Dy_Wt      N5.1
        Yield      N4.1
SEQUENCE
        ENTER    Plot_No   Keyin Plot Number
        FINDRECORD
        ENTER    Dy_Wt      Keyin Dried Weight(gms)
        LET      Yield = Gn_Wt * Dy_Wt / Sm_Wt * 10000
END
```

Well it is almost the same except for that line second from the end. This line calculates the drymatter per hectare assuming that the plot that was cut was 1 square metre. It doesn't look too complicated. More details on the LET statement can be found later in this manual. End of task. Each of the three schema files above would be saved with the names indicated and the operator would compile and run them as required by the task to be performed. Thus, if green weights were to be collected, compile and run the HERBGRN.SCH etc.

Because these three schema files are so similar they would be created by typing the first schema, saving it, making the required changes and saving again with a new name. This would be repeated a third time to obtain the third schema.

During each stage of the data capture itself, the records would be logged onto EPROM so that at the end we would have three copies of the data records, each copy becoming more and more complete. You may see many possibilities for this sort of data capture.

An animal weighing example

In many animal weighing tasks the data describing the animals IDentification and perhaps its previous weight already exist and a new field of data is to be added to the file at each weighing. This previous information most likely comes from a larger computer and its presence allows quite a few checks to be made on the data as it is collected. This improves the integrity of the new data. To demonstrate this an example will be worked through that assumes the data for animal ID and previous weight already exists in a file. If you want to try this you will have to create the earlier file either on another computer, or if you are clever, by writing a little schema file to do the task for you.

The classic animal weighing task which performs full checks on ID, animal already weighed, and comparing current weight against previous weight is as follows.

```
TITLE    Animal Weighing Demo
FILE     ANIMAL.DAT
LOG      PRINTER
ID
         Tag               N3
         Colour            L1
DATA
         Old_weight        N4.1
         New_weight        N4.1
SEQUENCE
         ENTER   Tag       Enter Tag No
         ENTER   Colour    Enter Tag Colour
         FINDRECORD
            IFFAILGOTO     ENTER Tag
         CHECKDONE         New_weight
            IFFAILGOTO     ENTER Tag
LABEL 1
         ENTER   New_weight      Enter WT(nn.n)
         CHECK   New_weight      Old_weight  -1.0  +3.0
            IFFAILGOTO   LABEL 1
END
```

There are a few things in this schema that look interesting. Most of it is pretty standard until we see the IFFAILGOTO . This statement follows a FINDRECORD and if that fails to find the record with the ID typed in, then it will take the operator back to the ENTER Tag statement so that they can re enter the ID. The CHECKDONE statement is ensuring that the field New_Weight does not already have a weight entered in it. If it does, then an animal purporting to be the one with the tag just entered has already been weighed. Clearly one of them is an impostor. Once again an IFFAILGOTO takes the operator back to the ID entry.

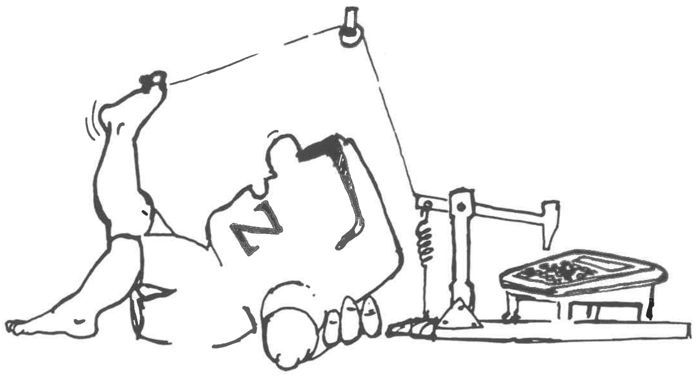
Once the operator gets through all of this ID checking stuff there is still a potential problem ahead. After the weight has been entered it is checked against the earlier recorded weight. If it is more than 1kg below the previous weight, or more than 3kg above, then suspicions are raised and the animal weight is asked for again. Finally the animal data is recorded, and this time it is logged on a printer rather than on an EPROM.

It is worth noting here that the CHECKDONE statement produces a more serious FAIL than does the CHECK. This is because, intuitively, if you continue after a CHECKDONE has failed you will be overwriting data that you had collected earlier. By contrast, a CHECK failure does not imply destruction of data. Infomate handles this by allowing you to continue with a warning after a CHECK statement fails, but preventing you from continuing after failing a CHECKDONE.

An example of a short segment of the data in an animal weight file conforming to the above is shown below.

```
504031.0
514022.824.2
514R33.935.1
522B34.0
531Y27.028.9
etc.
```

GETTING LAZY WITH THE ENTER STATEMENT



A major problem that occurs whenever anybody does something a lot is that they sometimes start thinking "wouldn't be nice if this could be done with a little less effort" . As a result, somebody else (i.e. me) who wasn't contributing any effort towards the original task is given a bigger task so that the original person doesn't have to have such a big task. If you catch my drift. Well that happened to the very earliest Infomate that only had one ENTER statement. These are some of the hopes that were expressed. Try the solutions offered in your own schema files.

PROBLEM "I get 2 records from each plot and don't want to type the plot number twice".

Solution: use **ENTER?<** rather than ENTER and you will be given the option of using the previous value that you typed by just pressing the ENTER key. The value previously entered will be displayed and if you wish you can edit that previous value. Think of this **ENTER?<** as ENTER-Optional-Previous. i.e. the ? means optional and the < means previous.

PROBLEM "All my data comes from one plot and I only want to enter plot number once".

Solution: use **ENTER=<** rather than ENTER or ENTER?< . This will force the value for this record to take on the value typed previously for this field. Call **ENTER=<** by the name ENTER-Equal-Previous. Now if the value is forced to be that used previously, it implies that once you have typed it once you cannot change it. Of course this cannot be so. To change it type an ⇧ key when at the next ENTER statement in the sequence and you will be taken back to the previously executed ENTER and will be allowed to edit the previous value. This scheme can be used throughout data entry to change your mind about what you want to put in the current record. Once you have entered the correct value use ⇩ or ENTER to move forward again.

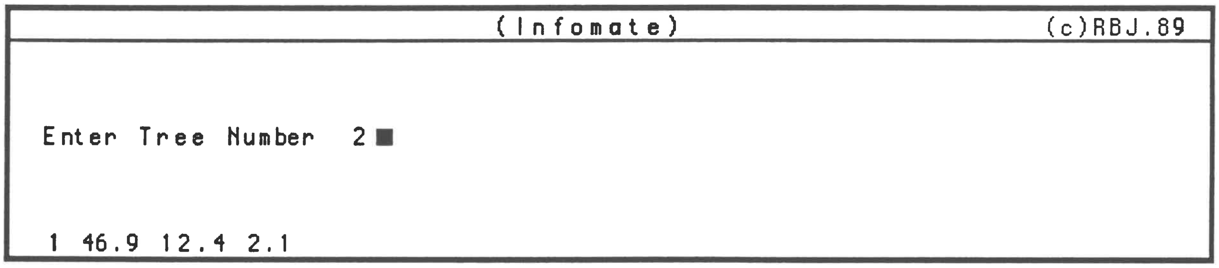
PROBLEM "Each plot number is the previous plot number plus 1, can't Infomate help"?

There are two options here and they are closely related to the previous ENTERs. They are **ENTER?<+** and **ENTER=<+** which are called ENTER-Optional-Previous-Plus and ENTER-Equal-Previous-Plus respectively. Each of them will add 1 to the previous value typed and will either use this as an optional value (the ?<+ case) which you can edit or accept by pressing ENTER , or will force the incremented value into the field (the =<+ case).

These incrementing ENTERs are probably best demonstrated by a short segment of a sequence. This sequence collects data from a number of rows (row number only changing occasionally) of sequentially numbered trees.

```
ENTER=<     Row_No     Enter Row number
ENTER?<+    Tree_No    Enter Tree number
```

On the first time through, the sequence will ask for row number (operator types 1), then tree number (operator types 1) and further data is collected. Second time through, the row number for the second record is set at 1 and the operator sees



```
                          (Infomate)                     (c)RBJ.89

 Enter Tree Number  2■


 1 46.9 12.4 2.1
```

By just pressing ENTER the tree number will be set at 2 and processing may continue. When the end of row is reached, the Tree number offered will be one greater than the number in the row and the operator knows that a new row number must be entered. This is achieved by pressing ⇧ to move back to the Row number statement, entering a row number of 2, and then entering a tree number of 1 again. Note that the ⇧ operation is the only way that you can change the value in an ENTER=< or ENTER=<+ statement.

I suggest that you try one of these examples to get the hang of it. Take care when using two ENTER=<+ statements in the same schema. Both will increment for each record which is most likely not what you want.

PROBLEM "I work one row from tree 1 to 50 and the next from 50 to 1, what do I do"?

There are narks in all walks of life! There is a solution, although it is a bit of a fiddle. **If** your tree number field in the above example is an N2 field then to create a negative counting sequence enter a value of -150 (note the minus). Now -150 plus 1 equals -149. If you only look at the last two digits then you have created the sequence 50, 49, 48, ... And what's more it works. When the sequence gets to 00 its time to use the⇧ key to return to the row number question again and add one to the row number.

PROBLEM "My plots are numbered 1 to 8 and I'm sick of pressing $\boxed{\text{ENTER}}$ "

Some people! But I suppose I'd better oblige. Use an **ENTER¦** then you will not have to press $\boxed{\text{ENTER}}$ provided you type enough characters for the field. For the N1 field used for plot numbers by the questioner, just press the plot digit. If there were 50 plots then you would need to press two characters. e.g. for plot 52 press a **5** then **2** , and for plot 6 press a **6** then $\boxed{\text{ENTER}}$, or **0** then **6**.

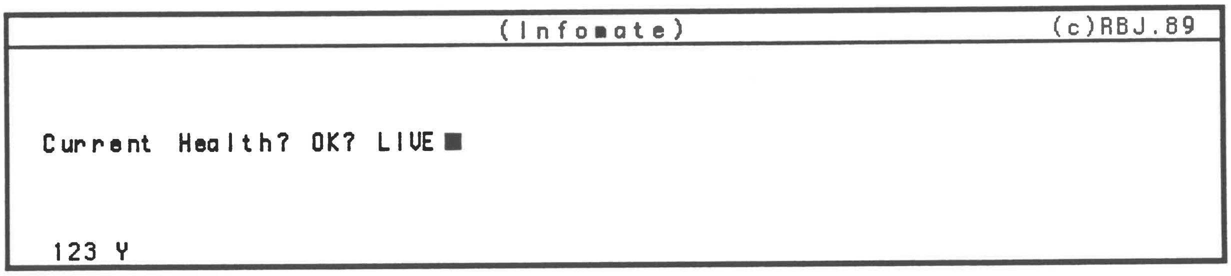
PROBLEM "I think of my animal IDs as a single unit even though it uses 3 fields"

There is a lot of this going around in animal circles. An example is a two field animal ID comprising a three digit tag number and single letter tag colour i.e. N3 plus L1 . The animal handler might call out "128Yellow". This would be translated by the Infomate operator as 128$\boxed{\text{ENTER}}$ Y$\boxed{\text{ENTER}}$ which is a bit of a handful. Two options are possible here. Use an **ENTERID** which allows you to type 128Y$\boxed{\text{ENTER}}$ without the intervening $\boxed{\text{ENTER}}$. Alternatively use an **ENTERID¦** which allows you to type 128Y without any $\boxed{\text{ENTER}}$ s at all. Both of the statements fetch an entire ID region in a single line and dissect it down into its component parts.

Now these ENTERIDs are a bit clever when you have tag numbers that are shorter than the full quid as it were. In the above example the old man of the flock, with tag number 1-Green can be handled by pressing 1G$\boxed{\text{ENTER}}$ or 001G or 01G$\boxed{\text{ENTER}}$ whichever takes your fancy at the time. Remember however that it must be possible to sort out which character comes from each field. If you had a two digit row number and a two digit tree number, an operator pressing 123 to mean row 1 tree 23 should not be disappointed to see the data entered as row 12 tree 3. We cannot please everyone.

PROBLEM "I want to edit the data that was previously held in the file"

An example here is a status (not that held by yuppies). I believe that it is common for animals to be proclaimed dead when they didn't turn up for last months weigh-in and then to turn up again live and well. So if an L4 field were set up as State_of_Health this would normally show the characters LIVE. If this field were processed by an **ENTER?** statement the screen might show



```
                    (Infomate)                    (c)RBJ.89

 Current  Health? OK? LIVE ■


 123 Y
```

Press $\boxed{\text{ENTER}}$ if the animal in the race is alive, and type **DEAD**$\boxed{\text{ENTER}}$ if it has just died.

CHECKING FOR ERRORS

Errors are very easy to make in field situations and Infomate would not be very useful if these were not easily located. Infomate can be programmed using the schema file of course to check data as it appears. But first, let us look at the philosophy behind Infomate errors.

Infomate error Philosophy

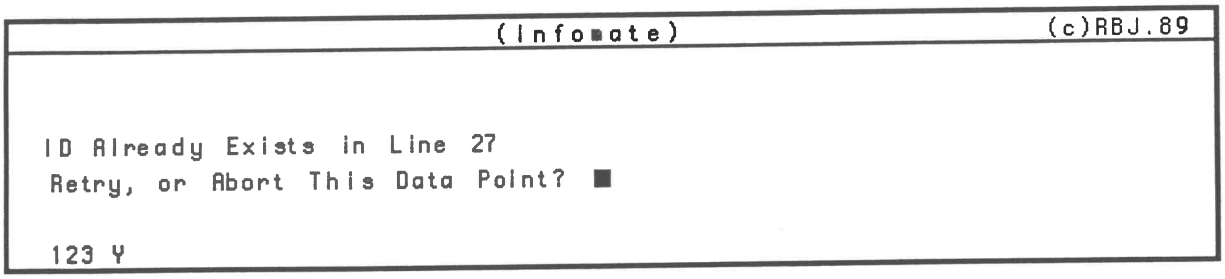
In the Animal weight capture examples above, the concept of error checking was introduced, but was not explained fully. In this section all of the CHECK statements available in Infomate will be discussed, and the various GOTOs as well.

The philosophy of errors in Infomate is that if a non catastrophic error is detected during the running of a sequence, it is not displayed immediately, but rather it is recorded for the operator to test. Errors come in two types which we call FAIL or WARN. A FAIL error is one that will eventually prevent you from proceeding. An example is if a FINDRECORD does not find the record you specify. If a record is not found then there is no way that the data can be stored because Infomate does not know where to store that record. That is a FAIL error. By contrast a WARN error is one that is left for the operator to decide if it is of sufficient concern to stop further processing of this record or not. An example of a WARN message is one where an animal weight is checked against a weight range and is found to lie outside the typical range. This could be caused by a weighing problem such as two animals on the scales, or by obesity. The operator can quickly determine the problem and make a decision. That is a WARN error.

A typical error check might follow a MAKERECORD like this

```
ENTERID        Enter ID of animal
MAKERECORD
IFFAILGOTO     ENTERID
```

Every time the ID entered is found to already exist in the file the operator is given the following options



```
(Infomate)                                        (c)RBJ.89

ID Already Exists in Line 27
Retry, or Abort This Data Point? ■

123 Y
```

To which you can type R to return to the Line suggested in the IFFAILGOTO line which probably allows you to re-type the ID again, or type A and return to the main menu.

If the IFFAILGOTO statement were not in the file then the operator would not discover the error until the END statement was reached. At that point Infomate would tell you what the error was, and if requested to Retry, would go back to the first step in the sequence so that you could step through and correct the error. Normally it is best to use an IFFAILGOTO statement so that the error is detected and corrected as soon as possible. In terms of errors you can think of the END statement as the following pair of lines

```
IFFAILGOTO      SEQUENCE
END
```

which restarts the sequence from line 1 if an error is discovered in the sequence.

In addition to these two error states, Infomate can post a catastrophic error which requires immediate attention. An example is when a RAMdisk file is full. Such errors are not able to be handled by a schema file IFFAILGOTO statement.

Checking that all is well

In addition to the errors that are posted by the MAKERECORD and FINDRECORD, there are a number of error states that can be checked by the schema file writer. These will check the condition specified and if a problem occurs set either the FAIL or WARN error states as appropriate.

The simplest CHECK statement is the CHECKDONE which looks at a field in an existing file to see if the field specified has a value in it or is in fact blank. You might use the CHECKDONE to see if an animal had already been weighed like this

```
ENTERID       Enter the ID
IFFAILGOTO    ENTERID
CHECKDONE     New_Weight
IFFAILGOTO    ENTERID
```

Now the CHECKDONE statement is a FAIL type error checker. If the animal has been weighed then the data in the field must be protected at all cost. It may well be that the data is in the wrong place, but that does not change the fact that the data there is data. To solve this problem you might have to go back to the View/Edit mode and move the data to its rightful place and put blanks in the field space before proceeding. Alternatively open up a Notes file in Pipedream and type in sufficient information to recover the problem at a later date.

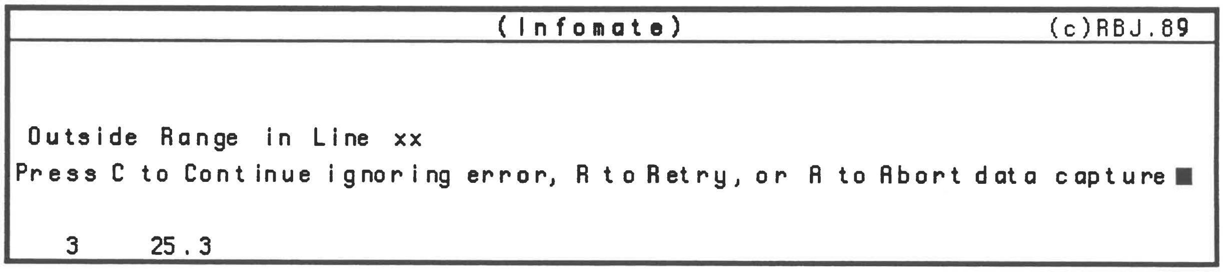
Two further CHECK statements allow you to check numeric ranges or for specific letter patterns. The numeric checker has a format like this

```
CHECK    New_Weight    Old_Weight     -1.0    +3.0
```

which is a WARN error checker testing that the field New_Weight is in the range

Old_Weight - 1.0 <= New_Weight <= Old_Weight + 3.0

If it is not, it will post the WARNing message "Outside Range in Line xx" like this:



```
                              (Infomate)                              (c)RBJ.89


 Outside Range in Line xx
Press C to Continue ignoring error, R to Retry, or A to Abort data capture ■

   3     25.3
```

to which pressing a C will allow you to ignore the WARNing message, pressing R will take you back to the line indicated in the IFFAILGOTO statement, and an A will return you to the main menu.

This form of check can also be used to test if a field has a value inside a specific numeric range using a # character in place of the second field name above. Thus to test if a value will fit inside an N2 field named Blotch_Count use

```
CHECK    Blotch_Count    #    0    99
```

which checks that Blotch_Count is in the range

```
0   <=   Blotch_Count   <=   99  .
```

Note that the # in this case should be read as "...is a number between..." .

To check a letter field requires a slightly different format. Normally it is necessary to make sure that the letter field entered is one of a relatively small set of options. An example to check if the sex letter typed was an M or an F you could use

```
CHECK    Sex    *MF
```

or if you didn't mind which case the letters were typed in use

```
CHECK    Sex    *MFmf
```

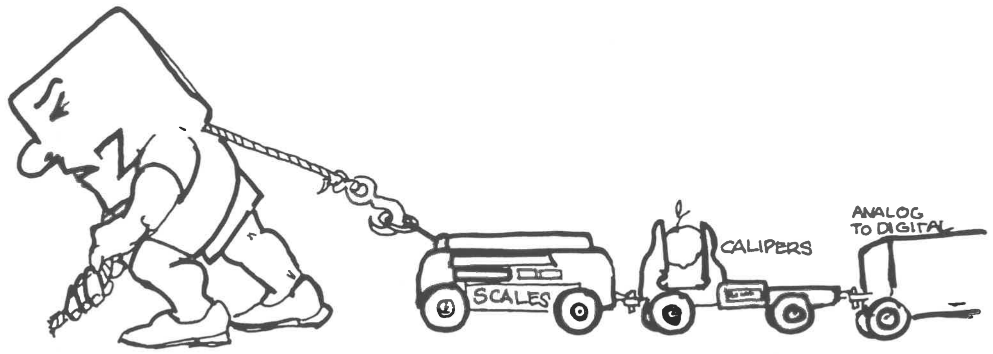
either of which will post a WARNing message "Check Failed in Line xx" .

The meaning of the asterisk (*) in the above lines is to inform Infomate how many characters are involved in the check. Thus if you wanted to check two character sequences, use two asterisks like in the next line

```
CHECK    Colour    **BKBRRDORYLGNBUVIGYWH
```

which will only allow the character pairs BK, BR, RD, OR, YL, GN, BU, VI, GY, WH. Note that ** is not a valid character pair in this case, but rather, it just lets Infomate know the size of the groups.

USING THE SERIAL PORT AND SCALES



Up to this point all operations placing data into Infomate files have involved the keyboard and various types of ENTER statements. Much data capture in the field involves the use of weighing equipment. The transcription of weights from the scales to Infomate by hand is obviously a step to be circumvented if possible. The approach taken in Infomate is to provide "drivers" for the different devices that might be connected to Infomate and to make these drivers handle the idiosyncrasies of each device and make them all look if not the same then at least similar. Because the Z88 only has a single connection point (or port as we call it) for such devices, then only one driver can be in vogue at a time. This does not mean that Infomate is limited to a single device on the port, but that if more than one device is connected, then a driver for that combination must be written. Initially the number of devices that are supported by Infomate will be limited to a few of the more popular weigh scales and a set of electronic calipers and an analogue to digital converter. This list will grow with Infomate.

To define which driver Infomate is to use, a single PORTIS line is added to the schema file immediately after the LOG line, or the FILE line if logging is not enabled. This line has one of the following forms

```
PORTIS       METTLER_1
PORTIS       SARTORIUS_1
PORTIS       TRUTEST_1
```

depending on which scale is to be connected. A full list is provided under the PORTIS statement in the reference section of the manual, and in the HELP facility of Infomate.

In some circumstances it is possible to insert into this line any special initialising sequences that are required. For example to set the Mettler driver to measure in imperial ounces you would use a line like

```
PORTIS       METTLER_1    U oz
```

Additionally, the baud rate, parity and Xon/Xoff status can be controlled by appending this information to the Driver name like this.

```
PORTIS        SARTORIUS_1[1200NY]
```

which sets the baudrate to 1200 baud, the parity to None and enables (Yes) the Xon/Xoff control. Full details of the baudrate options and the drivers are provided at the rear of the manual.

OK, so that tells Infomate where to find the driver, how does Infomate fetch a value from that driver? Provided the scale being used is capable of delivering a weight when the scales are steady then data capture from the scales is particularly simple. Connect the scales to the Z88 and replace the ENTER statement that would be used to collect the weight from the keyboard with a GETPORTstatement. Thus

```
ENTER      Weight     Enter Nashi weight
```

becomes

```
GETPORT   Weight    Nashi Weight is
```

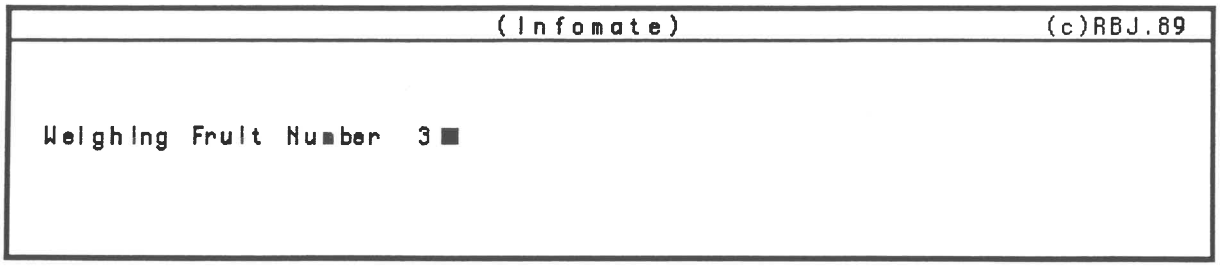
where the message at the right is now displayed briefly after weighing the nashi and processing continues.

A slightly different approach is required when using the GETPORT statement compared to the ENTER statement. The ENTER automatically stops and waits for you to do something, while the GETPORT just motors on. You will also need to let the operator know what they are to weigh and make the sequence pause until the operator is ready. This can often be achieved as part of the IDentification process.

Thus to weigh a group of fruit using a purely sequential number series the following schema fragment may work for you.

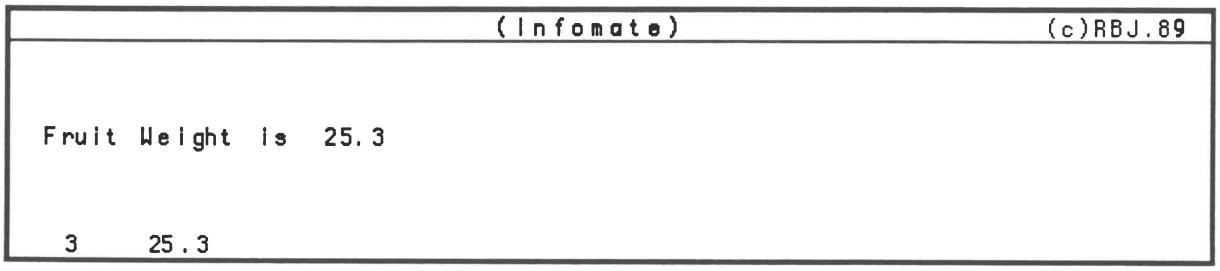
```
ENTER?<+     Fruit_Number     Weighing Fruit Number
MAKERECORD
GETPORT      Fruit_Weight     Fruit Weight is
```

This would produce the following screens



```
                                (Infomate)                        (c)RBJ.89

 Weighing Fruit Number  3■
```

If the fruit number was correct then place the fruit on the scales and press ENTER . You will see the brief display



```
                                (Infomate)                        (c)RBJ.89

 Fruit Weight is  25.3


   3     25.3
```

and will be returned to the fruit number step. If the operator needed to see the weight for visual checking purposes then add some ~ characters to the end of the GETPORT statement like this

```
GETPORT     Fruit_Weight      Fruit Weight is ~~~
```

This will cause the program to pause for about three seconds (1 second per ~) with the weight displayed, before continuing automatically. Alternatively a DISPLAY statement could be used. This is described in the reference section later in the manual.

Similarly if you wish to wake the operator up when the fruit is to be weighed, put some ! characters in the message. These will beep the Z88 speaker once for each ! character.

Some pieces of equipment that you might connect to the serial port require no human intervention at all. In fact it is a pain in these cases to have to observe some meaningless voltage being displayed to nobody in particular. In these cases leave the message area of the GETPORT statement blank and the GETPORT will proceed quietly with no bells and whistles, and without displaying its presence. Another place for this is where the number fetched needs some calculation performed on it before it is displayed.

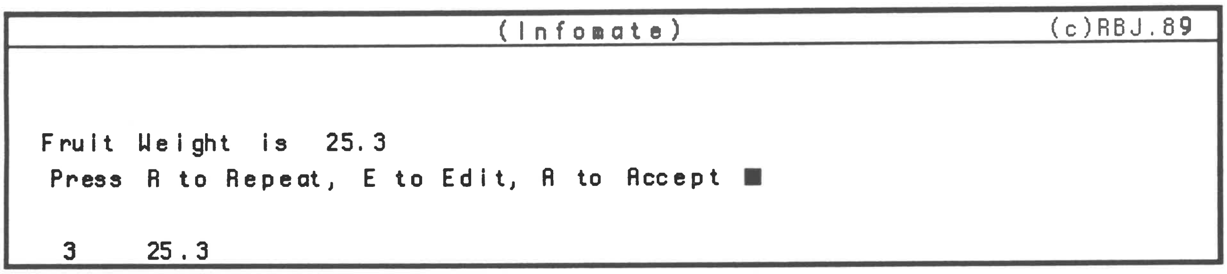
The following sequence segment shows this in the case where an imperial scale is being used to capture metric weights

```
DISPLAY  *       Load  scales!!
GETPORT  Imp_Wt
LET      Met_Wt   =   Imp_Wt/2.2
DISPLAY  Met_Wt  Wt  Was~~
```

Which is roughly equivalent to the following code for a metric scale

```
GETPORT  Met_Wt  Load  Scales!!~~
```

If at any stage you step backwards through your sequence to correct an earlier mistake, you are given the option of re-weighing the fruit, editing the weight that was obtained, or skipping over the GETPORT and using the value obtained like this



```
                              (Infomate)                         (c)RBJ.89


 Fruit Weight is  25.3
  Press R to Repeat, E to Edit, A to Accept ■


   3    25.3
```

The edit option may be useful in an animal weighing situation when the animal has left the weigh scale and is running at full tilt towards the far blue yonder, and therefore cannot be re-weighed.

The Driver Test Module

When you are setting up a link to an electronic scale or other equipment on the serial port it is usually inconvenient to create a special schema to test that the two pieces of equipment are communicating correctly. The Driver test module is the tool for this as it mimics all of the commands that you might use in a normal schema.

Before you can use the driver test module you must compile a short schema that contains the appropriate PORTIS statement, complete with any baud-rate and parity options if required. This schema will need an ID and DATA field, and a dummy sequence region. Probably the easiest way is to just grab any old schema and insert the appropriate PORTIS line and save it under a new name. Compile the schema and then after connecting the serial port equipment to the Z88, press the **D** key to execute the **"Driver Test Module"** from the main menu.

The Driver test will first initialise the serial port and driver selected, and then present you with a small menu of commands. If you type a **G** command it will simulate a GETPORT, and display the resulting values on the screen. An **S** command will mimic the SETPORT, but will first ask you to enter the message that the SETPORT is to send to the device. You can Tare the serial port device if Taring is appropriate and is allowed for by the driver using the T command. In some situations this may be the only way that you can Tare a remote device.

The **I** command will reinitialise the serial port device and the driver and may be needed if you have modified the parameters of the equipment connected. For example if the initial attempt at getting a weight from some scales showed a string of garbage characters, you may have set the scales baud-rate incorrectly. Change the baud-rate to the right value and issue an I command to get going again.

When all functions check out correctly you can return to the main menu using the **Q** key to access the Quit command

DESIGNING A SCHEMA FILE

In the previous chapter you have been presented with a number of schema files to perform various tasks and you have probably gained an idea how to form them. In this section a strategy for developing a schema file is introduced. If you follow this procedure for your schema design tasks you should end up with workable schemas every time.

All schema designs start with the design of the data file and once that is complete, the schema more or less creates itself. For this example we will use a horticultural task capturing two fruit diameters and a maturity score on five fruit collected at random from each tree in two rows of 10 trees. The fruit will be identified by row number, tree number and fruit number and the diameters will be recorded to 0.5 mm. Maturity scores will use the letters I, F, P, R, M. Additionally a comment field will be provided with about 10 to 12 characters.

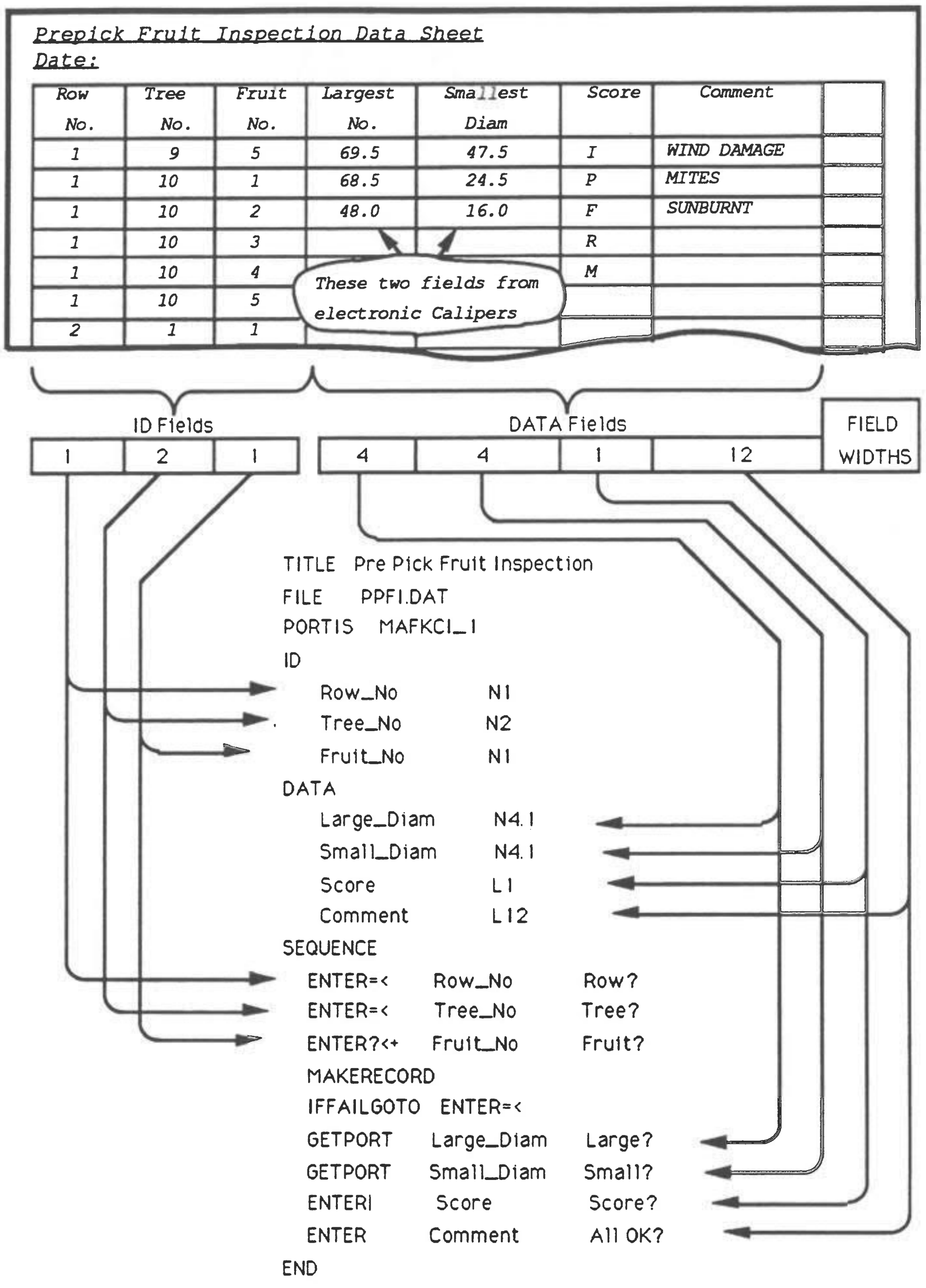
Let's first create a data sheet that contains as many of the unusual values that can go into each field as we can think of, as well as indicating the typical form of the results. This is the key step in the process and is shown opposite. From the data sheet we can see that the data columns are the fields and the data rows form the individual records for each tree. Each record has a unique ID formed from the three fields containing Row, Tree and Fruit Numbers. The four DATA fields for each record are respectively Largest Diameter, Smallest Diameter, Score, and Comment.

By inspecting the form of the numbers we can find the maximum width needed for each column, and can specify the two diameter columns to have one decimal place. A format of N4.1 is appropriate here.

The TITLE and FILE lines are specified from the data sheet title, and because we are to use Electronic calipers we need to define a driver (PORTIS MAFKCI_1) and use GETPORT statements to capture these numbers.

Because Row No and tree Number don't change very often the ENTER=< form of the ENTER statement is selected. This means these numbers will only be changed when we move to a new tree or new row. Fruit is always handled in a 1 to 5 sequence so Infomate can be given the task of incrementing that number.

When all ID's have been entered we must MAKE a RECORD to check that the ID is valid and that there is enough space. Each of the data fields can then be fetched. Note that the Score field must by definition be a single character so the ENTER¦ style is used. The comments at the end will normally remain empty but this gives us a chance to step back through the entry procedure if we decide to change our minds.



Prepick Fruit Inspection Data Sheet
Date:

Row No.	Tree No.	Fruit No.	Largest No.	Smallest Diam	Score	Comment	
1	9	5	69.5	47.5	I	WIND DAMAGE	
1	10	1	68.5	24.5	P	MITES	
1	10	2	48.0	16.0	F	SUNBURNT	
1	10	3			R		
1	10	4			M		
1	10	5					
2	1	1					

```
TITLE  Pre Pick Fruit Inspection
FILE     PPFI.DAT
PORTIS   MAFKCI_1
ID
   Row_No          N1
   Tree_No         N2
   Fruit_No        N1
DATA
   Large_Diam      N4.1
   Small_Diam      N4.1
   Score           L1
   Comment         L12
SEQUENCE
  ENTER=<     Row_No       Row?
  ENTER=<     Tree_No      Tree?
  ENTER?<+    Fruit_No     Fruit?
  MAKERECORD
  IFFAILGOTO  ENTER=<
  GETPORT     Large_Diam   Large?
  GETPORT     Small_Diam   Small?
  ENTERI      Score        Score?
  ENTER       Comment      All OK?
END
```

When a schema is seen in the form above, it looks very straight forward. We may wish to add various CHECKS to ensure that the data fields, (and perhaps the ID fields too) have reasonable values and that the Score is one of the letters I, P, F, R or M. These can easily be added if required.

ARITHMETIC AND THE MEMORY

In an earlier section a technique for performing arithmetic on field values was demonstrated briefly. In this section this technique is elaborated. The LET statement with its mathematical ability is probably the most powerful of all of the Infomate statements implemented. It allows the full power of the BBC BASIC as used on the Z88 to be incorporated into Infomate schema files to perform arithmetic operations on number fields or to divide up letter fields into pieces and then reassemble them in a different from. As before the usage of the LET statement will be demonstrated by examples.

Using the LET Statement

It is not unusual to have pairs of fields that always take on the same values in certain circumstances, or to have fields that are to be set to constant values. This can be achieved using LET statements of the form

```
LET    Field1   =   Field2
```

or

```
LET    Field3   =   6.2
```

In these examples Field1 and Field2 may be number fields or letter fields and Field3 must be a number field (remember that the fields are defined in the ID or data region of your schema). Note the spaces on either side of the = sign. The = sign should be read as "is replaced by" or "becomes" rather than "is equal to". If scaling had to be performed on a field value you could use the following form

```
LET    TempC   =   (TempF - 32) * 9 / 5
```

which will change readings from Fahrenheit to Centigrade scales.

The arithmetic on the right hand side of the = sign must conform to certain rules such as evaluating brackets first, multiply and divide next and plus and minus last. A full list of this calculation order or "precedence" is provided in the reference chapters of this manual. In addition to the usual algebraic components, Infomate supports a full range of trigonometrical and logical functions which are also listed in the reference section.

Examples of these are

```
LET    Area   =   Base ^ 2   * COT( RAD( Base_Angle))
LET    Area   =   PI *   Radius ^ 2
```

In association with these statements is a pseudo field (i.e. it behaves like a normal field in a file but it isn't in the file) called MEMORY. This may be used to store temporary results as part of the calculation process. This field is a Number field and is large enough to hold any practical number. An example of a calculation involving MEMORY is this monster which calculates humidity from wet and dry bulb temperatures.

```
LET  Vap      =  EXP(7.076-2.47*((1.46-(Dry/100))^2))
LET  MEMORY  =  EXP(7.076-2.47*((1.46-(Wet/100))^2))
LET  RH       =  (MEMORY-(.645+Dry*6.51E-4)*(Dry-Wet))/Vap
```

An alternative to this is to use SCRATCHPAD fields which are described later. One of the example schemas shows the use of SCRATCHPAD fields in humidity calculation.

Letter Field LETs

All of the above is fine for number fields, but what about letter fields? You can use LET statements for these too, but a different set of rules applies when adding groups of letters to each other. Thus

```
LET     State_Of_Health  =    "LIVE"
```

would set the field describing status to the four characters L, I, V and E. Note that the double quotes must be placed around the characters in the LET statement, but that they will not be placed in the file. You could achieve the same thing using

```
LET     State_of_Health  =    "LI" + "VE"
```

which demonstrates how strings of letters may be added together

There are also techniques to extract parts from groups of strings of letters. All the following are also equivalent to those above and produce a field with a "value" of "LIVE".

```
LET     State_Of_Health  =    LEFT$("LIVE  SNAKES",4)
LET     State_Of_Health  =    RIGHT$("SNAKES  ALIVE",4)
LET     State_Of_Health  =    MID$("OLIVER  TWIST",2,4)
```

With LEFT$, MID$ and RIGHT$ you can rip up and repair strings to your hearts content.

When is a Number not a Number?

It is sometimes necessary to set a Number Field to the "missing value", which is a string of blank characters in the Field. Now that poses two problems. The statement

```
LET NUMBER      =       "       "     \will not work
```

because there are characters (i.e. blanks) on the right and a number field on the left. Also, Infomate in its wisdom would reduce all of those nice space characters on the right to a single space. To solve both of these problem use

```
LET NUMBER$     =       " "
```

Note that the dollar at the end of the field name means "pretend this field is a letter field", and the single space character will be expanded to enough space characters to fill the field. Now you must be careful here because blanks are the only valid non numeric things that can occur in a number field. Thus

```
LET NUMBER$     =  "From the Left"   \does Not Work
```

MEMORY and letters

Earlier in this chapter you were introduced to the concept of the MEMORY pseudo number field. i.e.. MEMORY is like a field and stores numbers. You can also use MEMORY to store strings of characters by calling it MEMORY$. Now take care here because MEMORY and MEMORY$ are actually the same place in the Z88 memory and the $ just says whether to think of it as a number or as a string. You can therefore use

```
LET MEMORY$     =       "Don't forget this"
```

or any of the other string constructs above.

LET, TEST and the GOTO

The LET statement has a useful side effect which allows you to branch to different parts of your sequence file. Every time a Numeric LET is calculated, a flag is set to say whether the result was equal to zero or not. Your schema file can then test that flag to decide what to do next. It is normal to consider flags not as "zero/not zero" indicators, but rather as false/true indicators with zero being false. The TEST statement allows you to perform the same tests without actually storing the result.

An example of the use of the GOTO feature is when a schema file must collect different information depending on the value in a field. Consider this sequence

```
ENTER    WEATHER          Is it WET or FINE?
LET      MEMORY  =        (WEATHER  ="WET")
IFFGOTO  LABEL   FINE
ENTER    RAINFALL         How WET?
LABEL  FINE
```

Now the second line asks whether the weather is wet, and will set MEMORY to FALSE (or 0) if it is fine and to TRUE (not 0) if WET. The third line then asks IF the previous LET gave a False answer (i.e. 0) and if so would GOTO LABEL FINE in the sequence. This neatly by-passes the RAINFALL data entry when the weather is fine.

Now you do not have to use MEMORY in the LET statement, any field will do. In fact it is normal not to store the result of the test anywhere. You could equally well replace the second line with

```
TEST     (WEATHER  =  "WET")
```

which reads better anyway. In addition to the IFFGOTO there is an IFTGOTO with the opposite function of IF True GOTO .

The logic in these types of expressions can become quite complicated and the reader should record in the back of their minds constructs like

```
TEST   =   (Sex  =  "M") AND (Status  =  "LIVE")
IFTGOTO    LABEL  Man_Alive
```

which will only go to LABEL Man_Alive if both the field Sex is "M", and the field Status is "live.

Similarly,

```
TEST   =   (Sex  =  "M")  and  (Age  <  18)
IFTGOTO    LABEL    Boy
```

requires Sex to be "M" and age to be less than 18.

If you use the TEST statement to check the size of numeric fields, you should take care because the Z88 can represent numbers to about 9 decimal places, and that means that numbers that you think of as identical, Infomate may think are different. For example 6.0 and 5.99999999 are, as far as you and I are concerned, both 6 when it comes to counting kiwifruit. Infomate 'knows' that these numbers are different.

To avoid this use

```
TEST    Fruit_Count     <       6.5
```

rather than the more logical

```
TEST    Fruit_Count     <=      6
```

Similarly, use

```
TEST    ABS(Fruit_Count  -  6)   <  0.5
```

to test for Fruit_Count becoming equal to 6. While this may appear a little cumbersome, it is guaranteed to work, where

```
TEST    Fruit_Count     =       6
```

may fail sometimes.

LETs and Old values

You may recall in an earlier section the ENTER=< instruction was introduced. This allowed a value entered into the previous record to be used for the current record. This was useful when a field such as an orchard row number held the same value for a number of records. It is also possible to use this type of construction in LET statements. Thus to set a field to the value held in the last entered record use the term (<) immediately after the field name. For example

```
LET    Field1   =   Field1(<)
```

will move the value held in Field1 in the last record processed forward into this record. You can use the (<) term on any field name in any LET or TEST expression.

This concept of the current record and the previous record needs a little elaboration. When you start at the beginning of a Sequence region, all fields in the current record are set to blank so that you start with a clean slate. At the END of a sequence, and after the values have all been saved on RAMdisk, all of the "current record" values are stored in a special place called "previous record". Now you can select which of these two values to use by referring to the field by name which uses the current value in the field, or by using the field name followed by (<) which refers to the previous value. In a similar way the ENTER? statement always refers to the current record, while ENTER=< and ENTER?< refer to the previous record the first time they are used, but refer to the current record if they are backstepped to using the ⇧ key. The ENTER=<+ and ENTER?<+ instructions do the same but increment the value from the previous field first.

The SCRATCHPAD

No this is not the corner of the dog kennel, but is another little trick for the more experienced Infomate user. Often there is the need to have fields that are not to be stored in the file, but rather are just to take on intermediate values as part of the calculation process. Also it is often nice to have fields that carry values from record to record without the need to save them in the file. This is the function of the SCRATCHPAD.

Lets demonstrate SCRATCHPAD using a trivial example. We are to collect temperatures at a number of sites, but our thermometer measures in Fahrenheit and being a scientific trial the results must be recorded in centigrade. We could take a pocket calculator with us but surely the Z88 could do the task? Sure can! Consider this schema:

```
TITLE    Getting the TEMP
FILE     TEMP.DAT
ID
         Site_No N3
DATA
         Temp_C  N3.1
SCRATCHPAD
         Temp_F  N3.1
SEQUENCE
         ENTER   Site_No  Site?
         MAKERECORD
         ENTER   Temp_F   Temp in Degrees F
         LET     Temp_C   =      (Temp_F-32)*5/9
END
```

The only part of that schema that is different is that Temp_F is declared in a SCRATCHPAD area following the DATA area. The resulting record on RAMdisk would only contain the two fields Site_No and Temp_C; the Temp_F field is ignored and only serves as a calculation intermediate.

It is worth noting that scratchpad fields are completely optional, and that all of the scratchpad field values are maintained from record to record. Another way of thinking about these scratchpad fields is to view them as MEMORY fields that you have given names to; they behave exactly the same way, can be displayed, captured in a GETPORT or ENTER, and can even take part in the more sophisticated counting operations described later. The only thing that they will not do is allow you to have more than the maximum number of fields. i.e. they count as normal fields in terms of their space requirements in memory. Those who have programmed in higher level languages may like to think of scratchpad fields as global variables, or alternatively as temporaries depending on the application they are being put to.

When you refer to a scratchpad field using the (<) notation as for example Temp_F(<) , you are referring to the value that the SCRATCHPAD field held at the point that the previous record was saved to RAMdisk. This can be used to advantage if a field is to be changed to a value that depends on its previous value. For example, if you were counting data records, the statement

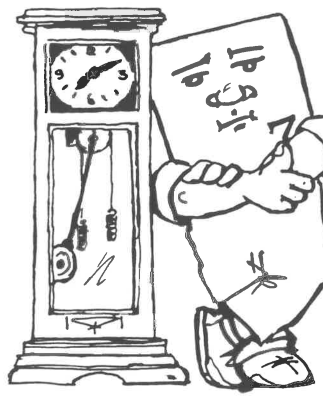
```
Count     =        Count+1
```

would increment everytime you passed through it even if you were just backstepping using the ⇧ key, whereas

```
Count     =        Count(<)+1
```

would behave predictably, and only increment once whatever path you took through the schema file.

TIME AND DATE AND THE STOP-WATCH



Inside the Z88 is a real time clock that can be used to time events or to record when they occurred. Provided that you keep the clock correctly set or at least don't do anything to upset it, you will be able to use a number of time keeping functions with the aid of the LET statement.

Recording time and date

The general form of the time function is TIME(argument), and it can be slotted at any position on the right hand side of a LET statement. The argument in the brackets is a way of letting you tell Infomate which particular time function you want. The following table lists the argument values along with their functions and the size of field that they will produce for the last second of 1991.

Function	Meaning	Format	Example
TIME(HMS)	Hours Mins Secs	N6	235959
TIME(HM)	Hours Minutes	N4	2359
TIME(DOW)	Day Of Week	N1	3
TIME(DOM)	Day Of Month	N2	31
TIME(M)	Month	N2	12
TIME(Y)	Year	N4	1991
TIME(DMY)	Day Month Year	N6	311291
TIME(DM)	Day Month	N4	3112
TIME(DOY)	Day Of Year	N3	365

Thus to set a field to the month number use the expression

```
LET   Month   =   TIME(M)
```

Month should be an N2 field unless you close down particularly early for Christmas when an N1 field will suffice.

You can truncate the values above by using shorter field sizes. For a two character year use a field called Year with an N2 type and use

```
LET    Year   =   TIME(Y)
```

The 1900 part will be chopped off without further ado.

Timing events

One important point about the times and dates above is that some are not true numbers and you cannot do arithmetic on them. Thus, computation of elapsed time should not use this schema segment.

```
LET   Elapsed_Time   =   Start_Time   -   TIME(HMS)
```

It will give you an extra 40 seconds in every minute and an extra 40 minutes in every hour. The reasons will be left to you to discover.

PROBLEM "But I actually want to measure elapsed time".

Do not fret dear friend, there is another function in Infomate that will allow you to measure elapsed time. This is the function called TIME(TIC) which counts in 1/100 th of a second intervals. It is set to zero every time you compile a new schema so you can use it as a time since I started this schema. It will last for some months before it gets too big for Infomate and will then start to behave a little strangely. It is not normal for an Infomate user to go for months without recompiling a schema so should not present problems. Another minor difficulty occurs if you adjust the Z88 clock back to the day before you last compiled a schema. Due to a bug in the Z88, negative time is not handled correctly in this case and the TIME(TIC) function suddenly becomes immense. But seeing that time travel is not possible even on a Z88 this should not present any difficulty either. If it does, just recompile the schema.

To use TIME(TIC) as an elapsed timer look at the following schema which was used to monitor service time through customs at Auckland International Airport. It records number of passengers in each group, how many bags they have and the date and time that they arrive at the customs desk. Also recorded is the elapsed time during their customs clearance.

```
TITLE   Airport Custom Service Investigation.
FILE    AIRPORTCS.DAT
ID
        Pax_Grp      N4
DATA
        Flt_Num      L7
        Date         N6
        Start_T      N4
        No_Pax       N2
        Bags         N2
        Service_T    N3
SEQUENCE
     LABEL 0
        ENTER=<+     Pax_Grp       Pax Group Num?
        MAKERECORD
        IFFAILGOTO                 LABEL 0
        ENTER=<      Flt_Num       Flight Number?
        LET          Date          = TIME(DMY)
        LET          Start_T       = TIME(-HM)
        LET          MEMORY        = TIME(TIC)
        ENTER        No_Pax        Enter No of Passengers?
        ENTER        Bags          No of Bags?
        DISPLAY      Pax_Grp       Hit ENTER to End
        LET          Service_T     = INT((TIME(TIC)-MEMORY)/100)
END
```

This is an interesting schema file to try out. Type it in as shown and imagine you are recording the movement of FLT TE120 through customs. If there is a gap in the flow of people you will need to press the ⇧ to take you back to the Pax_Grp question so that a realistic start time for the group after the gap can be taken. Otherwise the start of one group is assumed to be the finish time of the previous group.

The midnight problem and how to avoid it

You may have noticed a strange minus sign in the LET statements above which fetches TIME(-HM). This avoids the midnight problem! Read on. Even though Infomate is reasonably fast at computing expressions, it does take time. In certain circumstances it is possible that in the short space of time between when Infomate fetches a date, and when it fetches the time, that the date may have changed. This only happens if the date was taken just before midnight, and the time a very short time later but long enough to pass the magic hour. You end up with yesterdays date and todays time which will look perfectly valid in a months time, but will nonetheless be wrong. To avoid this, Infomate allows you to use a negative argument to the TIME() function. If you use this it will not look at the clock again, but rather will use the time and date that was collected for the previous TIME call. That is why in the example above the first call is to TIME(DMY) and the second one to TIME(-HM) . The DMY call actually captures a complete record of the time and date, and the -HM call just extracts the bits from it that it requires.

You can use this to advantage to allow you to record the time since the last record was taken by placing a positive TIME() at the end of your sequence and using a negative one at the start like this

```
LET     Last_Sample     =       TIME(-HM)
.
etc.
.
LET     MEMORY  =       TIME(HM)
```

The field Last_Sample will end up with the time taken at the previous cycle of the sequence, and MEMORY will be forgotten.

SLEEP and other animal behaviour

Infomate has a special schema statement which puts it to sleep. This is one way of automatically shutting down Infomate between observations if you wish to conserve power. SLEEP might be useful in behaviour type studies where there is often long periods of nothing between the active periods. By placing SLEEP at the start of a schema the Z88: will close down right at the start and wait for you to press the two $\boxed{\text{SHIFT}}$ keys. Immediately you do this it will continue the schema perhaps by first recording the time of the event.

Those further interested in SLEEP should study this night watchman schema. Every time that the night watchman passes the Z88 he has to press the two shift keys to wake it up. It will record the time and go back to sleep. That way it saves batteries and ensures that the night watchman has both hands on the job.

```
TITLE    Night-watchman  watcher  RBJ  18/7
FILE     WHENHE.DID
ID
         Event        N2
DATA
         The_Time     N4
SEQUENCE
      LABEL  START
         LET          MEMORY       =  TIME(TIC)
         SLEEP
         ENTER=<+     Event        Event  Number
         LET          The_Time     =  TIME(HM)
         TEST         TIME(TIC)-MEMORY)  <  6000
         IFFGOTO      END
         DISPLAY      The_Time     YerTooQuickMate
         GOTOLABEL    START
END
```

If the night watchman returns too quickly to the Z88 he will get a rude message and the clock will restart. Perhaps that is a little unfair.

The ALARM

Most of us associate ALARMs with SLEEP. So does Infomate if you wish. Using the SLEEP statement above you can put the Z88 to sleep. With the ALARM that is built into the Z88 you can wake it up again at regular intervals to ask you a series of questions. This might be used to allow you to record your current project number so that your time sheet could be filled out automatically. Use of the RND function at this point poses some quite amazing possibilities. But take care here because the Z88 has a built in feature to prevent accidental operation of the keyboard if the Z88 is woken up by the alarm when it is in your suitcase at 10 000 metres above the Arctic. You must switch the Z88 off and then on again after the alarm is raised to allow access to the keyboard. The alarm should be set to execute Infomate when it goes off. The procedures for this are laid down in the Z88 manual, and are also described later under the Actronic ADAM driver module. In this example the alarm feature is used to operate Infomate and the ADAM as a data logger.

THIS IS WHERE IT ALL COUNTS

Counting objects

Counting objects that present themselves to the observer in a random order is an extremely common research task that manifests itself over a broad range of scientific endeavour. The task presented here is perhaps typical; to observe a specimen under a microscope and to count cells of four different types. The observer works systematically through a grid pattern displayed onto the viewing area and wishes to count each cell type as it is found and identified. The cell types are defined as Flat, Dished, Elongate and Cuboid and it is expected that up to 20 cells of each type could be present in each sample. The data field definitions for this part of the problem would appear as

```
DATA
          .
          Flat          N2
          Dished        N2
          Elongate      N2
          Cuboid        N2
          .
          etc.
```

The grouping of these fields adjacent to each other is important, although they do not in fact have to have the same size. In the sequence region the data collection statement could be:

```
COUNTTYPE      Flat     FDEC
```

COUNTTYPE means we are going to count the quantities in the 4 fields (because there are 4 letters in the right hand string) starting with the field Flat. Each field will be incremented when the operator presses one of the single keys of the set FDEC. Thus to increment Flats just press an F, press it again and it will increment again. To decrement the field press the DEL key and then the letter of the field to be decremented. When all counting has been completed press the ENTER key and sequence processing will continue. There are a few limitations to the usage of this statement. Obviously it is not possible to increment letter fields or in fact any of the ID fields, and the definition should not attempt to increment fields that are past the last data or scratchpad field (whichever is last).

The letters do not in fact have to be mnemonics for the fields (i.e. F for Flat etc.) but might be labelled keys on the keyboard, or keys in particular positions on the keyboard. For counting while observing under a microscope you might be best to use the keys QWER or the digits 1234 because of their more easily located position on the keyboard. The operator would then just remember which finger to tweak to bump each field.

Transects

The COUNTTYPE statement can also be used for indicating the presence of objects rather than counting them. Thus, if the data fields were species of plants you could indicate that a species was present in the observation plot by pressing the appropriate letter. This has obvious application in the collection of transect data, where there may be a number of commonly occurring species at each of a number of sites located along a transect line. Less common species are probably best left to a general comment type field at the end of the main species fields.

Roll Calls

The transect idea in the last section is rather like a roll call. Each type of object present has its letter pressed and gets marked in the roll of fields. This scheme may have application in animal behaviour studies where the animals observed at a feeding trough at particular times are to be recorded. In this case it will be necessary to assign a letter to each animal if there are more than 10. Some form of keyboard overlay might be appropriate for this.

Histograms

In a way it is possible to think of the COUNTTYPE statement above as a histogram forming statement which works on various categories or types of objects. There is a closely related type of histogram that operates on sizes of a group of objects. An example might be a fruit sizing experiment where fruit weights are entered through the keyboard and a histogram of weights is formed in a number of fields. For kiwifruit trials we might arrange for 6 weight ranges in the following way.

```
DATA
        .
        .
        UNDERS        N3
        W65-84        N3
        W85-104       N3
        W105-124      N3
        W125-144      N3
        OVERS         N3
        .
        etc.
```

Because the data is collected on many fruit the sequence must use a REPEAT UNTIL KEYPRESS loop in the following way.

```
REPEAT      Sizing
            ENTER         MEMORY    Fruit  weight
            COUNTSIZE     UNDERS    6    45.0    20.0
UNTIL       KEYPRESS
```

This would loop continuously until the ENTER key was pressed twice rapidly. Each time round the loop, the weight entered into the MEMORY "field" would be divided down to see which category it belonged to on the basis that the first category is 45.0 to 45.0+20.0 , the second would be 45.0+20.0 to 45.0+2*20.0 etc. Any weights below 45.0 will be counted in the first category, and weights over 165 will be counted in the last category. Obviously this will build up a histogram of the fruit weights in the group being collected. In practice the ENTER statement might be replaced by a statement to fetch weights directly from electronic scales or lengths directly from an electronic caliper.

A variation on this would be to add a counter to indicate how many fruit had been processed. The following schema fragment would do this automatically from scales

```
REPEAT   Sizing
         ENTER?<+          Count     Fruit  Number
         GETPORT           MEMORY    Fruit  weight
         COUNTSIZE         UNDERS    6    45.0    20.0
UNTIL    KEYPRESS
```

On the first run the fruit count would have to be set to 1 and the sequence would pause showing the fruit number on each cycle. When the fruit was on the scales the operator would press ENTER and weighing would proceed. This would allow a specific number of fruit to be sized.

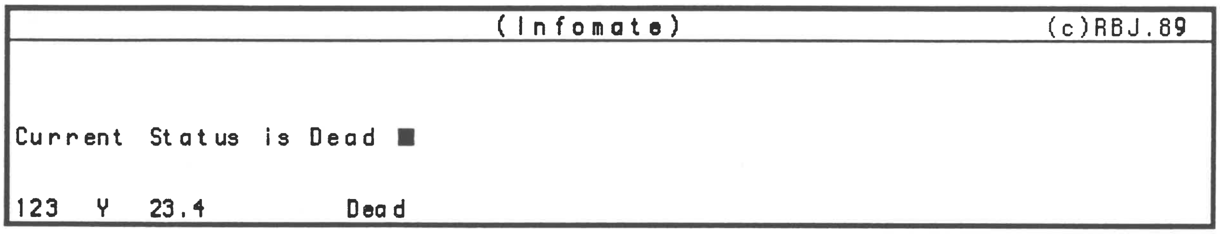
SOME MISCELLANEOUS TOPICS

Showing off your fields

Occasionally there is a need to display the value in a field. This could be the value that has been placed in the file from a keyboard entry, a value from a device connected to the Z88, or a calculated value. This can be performed using a DISPLAY statement. The DISPLAY statement might look like this

```
DISPLAY     Status      Current  status  is
```

Whenever this statement was executed in a sequence the screen would show something like



```
                              (Infomate)                          (c)RBJ.89


Current  Status  is Dead ■

123  Y  23.4        Dead
```

Processing would continue as soon as a key was pressed. The DISPLAY statement is often useful to wait for scales to be loaded. One of the following two forms may be tried.

```
DISPLAY      ID       Hit  Enter  to  weigh
```

OR

```
DISPLAY      #        Hit  Space  when  ready
```

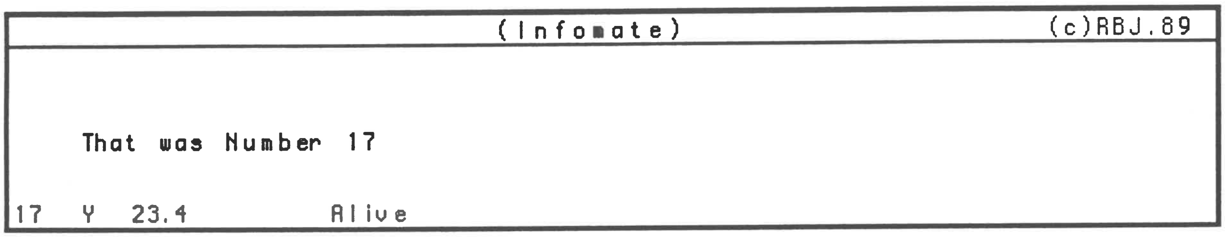
Note that the last of these has a hash character (#) instead of a field name. This will just display the message without a field value, and should be thought of as the word "...message...".

Adding Bells and Whistles

The DISPLAY, GETPORT and ENTER statements all show messages on the screen. To these messages you can add beeps and pauses using the characters "!" and "~" respectively. These should be added at the end of the message like this.

```
DISPLAY COUNT   That was Number!!~~~
```

which would beep twice (two !s) and show the message



```
                         (Infomate)                          (c)RBJ.89


    That was Number 17


17   Y   23.4          Alive
```

Then it would pause for three seconds (three ~'s) and continue without the need for a key to be pressed. If you did in fact press a key, the three second delay would be immediately terminated.

Now you might like to put beeps (!) in all of your ENTER statements so the operator is warned that data entry is required, and put pauses (~) at the end of all of your GETPORT statements so that the data captured from the scales stays on the screen for a while after capture for you to inspect.

Decimals by implication

And for those who like life in the fast lane try this. It seems silly to put decimal point in every number in a file when we know exactly where they will be anyway. This might make a difference to the size of a big file. For example if we recorded 500 animals with three digit ID's and an N4.1 weight (e.g. 23.2) then 12.5% of the file would be decimal points. We could put another 62 animals in that space. For those conservative people who like to save space there is an implied decimal format that eliminates the decimal. An N4.1 field with the point removed is described as an N31 field. i.e. 3 digits with one **implied** decimal point. N fields that do not have a point in their format are implied decimal fields. When the data is typed into an implied field you must type in the decimal, but it will not be stored. For users without space problems forget implied decimals.

Just an aside on implied decimals. If you are using an ENTER¦ statement you will need to type one character more than the field width to enter the data without hitting a return. An N31 field compresses numbers like 23.7 down to 237 . However as you must type 23.7 you need 4 character spaces and not 3 as inferred in the N31 specification. If you do not want to type the decimal point then use N3 fields (no decimals) rather than an N31 , and type 237 to mean 23.7 .

Taking notes on the fly

The Apple II Recording System used a concept called an "Exceptions File". This was a catch all file into which the operator could put miscellaneous comments and notes to themselves about problems or observations made during the data collection. It was also possible in that package to push data records into the exceptions file at the press of a key as it were. Infomate does not have an inbuilt exceptions file but it is possible to implement all of the functions of that file if a few more key strokes are acceptable. Of course the way to do this is to use Pipedream. Once the schema file has been compiled there is no further need to have Pipedream and the schema file sitting waiting. Pipedream might as well be performing a more useful role. So make it an exceptions file editor.

To do this either open a new Pipedream from the left hand column of the Index, or go into an existing Pipedream whose contents you no longer need (i.e. have saved), and press the key sequence ◇BNEW . This will clear the existing contents.

For a name for this exceptions file I suggest using an extension of .XEP which means exceptions. It makes sense for the first part of the file name to match that of the data file that it is commenting on. This means for a data file collecting fruit weights you might have the file set

```
FRUITWT.DAT
FRUITWT.SCH
FRUITWT.XEP
```

for the data, schema, and exceptions files. Now specify the current filename to Pipedream using ◇FC so that your files name is ready for the next time that you save the file.

Whenever you want to add a comment into the file just press □P and type it. At frequent intervals save the Pipedream file to RAMdisk using ◇FS ENTER , and you may wish to save that to EPROM occasionally too. To return to Infomate use □ZI as usual. This approach also allows you to edit previous comments, or arrange comments in different orders, or even in different sections for easing later interpretation, or for giving to different people. The options are endless.

One point to remember here is that by saving the files in Pipedream format, they will have a few extra lines added to them when they are read into another computer. If this causes you concern then each time you save the files, or failing this, on the final time that you save the file, move the cursor down in the "save file" box and change the assignment for "Save plain text" to a **Y** .

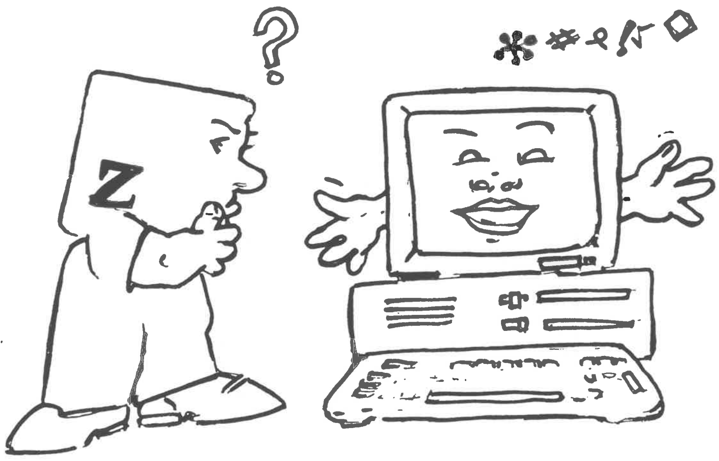
As an alternative to this it might be possible for you to leave a few dummy ID's in the file specifically for the odd animal that comes along later. This means that you can collect all of the data for that animal using the normal sequence, and just put a brief note into the Pipedream file to say which animal it is, or even place this information into a comment field in the file itself. The options are endless.

Using Pipedream to Inspect Data

Pipedream can be used to inspect data in the standard Infomate format but care should be taken. If the data is edited in Pipedream then the operator must ensure that all data records end up with the same record length as that specified in the corresponding Infomate schema file. This can be done by moving to each line that could have been affected by an edit and pressing ◇⇨ to ensure that the last character in the line is in the correct position relative to lines before and after it. Thus repeated application of the sequence ⇩◇⇨ and observation of the horizontal movement of the cursor will be an adequate check.

Also ensure that there are no blank lines at the end of the file before saving it. The character sequence ◇⇩ should move the cursor to the end of the last record in the file and not to the blank line after that. Finally, save the file using the "save plain text" option at the end of the file save page.

FILE TRANSFER TO OTHER COMPUTERS

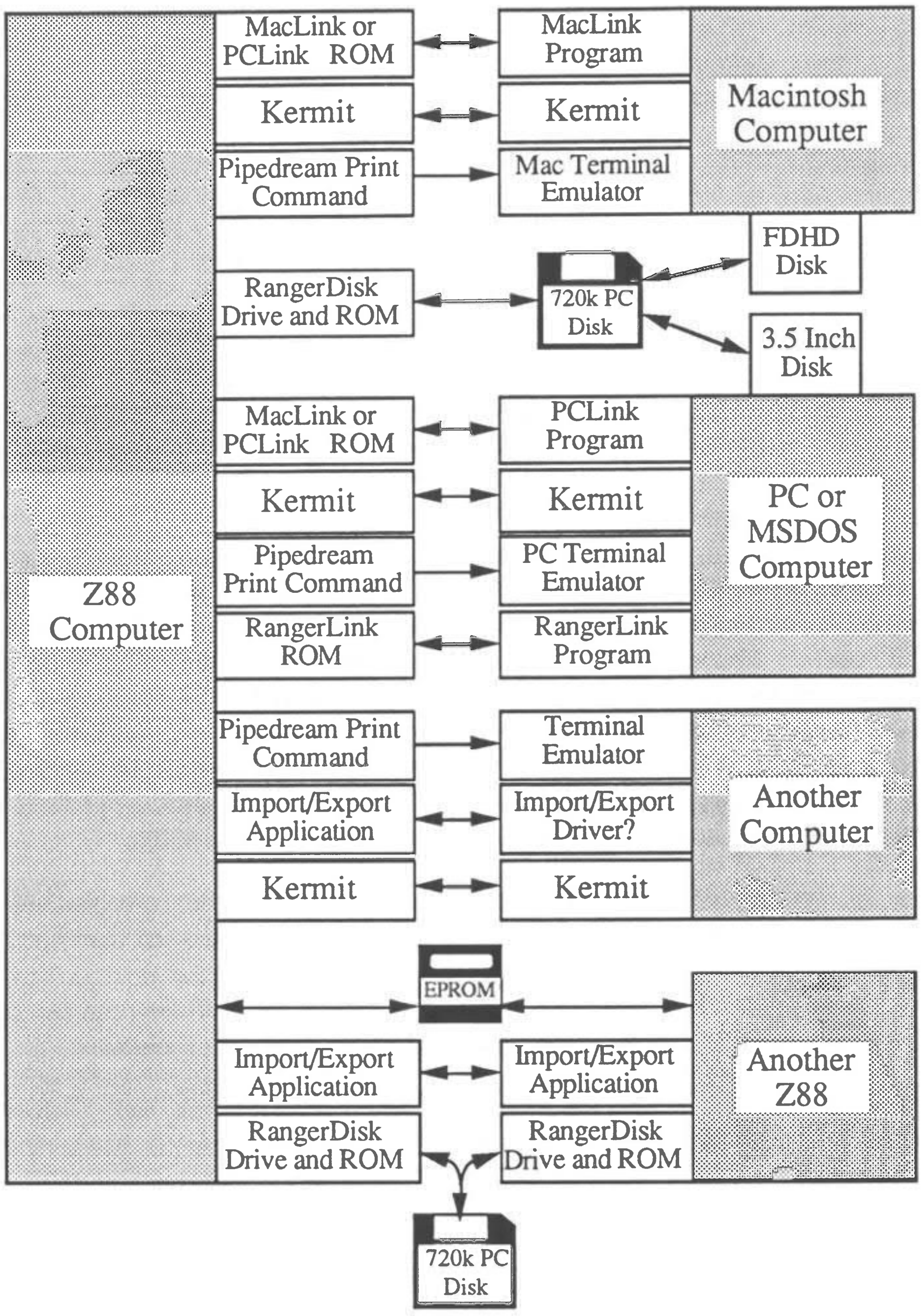


The Infomate developers do not see the Z88 as an appropriate medium for handling data analysis, and to that extent have not provided any assistance in this area at all. We suggest that all of this type of work is handled on a PC, a Macintosh, or on a VAX or larger computer. These machines all have sophisticated data processing packages that must always exceed the possibilities offered by the Z88. Even a fairly ordinary spreadsheet on one of these computers is capable of doing a lot of data manipulation, and more sophisticated packages like DataDesk on the Macintosh can perform quite marvellous data analysis. To this end **all** copies of Infomate are bundled under licence with a copy of a program called variously PCLink, or MacLink. Both of these are in fact the same package on the Z88 but have different programs supplied for the computer to which they are connected. In addition to this Link program there is a more sophisticated PC linkage program called RangerLink. This can be bundled with Infomate under licence as well for a small additional cost. We recommend this Link package for PC users as it offers higher data transfer rate, and much tidier backup of groups of files.

On top of these the standard Z88 contains two low level communication packages that might be useful. These are a VT52 terminal emulator to provide simple terminal facilities, and a basic file transfer protocol called Import/Export which is extremely good for moving files between Z88s. Provided a simple driver is obtained for another computer, Import/Export can be used to transfer to and from any computer. In fact drivers exist now for many common computers.

Lets now look at the individual file transfer packages and how to use them. The following picture provides a summary of these packages and indicates which ones can be used for each combination of computers.

Some Possible Links Between the Z88 and Other Computers



Maclink and the Macintosh

The MacLink program on the Macintosh was written after the MSDOS version PCLink and the higher level of user friendliness that has resulted from this is quite striking. Once the cable has been connected between the Z88 and the "Telephone" port of the Macintosh, and the Link program has been started up on both, all operations are then performed on the Macintosh. The Link program is started by double clicking the Macintosh Application and by typing □ L on the Z88. The Z88 becomes a "disk drive" for the Macintosh and may be selected using the DRIVE button on the Macintosh screen. All file transfers are performed from left to right so the first task is to open up the various "Folders" on the Macintosh, and the Devices and directories on the Z88 so as to have the file being moved in the left column and the place it is going to on the right. Next the operator should select the file conversion to be performed. This will normally be "None" unless the document is a spreadsheet document or a normal word processing document in which case the appropriate conversion can be selected. Next select the files in the left hand box that are to be moved using the **mouse click** and perhaps the **command or SHIFT key** to select more than one. When all are selected click the transfer **box** and file transfer and conversion if required will proceed. In practice its simpler than it appears here.

The Link package is not fast (about 100 bytes per second) so have your coffee ready for larger files.

The cables required to link the Z88 to a Macintosh are provided with MacLink or may be built up using the circuit details at the end of the manual.

PClink and the PC

PClink is a fairly intuitive package to operate on the PC. Apart from its speed, it has a few minor difficulties. It will only handle one file at a time, conversions to other file formats must be handled as separate operations, and the conversions must be performed in the directory that contains PCLink and the conversion modules. If these points are adhered to there should be no problems.

To operate the link connect the Z88 to the PC's COM.1 serial part and run PCLink on the Z88 using □ L. All further operations are now handled on the PC. Move to the directory on the PC containing PCLink and enter the command PCLINK. Communications will be established and a display will appear on the PC showing the two computers in diagrammatic form.

To prevent conversion difficulties it is suggested that files that need to be converted be moved to the PCLINK directory before conversion takes place. It is often best to work all transfers to and from this directory. If you perform conversions in another directory the resulting converted files will not contain any data.

Printing to a Terminal Emulator

One quite tidy method of transferring files to another computer is to use Pipedreams Print command (◇ PO) to send the file out through the serial port, and to set up a terminal emulator on that other computer to accept the file as though you had typed it. Some terminal emulators allow the stream of characters to be captured directly to a file on the main computer making this task even easier. Obviously both computers must be set up to use the same baud-rate and other serial port parameters. The only difficulty with this approach is that Pipedream inserts extra characters at the end of each physical page as though it were printing (which in fact it is). These characters may be eliminated by setting all Pipedream margin settings in the ◇ O (options) page to zero.

Note that you can use the Z88's internal VT52 emulator to talk to the host computer and prepare it to receive the characters you are to send.

One Emulator we have used extensively on the Macintosh is VersaTerm Pro.

Unfortunately a method to achieve transfer in the reverse direction has not been found, but does seem possible. The Z88 VT52 emulator might be usable for this, but is probably too basic in it's operation.

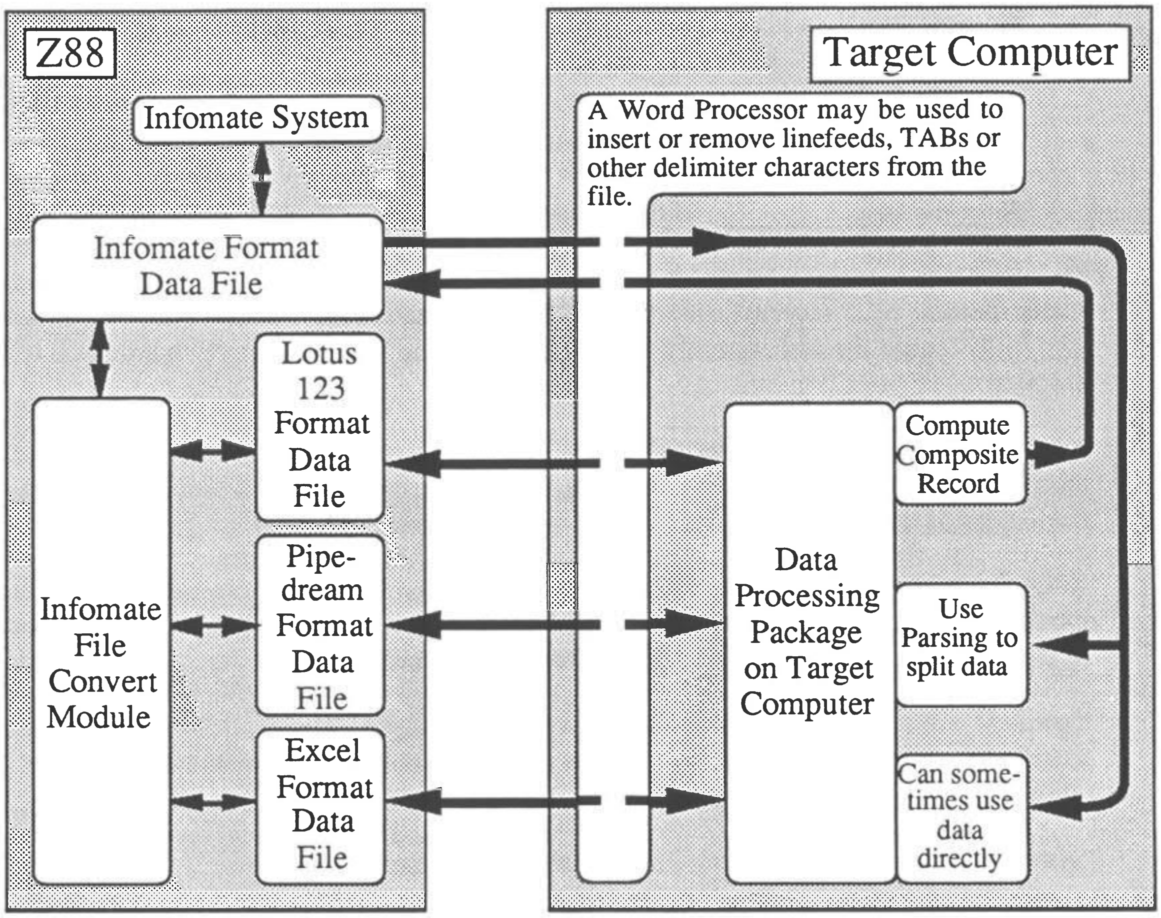
Kermit

The Z88 User Club Program Library holds a very tidy implementation of Kermit for the Z88 which operates in the BBC BASIC environment and uses about 15k of memory. Kermit is available for every important computer known (?) and thus provides a very tidy transfer tool for the more difficult computers. Unfortunately the Z88 Kermit does not currently offer a Server 'Bye' command which limits it's usefulness in the mainframe computer environment.

FILE CONVERSION

There are almost as many data file formats as there are data manipulation packages, and all have their own subtle differences to confuse and to anger the data collector. All have their own particular reasons for choosing their particular formats, and most are perfectly valid. This means that in general you will have to convert the file collected by Infomate before you can use it on another computer. Infomate provides some internal means of doing this for a selected formats, and techniques are provided later in this chapter to describe techniques using the target computer as a file converter.

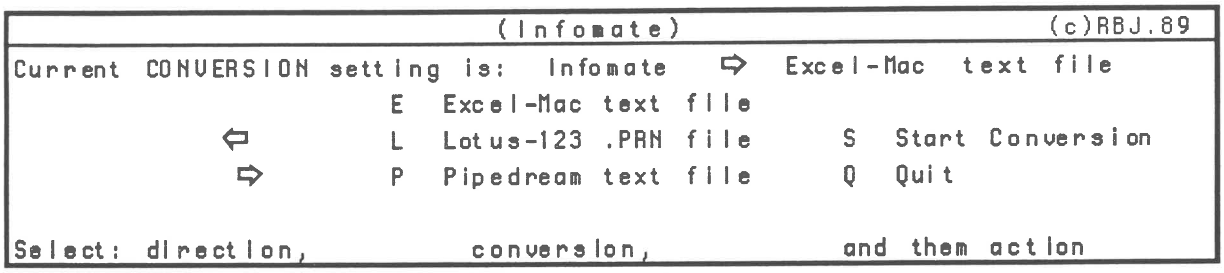
In summary the picture below shows some of the possible paths from the Infomate format file to a form suitable to the target package.


Z88
Infomate System
Infomate Format Data File
Lotus 123 Format Data File
Pipe-dream Format Data File
Excel Format Data File
Infomate File Convert Module
Target Computer
A Word Processor may be used to insert or remove linefeeds, TABs or other delimiter characters from the file.
Data Processing Package on Target Computer
Compute Composite Record
Use Parsing to split data
Can some-times use data directly

The File Conversion Module

The main Infomate menu provides a basic conversion module to convert Infomate data files to and from three file formats that will be useful in a range of applications. This program is not particularly fast but should handle all of the possible scenarios for file conversion if Lotus 123 (or its lookalikes), Microsoft Excel (and most Macintosh spreadsheets) or Pipedream on the Z88 are the targets. In many cases these formats will work on other computers and packages as well. Details of these formats, including that for Infomate are provided in the reference section of the manual.

To use file conversions you must first compile the schema file which describes the format of the data file to be converted. Then select "File Conversion" from the main Infomate Menu. The conversion module shows a screen like this



```
                              (Infomate)                              (c)RBJ.89
Current CONVERSION setting is:   Infomate   ⇨   Excel-Mac  text  file
                         E   Excel-Mac text file
           ⇦             L   Lotus-123 .PRN file       S   Start Conversion
            ⇨            P   Pipedream text file       Q   Quit

Select: direction,            conversion,              and then action
```

The aim of the exercise is to choose an arrow direction and a file type to make the top line of the screen indicate the appropriate conversion. The one indicated above shows a conversion set-up to take an **Infomate file** and convert it **to** (**⇨**) **Excel** for the **Mac**. When the appropriate selection has been made, press S to Start conversion.

You will next be asked to enter the Infomate file name. The default for this will be the file named in the schema file and users should note that if a different file is entered, it is assumed to have the same field format as that described in the currently compiled schema.

When the Infomate file has been selected you will be asked to enter the file name of the alternative format file. If you are converting to Infomate form this file must already exist in :RAM.1, but it will be created if you are converting an Infomate file to a new format.

Operation will start once both files names have been specified. If you use alternative names to that specified in the schema file you will receive a warning to say that formats are assumed to be the same as that in the schema, or that files will need renaming before use.

To exit from the file conversion module use the Quit command.

Pipedream and Infomate

The Pipedream files created by the file conversion module should be loaded into Pipedream "as plain text" and when saved from Pipedream for conversion to Infomate form should also use "plain text" format.

Note that although data files must be saved in 'plain text' form when using Pipedream, this is not necessary with schema files because Infomate has been 'trained' to recognise all of the extraneous characters in Pipedream files and ignore them.

Microsoft Excel and Infomate

An Excel text file from the conversion module can be loaded into Excel as though it were an Excel file. Ensure that the TEXT column delimiter is set to TAB when you open the file. This is the default for both PC and Macintosh versions. When saving from Excel for Infomate usage perform a "Save As" using the "options" "text" form for the data.

PC Lotus 123, VP Planner, PC Quattro and Infomate

Files converted from Infomate into Lotus 123 form can be loaded into Lotus using the "/FILE IMPORT Numbers" command. In Quattro use /TOOLS IMPORT COMMA&""DELIMITED. They can be created from Lotus 123 for conversion to Infomate format using the "/PRINT FILE" option. In this form they appear as print files with the appropriate number of blanks inserted between fields to make columns line up. Infomate's file conversion module scans these files to work out where the field contents lie and then extracts the minimum field space from the data. Data is assumed to be right justified in the field. Similar operations work correctly for VP Planner and Quattro as well.

Note that 'missing' values are created by Infomate as a single space character surrounded by double quotes, and that this is not a true missing value flag for any of the above packages. If this creates a problem you may need to convert these 'missing' values to empty cells using a suitable macro. Alternatively, you could import the Infomate file directly and perform the conversion on the target computer using the technique discussed later in this chapter.

PC Minitab and Infomate

The following may serve as an example of how to use the Infomate file conversion package to create and use files for a package that is not directly supported by the Infomate Menu. Minitab is a popular data analysis package that has appeal with researchers due to its simple operating concept but powerful functionality. A workable system to create Infomate files using Minitab, transfer them to the Z88 and use Infomate for collecting data, and finally returning the results to Minitab has been worked out by an Infomate user. The ID numbers and replicate data were prepared on Minitab, complete with a number of empty columns that were to receive data.

The procedure is as follows:

1. Use the Minitab 'Write' command with 'Format' sub-command to create the data file with specified field widths for each data column. You can use 'X' format to put the current number of spaces in the empty columns, or alternatively create data columns full of zeros.
2. The 'Set' command using abbreviations for patterned data is useful for creating columns with sequences of plot numbers etc.
3. Transfer file to Z88 with PCLink II or RangerLink.
4. Load the file into Pipedream as plain text and immediately save the file again (as plain text too). This deletes all of the linefeed characters.
5. Collect the data using Infomate.
6. Convert the resulting file to Lotus.PRN form using the file conversion in Infomate and transfer it to the PC.
7. Use the Minitab 'Read' command to load the .PRN file into Minitab.

PC Paradox and Infomate

Paradox can accept an Infomate file using the following procedure.

1. Convert the file to Lotus 123 form using the file conversion module in Infomate.
2. Transfer it to the PC and use a word processor to replace all of the TABs with comma characters.
3. In paradox, use the command sequence Tools/ExportImport/Import/Ascii/Delimited.

The above brings in the data perfectly, and unlike Lotus 123 handles 'missing' values correctly.

To transfer data from Paradox to Infomate the general procedure would be:

1. Create a paradox report on the table that you want to export data from. The report should be a tabular report.
2. All text fields placed on the report should be calculated fields. (Use the Paradox FIELD PLACE CALCULATED command from the report generator menu). This will allow you to use the Paradox FORMAT function to force fields to be right alligned (the 'ar' option). You also need to specify the field width when left alligning text.

   For instance to place the text field called Tag which is a maximum of 8 characters long, the expression for the calculated field would be FORMAT("w8,ar",[Tag]) .
3. Place the fields that you want to export right next to one another in the table band.
4. Remove all other blank lines and text from the report specification.
5. Set the page length to continuous with the SETTINGS PAGELAYOUT LENGTH command from the report generator menu.
6. Print the report to a file.

Converting from Infomate form on the Target Computer

Techniques exist to perform the required file conversions on the target computer as an alternative to using the Infomate File conversion module. These work well in certain circumstances, but care should be taken, particularly with Implied Decimals. There are generally two problems to solve when using a PC, splitting up the fields and adding a linefeed character. Linefeeds tend not to be a problem on the Macintosh.

The dreaded Linefeed Problem - PC's

The standard on a PC for an end of line mark is the combination Carriage return - Linefeed. Infomate, Pipedream, and most Macintosh programs use a Carriage return alone. The effect of this is that an Infomate or Pipedream file that has not been correctly converted when transferred to the PC will appear as a single line disappearing off the right of the screen.

To fix this problem you will need to perform a global change of the carriage return character to the combination carriage return - linefeed. Most Word processors will allow this to be performed easily.

On PCWrite to make the change use the sequence F9 Alt-1-3 (i.e. press F9 then hold down Alt,type 1 then 3 on the numeric key pad, then release Alt). Now press F10 Alt -1-3 then Alt-1-0. To repeat this change for the entire document press Alt-F10 F9 and the text will appear as it should.

In the reverse direction you can use Pipedream to remove linefeed characters from a file. It seems that if linefeeds exist in a file that is read by Pipedream, they will be ignored. So read the file into Pipedream, and then immediately save it again. For both the load, and the save operation use the 'plain text' option. The linefeeds will magically disappear. Do not change any of the data with Pipedream during this load and save sequence.

Creating an Infomate file in Excel on the Macintosh

If you start with a file in Microsoft Excel that you wish to transfer to Infomate without going through the conversion module then the following procedure can be used. This technique applies equally well with subtle variations to other spread sheets.

The procedure is to create a new column of data that is a packed text version of the data from the columns of interest. This involves creating a formula that converts numbers to text, and adds the appropriate number of spaces or zeros to the left.

As an example consider the Infomate format

```
ID                                          Stored in Columns
        Row       N2                                  A
        Type      L1                                  B
DATA
        Wt        N4.1                                C
        Count     N3                                  D
        Colour    L3                                  E
```

We will make all numbers contain leading zeros and all letter fields have leading blanks. In the case above, all entries in the Type field are known to have exactly 1 letter. The Excel Formula to use to create a new column F will be

```
=TEXT(A1,"00")&B1&TEXT(C1,"00.0")&TEXT(D1,"000")&RIGHT("         "&E1,3)
```

This shows all Fields linked together using the text form of the add operator which is &. Numbers are converted to text using the function TEXT(reference, "format") where reference refers to the cell of the spreadsheet to be converted, and format is a pattern for the storage made up of zeros and decimal points.

Fields that are already text can be added directly as is the case for B1 above, provided they are guaranteed to be the correct length. Alternately use the construct RIGHT("   "&E1,3) to ensure that they are right justified and are of the length required.

The formula above should then be propagated down the column so that the packed form of the data is created over the length of the column. Now select the column over the length of the data and copy it using "Edit" "Copy". Open a new spreadsheet using "File" "New" and paste it into the top left corner using "Edit" "Paste special". This new spreadsheet can then be saved using "Save As" "Options" "Text" and when transferred to the Z88 should be Infomate compatible if you did everything correctly.

The above procedures could be automated using Excel Macros if desired.

Parsing - Microsoft Excel 2.2 on the Macintosh

Providing that your data file does not appear as a single column of multi-digit numbers, then when it is read directly into Microsoft Excel it will all be placed as text into column 1 of the spreadsheet. If Excel does not detect any non numeric data in the file it will convert it into a number using its own internal number representation. So if your data looks just like a big number you may have to fool Excel into thinking that it is not a number. More on this later. When you have it loaded into Excel as text, it can easily be split up into columns using the Parsing Command. First Select the Column to be split up by clicking the column letter at the top and then select "Data" "Parsing". The menu presented will show you the first set of fields in the column and will insert square brackets where it thinks the columns are if you click "Guess". You should now edit the guesses to match up with your data positions by moving or inserting the square brackets. e.g.. [12][12.34][ABC] will split the data into columns containing 2, 5 and 3 characters respectively. Clicking "OK" will complete the task.

Now, if Excel manages to read some of your data as pure numbers this operation will not work. You can guarantee that it will work by (a) including alphabetic fields in your records, (b) by guaranteeing at least one column of space characters by making a field wider than it needs to be, or (c) by ensuring at least two fields have a decimal point recorded.

There are Parsing techniques in all recent versions of Excel, Lotus 123 and other spreadsheets on both PC's and Macintosh's. PC Excel does not require LF's to be in the file, and will accept the carriage return as a terminator character.

Column Inserting - Microsoft Word 4.0 on the Macintosh

Recent versions of Microsoft Word on the Macintosh allow operations on columns of characters. These are selecting using the mouse while the option key is pressed. This feature can be used to insert columns of TAB characters or other delimiters into a file, and thus convert an Infomate file into a normal TAB (or other character) delimited file that can be used by the target processor. The technique described here is a manual one but could be automated using a macro once the details are worked out.

If you load your file into Word 4.0 and change the entire file to Monaco 12 pt (or some similar fixed character width font) all characters will be in vertical columns above each other.

Now change all of the paragraph marks (carriage returns) to TAB (or whichever delimiter you wish to use) plus carriage return using the "Utility" "Change" command. Note that you should change ^p (which is a paragraph mark) to ^t^p (a TAB followed by a paragraph mark). You may then need to insert a TAB at the end of the last line of your file. Hold the option key down to set column mode and drag the cursor to select all of the TAB keys. Cut these using Command X.

To insert a column of delimiters, just put the cursor in the first record of the data at the appropriate position and perform a paste (Command V), which will insert a column of delimiter characters into that position right down through your data. This should be repeated for each delimiter position.

Two important points with this. First, make sure that you insert the final delimiter before the last paragraph mark before you cut the column of TABS, and secondly save the data using "Save As" "file format" "text only".

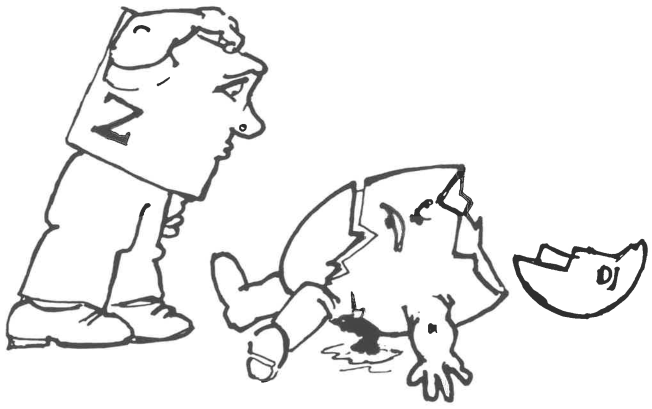
Column Inserting - PCWrite 3.0 and Word 5.0 on the PC

The column inserting technique described in the last section is also possible on the PC and follows exactly the same path as that used for the Macintosh. Read the above section first if you are to use this technique. On the PC you will generally not need to change fonts as you do on the Macintosh as the characters are fixed in width by definition. For Word 5.0, column selection is performed using Shift F6. Move to the top of the delimiter column, press Shift F6, move to the bottom and perform the cut. PC Write uses a similar construct but with Ctl SHIFT F6. Consult the manuals, for more details.

WHEN THINGS GO WRONG

Things you did wrong

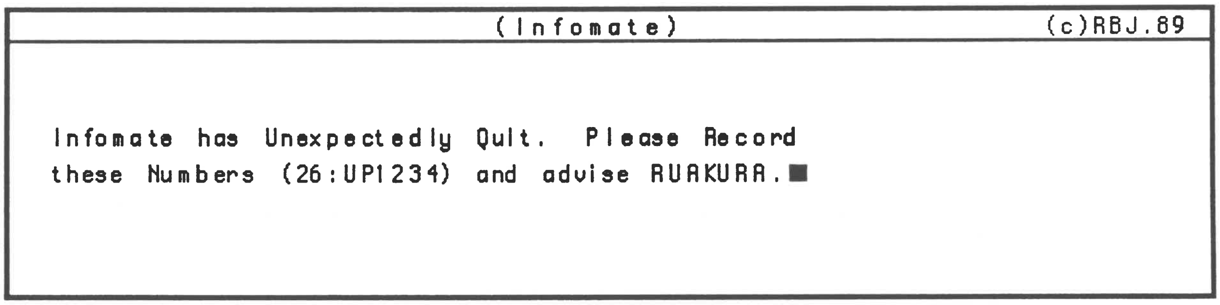
Obviously such a powerful package as Infomate will offer you lots of opportunities to do things wrong, particularly when you are learning to use its power. Misspelled field names, letters in number fields, or using lower-case letters rather than capitals are but a few examples.



At the end of this manual is a list of the error messages that you could get while running Infomate, and their possible causes.

Things we did wrong

There may still be bugs in Infomate that we have not found so if you find one please let us know. Some are horrible ones that say something like



```
                    (Infomate)                         (c)RBJ.89

Infomate has Unexpectedly Quit.  Please Record
these Numbers (26:UP1234) and advise RUAKURA.■
```

If that happens please do advise us of the numbers, and what particular sequence of events caused them.

Lets blame somebody else

There are a few problems that are due to difficulties in the Z88 and its operating system. All of these that we already know about that affect Infomate directly have been circumvented. However there are some bugs in the Z88 that you must avoid.

1. **Never** leave a file in the device :RAM.- . If a RESET or ◇ PURGE occurs when there is a file in :RAM.- the operating system will "become confused" and loss of data is a possibility.
2. There are difficulties with the PrinterEd popdown which will cause the Z88 to lock up. Take care when using this that you have your files backed up to EPROM or to another computer.
3. Sometimes EPROM modules do not make complete connections with all of the address lines in the slot and their data will appear to be scrambled. Before and after saving data to EPROM always check that your current files appear OK using the EPROM catalogue command. If difficulty occurs, use another EPROM to save your data.

Running out of memory

If you get the dreaded No Room message at any stage, then the Z88 has not got enough space to run the task that you have requested, and you might have to try the following things to solve your problem.

1. Remove any applications from the pending application list of the Index that you do not need by putting the cursor onto the offenders and using the ◇KILL command. Further details are provided in the Z88 manuals.
2. If the problem occurred when Infomate is a pending application then make sure that you exit from Infomate using the Infomate "Pack and Exit" command. This command will squeeze the Infomate package down as much as possible before moving to the Index. You could save 30k of the available memory by doing this.
3. Move all unwanted files from the Z88 to another computer.
4. If you have files existing on the Z88 in more than one format (e.g. Infomate format and Excel text format then you can probably delete one version safely.
5. Buy more memory. It is possible to get RAM memory modules up to 1 Megabyte in size now and it comes in convenient hunks down to 128k bytes. If the problem you are having is basically a storage requirement problem rather than a storage wastage problem then this may be your only solution.

Enhancements and fixes

There were a number of enhancements proposed by users of Infomate during testing of version 1, and many of these have been incorporated into version 2. We encourage users to send details of the things that they would like to see in Infomate, however crazy or difficult they may seem to be. If there is sufficient merit in the ideas, and, more importantly, if we can find time to make the changes, we will do so. This is particularly true of bugs that you find. As problems are located we will fix them, but will probably release "bugfix" versions of the software less often so don't expect to visit us and walk away with a fixed up version of Infomate. So keep putting those ideas and problems on paper.

Infomate Size Limitations

Currently the following limitations apply to Infomate

Maximum number of fields	40
Maximum number of schema lines	150
Maximum number of sequence steps	100
Maximum number of records in a file	2000
Maximum characters in a record	85
Maximum constants in a schema	40
Maximum Record length + Number of fields	92
Maximum (ID letters*3) + (ID numbers *2)	18

Note: The last two entries are limitations on combinations of parameters. i.e. the total number of characters in your record, plus the number of fields must add to less than or equal to 92. Similarly, the number of ID letter-type characters times 3, added to the number of number type characters times 2 must not exceed 18.

EXAMPLE SCHEMA FILES

In the following few pages there is a collection of schema files as created for real tasks by users with specific problems. These are available on PC or Macintosh disks. If you have schema files that you would like to place in our library, send them in on floppy disk complete with a description of what they do and we will endeavour to maintain the library.

Flip Flop Schema

In the following schema file two people are weighing herbage samples collected by one mower but using two grass catchers. While one catcher is being weighed, the other is being filled. Now that would be easy if both catchers always had the same tare value. Alas that is not the case.

The following schema uses MEMORY to remember which catcher is to be weighed and uses the corresponding tare value. It then sets MEMORY so that next time the other tare is used. You may have to try this to see how it works, and make some paper models of the catchers to remind you of the sequence that is occurring.

```
TITLE     Flip  and  Flop  Weigh  Grass
FILE      TOGGLE.DAT
LOG       LOG
ID
          Sample_No         N3
DATA
          Weight            N4.1
SCRATCHPAD
          Raw_Weight        N4.1
          Tare1             N4.1
          Tare2             N4.1
SEQUENCE
          ENTER    Sample_No         Sample  Number  ?
          MAKERECORD
LABEL                                \First  time  sequence
          ENTER=<  Tare1   Tare1  is
          ENTER=<  Tare2   Tare2  is
          TEST     MEMORY  >  1.5
          IFTGOTO  LABEL   Flop
LABEL     Flip                       \Catcher  1  sequence
          ENTER    Raw_Weight        Weigh  Sample(C1)
          LET      Weight  =         Raw_Weight  -  Tare1
          ENTER?<  Tare1             New  Tare1  is
```

```
                LET      MEMORY  =         2
                GOTO     END
      LABEL     Flop                       \Catcher 2 sequence
                ENTER    Raw_Weight        Weigh Sample(C2)
                LET      Weight  =         Raw_Weight - Tare2
                ENTER?<  Tare2             New Tare2 is
                LET      MEMORY  =         1
      END
```

Mountain Bike Race System using Infomate and Pipedream.

The following set of schemas was set up by an Infomate user to collect the data for, and monitor the proceedings of, a mountain bike race meeting. It has been used successfully on a reasonably large meeting, but probably requires reasonably expert Infomate and Pipedream users to solve any problems that might occur during the meeting.

The first schema MTBENTRY.SCH is used to enter rider numbers and class of registration. Experts ride three laps and are issued with numbers from 1 to 99, while sports riders do two laps and take numbers from 101.

```
      TITLE     Mountain Bike Race Entry Listings 1Sept1991
      FILE      MTBSEP1.DAT
      ID
                Rider            N3
      DATA
                Class            N1
                Lap_No           N1
                Lap1time         N5
                Lap2time         N5
                Lap3time         N5
                Time             N5
      SEQUENCE
                ENTER    Rider   Rider's Number?
                MAKERECORD
                ENTER    Class   Class?(1-7)
      END
```

MTBTIMER.SCH is used at the race run time as the stop-watch. The first 'Rider' (number 0) is used to capture the race start time (this data would be best kept in a SCRATCHPAD variable), at the start gun. As each rider completes a lap, their competition number is entered. At the end of the race this data is saved to EPROM so that it can be retrieved if problems occur.

```
TITLE    Mountain Bike Race Lap Timing 1 September 1991
FILE     MTBSEP1.DAT
ID
         Rider           N3
DATA
         Class           N1
         Lap_No          N1
         Lap1time        N5
         Lap2time        N5
         Lap3time        N5
         Time            N5
SEQUENCE
         ENTERID  ?
         FINDRECORD
         LET      Lap_No   =        Lap_No + 1
         TEST     (Lap_No = 1)
         IFFGOTO  LABEL 1
         LET      Lap1time  =       TIME/100
         GOTO     END
LABEL 1
         TEST     (Lap_No = 2)
         IFFGOTO  LABEL 2
         LET      Lap2time  =       TIME/100
         GOTO     END
LABEL 2
         TEST     (Lap_No = 3)
         IFFGOTO  END
         LET      Lap3time  =       TIME/100
END
```

MTBCALC.SCH is a race time calculator and checks that each rider completed the requisite number of laps, and then calculates the overall race time for that rider in seconds. The lap times are recalculated from the times at the end of each lap.

```
TITLE    Mountain Bike Race Time Calculation   1 Sept 1991
FILE     MTBSEP1.DAT
ID
         Rider    N3
DATA
         Class    N1
         Lap_No   N1
         Lap1time         N5
         Lap2time         N5
         Lap3time         N5
         Time     N5
SEQUENCE
         NEXTRECORD
         IFFAILGOTO       IFFAIL
         TEST     Rider<100
         IFFGOTO  LABEL  R100+
         TEST     Lap_No=3
         IFTGOTO  LABEL  SET3
         TEST     Rider=0
         IFFGOTO  LABEL  SET9'S
         LET      MEMORY   =   Lap1time
         GOTO     END
LABEL  SET3
         LET      Time   =   Lap3time-MEMORY
         GOTO     LABEL  TIDY
LABEL  R100+
         TEST     Lap_No=2
         IFTGOTO  LABEL  SET2
LABEL  SET9'S
         LET      Time   =  99999
         GOTO     LABEL  TIDY
LABEL  SET2
         LET      Time   =   Lap2time-MEMORY
LABEL  TIDY
         LET      Lap3time   =   Lap3time-Lap2time
         LET      Lap2time   =   Lap2time-Lap1time
         LET      Lap1time   =   Lap1time-MEMORY
```

```
            TEST      Lap3time<1
            IFFGOTO            LABEL  LAP2
            LET       Lap3time$   =   " "
    LABEL  LAP2
            TEST      Lap2time<1
            IFFGOTO            LABEL  LAP1
            LET       Lap2time$   =   " "
    LABEL  LAP1
            TEST      Lap1time<1
            IFFGOTO            END
            LET       Lap1time$   =   " "
    END
```

The Infomate file is next converted to Pipedream form using the file conversion module and is loaded into a prepared spreadsheet at an appropriate slot. Data is then changed to number form in Pipedream and the data sorted and printed using specially prepared CLI files which are not shown here. Details of these depend on the way that the scoring is handled. You could extend the earlier schemas to include rider name with little difficulty, and might use LOG to EPROM as a more secure backup. There is plenty of potential for the budding programmer to expand on these themes: convert the times from seconds to hours, minutes and seconds; print place lists by class of rider etc. etc.

Calculation of Relative Humidity

The following schema shows how a set of raw data can be transformed to the form required using a model of a process. In this example a wet bulb and dry bulb temperature reading are converted into air temperature and humidity for recording. This is a classic example of the use of SCRATCHPAD fields as intermediates in the calculation. The model used to convert from the two temperature readings to humidity is not described here. (Maybe because even we don't understand it?)

```
TITLE     Infomate RH 9/12/91 Keyboard entry of temperature
FILE      RHKEY.DAT
ID
          Read_No  N4
DATA
          TDry     N4.1     \Dry bulb temperature
          RH       N3       \Relative humidity in percentage
SCRATCHPAD
          TWet     N4.1     \Wet bulb temperature
          VP_Dry   N4.1     \Aqueous vapour pressure dry
          VP_Wet   N4.1     \Aqueous vapour pressure wet
SEQUENCE
          ENTER=<+          Read_No  Type Reading No
          MAKERECORD
          IFFAILGOTO        ENTER=<+
          ENTER    TDry     Temp Drybulb?
          ENTER    TWet     Temp Wetbulb?
          LET      VP_Dry  =  EXP(7.076-2.47*((1.46-(TDry/100)  )^2))
          LET      VP_Wet  =  EXP(7.076-2.47*((1.46-(TWet/100)  )^2))
          TEST     VP_Dry=0
          IFTGOTO  LABEL  BadVP
          LET  RH  =  ((VP_Wet-(.645+TDry*6.51E-4)*(TDry-TWet))/
                            VP_Dry)*100
          DISPLAY  RH       RH=
          GOTO     END
LABEL  BadVP
          DISPLAY  *        Bad Vapour Pressure!!~~
          GOTO  ENTER  TDry
END
```

NOTE: Some of the lines above are too long for this page and have been extended on to the next line of the listing. Do not type them this way, but rather put them all on a single line.

Forestry Data collection.

The following schema could be used to assess a forest for tree volume. It records tree ID's, and their diameter and height. Now diameter is easy with a tape measure, but holding the tape on the highest point of a tree is fraught with difficulties beyond the most intrepid forester. Also it requires a second person to read the bottom of the tape. To get around this difficulty, the angles to the top and bottom of the tree from a suitable distance are measured, and height is then estimated using one of four methods that take into account the slope of the land and the availability of trigonometry functions. Now Infomate is good at trigonometry so the following schema will do the whole lot for you as soon as you have selected the method, and typed in all of the angles. In this example, all data is stored in the file so that the 'big' computer can check Infomate's working. There is little trust in this world eh? We will leave drawing the geometrical constructs to the reader.

```
TITLE     Stratum  Height
FILE      STRATUMHT.DAT
LOG       STRATUM
ID
          Plot_No           N6
          Tree_No           N3
DATA
          DBH               N41
          Method            N1
          SlopeD            N31
          UpperR            N41
          LowerR            N51
          AddFact           N21
          Calc_Height       N31
SEQUENCE
          ENTER?<  Plot_No   Plot  Number
          ENTER    Tree_No   Tree  Number
          MAKERECORD
          ENTER    DBH      DBH  (cm)
          ENTER?<  Method    Method
          ENTER    SlopeD    Slope  Distance
          ENTER    UpperR    Upper  Reading
          CHECK    UpperR   *    0    50
          IFFAILGOTO        ENTER  UpperR
          ENTER    LowerR    Lower  Reading  (+/-)
          CHECK    LowerR   *    -10    +10
          IFFAILGOTO        ENTER  LowerR
          ENTER    AddFact   Additional  factor
          TEST     Method=1
          IFFGOTO  LABEL 1
          LET      Calc_Height  =  SlopeD*COS(RAD(LowerR))*(TAN(RAD
                         (UpperR))-TAN(RAD(LowerR)))  +AddFact
          GOTO     LABEL END
    LABEL 1
          TEST     Method=2
          IFFGOTO  LABEL 2
          LET      Calc_Height  =  SlopeD*(UpperR-LowerR)/  (SQR(
                         10000+LowerR^2))+AddFact
          GOTO     LABEL END
    LABEL 2
          TEST     Method=5
          IFFGOTO  LABEL 3
          LET      Calc_Height  =  SlopeD*(COS(RAD(LowerR))^2)*(TAN(RAD
                         (UpperR))-TAN(RAD(Lower)))+AddFact
          GOTO     LABEL END
    LABEL 3
          TEST     Method=6
          IFFGOTO  LABEL 4
          LET      Calc_Height  =  SlopeD*(UpperR-LowerR)*(100.+0.03*
                         LowerR)/(10000+LowerR^2)+AddFact
          GOTO     LABEL END
    LABEL 4
          DISPLAY  Method  Invalid  Method  =
          GOTO     ENTER?<  Method
    LABEL END
          DISPLAY  Calc_Height  Height  Estimate  is  ~~
    END
```

NOTE: Some of the lines above are too long for this page and have been extended on to the next line of the listing. Do not type them this way, but rather put them all on a single line.

THE REFERENCE SECTION

DATA FILES REFERENCE

Data files are the files that Infomate will build up as you collect data. You need to think about how they are constructed because your data files will be different from everybody else's. You will also need to think carefully where your data is to go when it is sent on to a larger computer, because that computer may place restrictions on the form that your data will take. Thus while it may be nice to have M and F meaning male and female in your data file, this will be a nuisance if your data analysis program requires numbers in these fields. Think about these sort of problems early, it is always harder to change them later.

Your data analysis package may not accept data in the form that Infomate stores it internally, but there are conversion programs available within Infomate that will convert to some more commonly used forms. If your format is not handled directly ask an expert for help.

Records

Infomate Data files are divided into records which are each terminated by a carriage return character. All records must be of the same length to be used by Infomate. Records are in turn divided into fields which are the basic unit of data collection. A field might hold an identification number or a weight for example. Fields may comprise from one to at most 24 characters and for Infomate are of two basic types, numeric (or number) fields, and character (or letter) fields. Fields are also grouped into two regions of a record, the ID or identification region which is first and holds the unique identification of the record, and the DATA region which always follows the ID region. Both ID and DATA regions may be composed of mixtures of numeric and character fields, although ID regions place a few restrictions on character fields. Additionally, scratchpad fields may be added to your schema. These are not part of the record and are not stored in the file.

The ID fields

Let us look at the function of an ID region. It's prime aim is to provide a unique identity to each and every record in the file. Thus no two records can have the same ID. Examples of unique ID's might be for an animal weighing trial the animal tag number plus the tag colour (which is related to year born). For a produce collection trial it may be a combination of row, treatment, and plant number. For an animal behaviour study it could be the date and the time that an observation is made. Commonly an ID will be a combination of more than one field.

There are a few limitations on ID fields that do not apply to data fields. ID Number fields must contain only the digits 0 to 9 and the space character. Decimal points are not allowed. ID Letter fields must contain only the letters of the alphabet plus the slash (/) and blank character, and no distinction is made between upper and lower case letters; ab123 and AB123 are considered to be the same, although they will be stored in the form that they were typed.

In addition, the total number of characters in an ID region is limited to 6 for Letter fields and 9 for Number fields. Where there is a mixture of letters and numbers, the following limits apply.

Numbers	0	1	2	3	4	5	6	7	8	9
Letters	6	5	5	4	3	2	2	1	0	0

Thus if there were 3 letters in an ID then the maximum number-field-digits allowed would be 4, no matter what the organisation of the fields was. The reason for this limitation is that Infomate maintains an abbreviated copy of all ID's used so that it can quickly find if a particular ID already exists, and if so where in the file it exists. If you have designed your data and ID fields correctly this should not be a limitation.

Because Infomate requires every record to have a unique ID you should take care in defining these fields. There is a strong temptation to use too few ID fields and end up with records that try to have the same ID, or to have too many and end up with data in the identification area. There is a strong differentiation between DATA and ID in Infomate.

The DATA fields

Data fields enjoy far more freedom than those for ID fields. Letter fields can have up to 24 characters in them and may use any printable character on the keyboard. A distinction is made in data letter fields between upper and lower case characters. Thus abc is different from ABC.

Number fields may contain decimal numbers and can each be up to 9 characters in length. If required, number fields can automatically have the decimal point suppressed in the file to conserve storage.

SCHEMA FILES REFERENCE

A schema file must follow the strict format shown below from the first TITLE line to the final END. This section of the manual describes the schema file in more detail, using the order that a schema is normally laid out. At the end of the section is a chapter on the intricacies of the LET statement which is probably the most complex of all the Infomate statements.

```
TITLE     Your title
FILE      Your data file name
LOG        (optional PRINTER log or EPROM log filename)
PORTIS    (optional device name)
ID
       definitions of your ID fields
DATA
       definitions of your data fields
SCRATCHPAD
       optional definitions of your scratch fields
SEQUENCE
       definitions of your data capture sequence
END
```

The two optional lines may be left out if not required, and the SCRATCHPAD line and field definitions after it are also completely optional and may be left out. To make a schema file easier to read you may put as many extra space characters into each line as you wish. These will be reduced to single spaces when the schema file is compiled by Infomate. Note that it is not permissible to place blank lines in your schema files, nor is it permissible to use TAB characters to space your text. TAB 's have a very different meaning to Pipedream from that usually defined for computers and they should be avoided at all cost. Examples of well formatted schema files are shown in the previous chapter, and you are advised to adhere to this format as much as possible. These formatting points will help you understand your schema files when you look at them again in 6 months time and will make it easier for other people to understand them when you go to them for help or to help them.

Comments in schemas

You may add comments to the end of any of your schema lines by starting it with a backslash character (\). All characters from the \ onwards will be completely ignored by Infomate during compilation and are allowed for your own convenience. Obviously you cannot follow a comment by an Infomate statement. It is preferable that you do not put comments at the start of a line in place of the normal statements, as even though Infomate will tolerate these comments, it will get a little confused when it tries to tell you which line your error has been located in. A way around this is to use extra LABEL statements as dummy Infomate lines in which to place comments.

An example of a comment line is

```
        MAKERECORD               \Run the 100 metres fast
```

If you wish to insert a large area of comments you could do so after the schemas END statement. In this case you would not need to start the comments with the backslash character. This is probably a convenient place to add your hopes and aspirations for the schema, and to list problems or limitations.

Organisation of the Reference Section

The following is a list of the possible lines that can occur in a schema, showing the parts of the lines that are optional and the various formats that they can take on. They are listed in the order that they are presented in in the following chapter.

```
TITLE      {message}
FILE       {filename.ext}
LOG        {EPROMlogfilename}
LOG        PRINTER[optbaudstring]
PORTIS     {drivername}[optbaudstring]   {optINITstring}
ID
           {IDfieldname}     {IDfieldtype}{IDfieldwidth}
DATA
           {DATAfieldname}   {DATAfieldtype}{DATAfieldformat}
SCRATCHPAD
           {SCRfieldname}    {SCRfieldtype}{SCRfieldformat}
SEQUENCE
ENTER      {fld}    {message}
ENTER¦     {fld}    {message}
ENTER?     {fld}    {message}
ENTER?<    {fld}    {message}
```

```
ENTER?<+  {fld}   {message}
ENTER=<   {fld}   {message}
ENTER=<+  {fld}   {message}
ENTERID   {message}
ENTERID¦  {message}
MAKERECORD
FINDRECORD
NEXTRECORD
CHECK     {fld}   {checkstring}
CHECK     {fld1}  {fld2}  {val1}  {val2}
CHECK     {fld1}     *    {val1}  {val2}
CHECKDONE {fld}
IFFAILGOTO        {line}
TEST      {BASICexpression}
TESTRECORD
IFFGOTO   {line}
IFTGOTO   {line}
IFOGOTO   {line}
GOTO      {line}
LABEL     {labelcharacters}
DISPLAY   {fld}   {message}
DISPLAY   *       {message}
REPEAT    {message}
UNTIL     KEYPRESS
REPEAT    {message}
UNTIL     STEADY  {fld}   {value}
REPEAT    {message}
UNTIL TEST
COUNTYPE  {fld}   {typestring}
COUNTSIZE {fld}   {noflds}  {firstval}    {valincr}
LET       {fld}   =       {BASICexpression}
GETPORT   {fld}   {optmessage}
SETPORT   {message}
SLEEP
END
```

The above is not an executable schema so do not try to compile it. I assure you that it will fail.

TITLE

Format: TITLE {message}

The TITLE statement must always be the first line of a schema file and performs no other purpose other than to provide you with a description of the function of your schema file. The title can be displayed at any time within Infomate using the Show Schema command so take care to put into the title all relevant information such as function, date, name of author and anything else that makes this schema different from any other schema that you are likely to write. An example of a Title line is:

```
TITLE        Ag500 June wt capture RBJ 16/7/90. Mod 19/7
```

It is also essential that whenever you modify a schema file that you check that the title is still appropriate to the function that it performs. There is nothing worse than a title displayed in an Infomate Show Status screen that says that the schema does one thing when in fact it now performs an entirely different function.

FILE

Format: FILE {filename.ext}

This describes the name of the data file that you will be placing your captured data into for this schema file. Conventionally the file will have an extension containing the characters .DAT and the filename will contain at most 12 characters before the full stop. The extension should be three characters or less, and should avoid any of the special symbols blank or /+=-*&^ etc. If your files are to go to an MSDOS machine they should have no more than 8 characters before the full stop. An example of a File line is

```
FILE            WTS1807.DAT
```

The filename should not contain a device or directory name; that will be added automatically by Infomate when you move to the appropriate place using Filer while running Infomate. For example your real file name for Z88 purposes may be :RAM.1/WTSDirectory/MyBody.WTS , but for purposes of the FILE statement use the name MyBody.WTS . You must then "move" Infomate to the device/directory called :RAM.1/WTSDirectory/ when you start Infomate up.

LOG

Format:	LOG	{EPROMlogfilename}
OR	LOG	PRINTER
OR	LOG	PRINTER[optionalbaudstring]

The LOG statement is completely optional in Infomate. If you wish to make an EPROM file or printer log of the records that you collect you can do this by inserting a LOG statement at this point. The EPROMlogfilename should have 12 or less characters, and should not contain an extension as in a FILE statement. It will have an extension of the form .LGx added to it by Infomate, where the x is a digit from 0 to 9 or a letter from A to Z. These extension names will be incremented as required and need not concern the operator. If you are logging to a printer, you cannot be using any other device on the Z88 serial port. If you are logging to EPROM, you must have an EPROM inserted before you can run Infomate, and there must be space on it for your data. You cannot log to both printer and EPROM because this indicates a level of mistrust that will not be tolerated. Note that an EPROM log is completely independent of the data file that is maintained in RAM disk on the Z88. Also note that the log file is extremely safe from corruption. For further information read the chapter earlier about EPROM log files. Examples of Log lines are:

```
LOG     PRINTER
LOG     PRINTER[1200NY]
LOG     WEIGHTS
```

PORTIS

Format: PORTIS {drivername}[optionalbaudstring] {optionalINITstring}

The PORTIS statement is also optional in Infomate. If you are collecting data from scales or a similar device then the name of the driver which handles that device is defined by this statement. Note that there is a finite list of drivers available and you must ensure that the device you are using is included. If the device you are using requires a special set up when it is initialised, the set-up string can be inserted after the driver name. An example of the usage of this would be to set a Mettler scale to weigh in oz 's rather than gms. In that case the INIT string would be U oz .

Examples of PORTIS lines are:

```
PORTIS  METTLER_1
PORTIS  METTLER_1          U oz
PORTIS  METTLER_1[9600EN]
```

A Note on Baudrates etc

Both the LOG PRINTER and PORTIS command can have optional character strings added to them to specify the baud rate, parity and flow control characters for the serial port. This string has the form [bbbbpx] and is added immediately after (i.e. no spaces) the word PRINTER or the driver name like this

```
LOG     PRINTER[1200NY]
```

OR

```
PORTIS  METTLER_1[9600EN]
```

If these are present, then the serial port will be set to those values whenever logging to the Printer, or when initialising the device drivers. These values will overwrite the values in the Z88 control panel, and will be used instead of the default values for the driver. If the LOG PRINTER does not have a baud rate specification it will use the current version in the Z88 control panel. If the PORTIS line does not have a baud rate specification, it will use the default baud rate that is specified in the driver itself. The range of values for baud rate, parity and Xon/Xoff are

Baudrate	75, 300, 600, 1200, 2400, 9600, 19200, 38400
Parity	None, Space, Mark, Odd, Even
Xon/Xoff	Yes, No

Note that the first character only of the Parity and Xon/Xoff words should be used, and that there must be no spaces from the driver name to the last] character.

ID region

The ID region starts with the line

```
            I D
```

and is followed by the definitions for each ID field. These have a

Format: {IDfieldname} {IDfieldtype}{IDfieldwidth}
etc. etc.

The ID region contains the names and descriptions of the fields that make up the unique identification of each data record in your file.

The **IDfieldnames** should be strings of letters and numbers without spaces that describe the field. e.g. Tag, Colour, Row etc. The field names should be different from every other field name in the current schema regardless of whether they are ID, DATA, or SCRATCHPAD fields. Infomate is case sensitive and Tag and tag are different field names although it is advisable not to use two such similar names. It is suggested that a convention be used throughout a schema. For example start all names with a capital and make all other letters lower case. If a space is required use the underscore character _ . An example of this might be Tag_Number or Fruit_No .

IDfieldtypes may be of two forms, Letters or Numbers signified by the code letters **L** or **N** respectively.

IDfield width is a single digit defining the number of characters in the field. You should check the limitations on ID field widths discussed in the previous chapter.

An example of an ID field set definition is

```
            I D
               Tag           N3
               Colour        L1
```

which defines an ID with a 3 character numeric animal tag number and a single character letter field for tag colour.

DATA region

The DATA region starts with the line

DATA

which also by default defines the end of the ID region. The DATA line is followed by the definitions of each DATA field. These have a

```
Format:     {DATAfieldname}     {DATAfieldtype}{DATAfieldformat}
            etc.                          etc.
```

The DATA region contains the names and descriptions of the all of the fields that make up the body of a data record in your file. This will also define the length of your data record in the file.

The **DATAfieldnames** should be strings of letters and numbers without spaces that describe the field. e.g. Weight, Length, or Colour etc. As with ID fields, Infomate is case sensitive and Weight and WEIGHT are different field names although it is advisable not to use such similar names. The field names should be different from every other field name in the current schema regardless of whether they are ID, DATA, or SCRATCHPAD fields. It is suggested that all names start with a capital letter and are followed by lower case letters. If a space is required use the underscore character _ . An example of this is the field name Old_Weight .

DATAfieldtypes are of two forms, Letters or Numbers signified by the letters **L** or **N** respectively.

DATAfieldformat is a sequence of one to three characters describing the format of the data to be placed in the record of the file.

For Letter fields the format will be one or two characters defining the number of characters in the field. Thus an L3 field is a letter field with 3 characters. The maximum size of a letter field is 24 for DATA fields.

For Number fields the first character is the total field width and the last is the number of decimal places to be recorded. If a decimal point is to be stored in the file then the two digits should be separated by a decimal point. For example if numbers like 23.6 are to be stored then use format N4.1 . If you want the decimal point to be eliminated at storage time to save space, then use N31 to store the numbers in the form 236 (i.e. 23.6 with an **implied decimal point**). The maximum size for a numeric data field is 9 digits, which is about the accuracy of the arithmetic in the Z88. Obviously the number of decimals cannot exceed this.

An example of a data region definition is:

```
DATA
          Weight            N3.1
          Wool_colour       L2
          Dag_score         N2.0
          Comment           L10
```

SCRATCHPAD region

The SCRATCHPAD region is optional and may be left out if desired. It starts with the line

SCRATCHPAD

which also by default defines the end of the DATA region. The SCRATCHPAD line is followed by the definitions of each SCRATCHPAD field. Scratchpad fields are fields that are not to be stored in a file, but are either intermediate values that are used to simplify the calculation of other field values, or are values that are to be "remembered" from record to record. Scratchpad field values will be cleared whenever a new schema is compiled, and will not be affected by reading a record using the FINDRECORD, nor by writing a record when an END is met in the schema file.

You can think of scratchpad fields as extra MEMORY locations with more meaningful names and with complete format specifications. The format of a scratchpad field follows the identical definitions as for the DATA fields above. As with DATA and ID fields, the field names should be different from every other field name in the current schema .

SEQUENCE region

The sequence region contains a description of the way that you want the data to be collected. It starts with the line

SEQUENCE

which also has the function of terminating the data or scratchpad field definitions. The contents of the sequence region will vary considerably depending on your requirements and should be thought of as a computer program. The sequence is a definition of the steps you wish to go through to collect each record of your data. In the following sections we provide a list of the different types of statements that may be placed in your sequence region. Many of them will not be useful to you in particular, and most applications will require only a few different types.

ENTER

Format: ENTER {fld} {message}

Normal, no frills, entry of a field value from the keyboard. The message {message} will be displayed to the operator who may then type the new value for that field. The entry should be terminated using the ENTER key. Examples:

```
ENTER   WT      Type animal weight
ENTER   ROW     Key in Row Number
```

You may also append ! characters to the end of the message part of an ENTER statement to make the Z88 beep to warn the operator that data entry is required. One ! produces one beep.

ENTER¦

Format: ENTER¦ {fld} {message}

Modified entry of a field value from the keyboard which allows the operator to skip typing the ENTER key. The entry will be terminated when sufficient keys have been typed to fill the field, or if less than the full number are required, when an ENTER key is pressed. Thus, for a field of type N3 the following entries will be accepted

```
123             which puts 123 in the file
12ENTER         which puts  12 in the file
1ENTER          which puts   1 in the file
```

Note that if the field has an implied decimal point you will need to type an extra character (the decimal point). Thus a field of type N31 would accept entries of the form

```
12.3              which puts 123 in the file
12ENTER           which puts 120 in the file
```

An example of an ENTER¦ statement follows:

```
ENTER¦   Colour   Press Colour letter (RGB)
```

You may also append ! characters to the end of the message part of an ENTER¦ statement to make the Z88 beep to warn the operator that data entry is required. One ! produces one beep.

ENTER?

Format: ENTER? {fld} {message}

Entry of a field value giving the operator the option of editing and using the value that is currently held in the file in that field position. If the ENTER key is pressed at any stage the value currently displayed on the screen will be entered into the field. An example is:

```
ENTER?   Fate    Edit old Fate
```

This statement will normally be used if you wish to **modify** the data that is held in a file using the the keyboard rather than adding new data to the field. An example is to allow addition to or modification of comments held in the file.

You can use these at a number of points in one schema with the same comment field, enabling you to edit the same comment at each point if required, and thus allow creation of a composite comment from any point where comments could be generated.

You may also append ! characters to the end of the message part of an ENTER? statement to make the Z88 beep to warn the operator that data entry is required. One ! produces one beep.

ENTER?<

Format: ENTER?< {fld} {message}

Entry of a field value giving the operator the option of editing or using the value entered for this field in the previous record. This would typically be used to enter field values that don't change for each entry e.g. the row number when collected vegetable weights a row at a time. For example:

```
ENTER?< Row      Row Number
```

You may also append ! characters to the end of the message part of an ENTER?< statement to make the Z88 beep to warn the operator that data entry is required. One ! produces one beep.

ENTER?<+

Format: ENTER?<+ {fld} {message}

Entry of a field value giving the operator the option of editing or using the value entered for this field in the previous record incremented by 1. It is typically used to enter field values that follow a simple numerical sequence for each entry e.g. the vine number when recording fruit data along a row. It is possible to make the sequence count backwards by using a large negative value for the initial value. Thus for an N3 field entering -1999 will produce the sequence 999, 998, 997, etc. This truncation effect occurs when Infomate tries to put a 5 character negative number into a 3 character field.

An example of the ENTER?<+ statement is:

```
ENTER?<+        Vine     Vine number
```

You may also append ! characters to the end of the message part of an ENTER?<+ statement to make the Z88 beep to warn the operator that data entry is required. One ! produces one beep.

ENTER=<

Format: ENTER=< {fld} {message}

Entry of a field value automatically (i.e. without the operator actually typing anything) using the value entered for this field in the previous record. This would typically be used to enter field values that don't change very often e.g. the row number when collecting vegetable weights in a long row. If it is required to change the value entered into an automatic ENTER= type statement while running the schema, the operator may back-step to it using the ⇧ key, providing there are statements following the ENTER= that allow operator intervention. An ENTER=< statement might look like this:

```
ENTER=<  Row      Enter  Row  Number
```

You may also append ! characters to the end of the message part of an ENTER=< statement to make the Z88 beep to warn the operator that data entry is required. This beep will only occur when this statement actually requires data to be entered. One ! produces one beep.

ENTER=<+

Format: ENTER=<+ {fld} {message}

Entry of a field value automatically using the previous value entered for this field with 1 added to it. This would typically be used to enter field values that change in a systematic way e.g. the vine number when collecting fruit data along a row.

An example of the ENTER=<+ statement is:

```
ENTER=<+          VineNo   Enter  vine  number
```

You may also append ! characters to the end of the message part of an ENTER=<+ statement to make the Z88 to warn the operator that data entry is required. This beep will only occur when this statement actually requires data to be entered. One ! produces one beep.

ENTERID

Format: ENTERID {message}

Allows entry of an entire ID set in a single line. This is most often required in an animal weighing situation where the full ID may consist of a number of parts which are thought of as a unit. Thus, for the following ID definition

```
ID
          Year_born        N1
          Tag              N3
          Colour           L1
```

you might enter ID's as 8123Y meaning year born is 1988, tag is 123 and tag colour is Y for yellow.

Infomate will use quite a bit of intelligence in interpreting ID's entered in this way, provided the result is unambiguous. Thus for the above ID definition a 1 or 2 digit tag may be entered without leading zeros e.g. 812R will be interpreted as year 88 tag 012 colour Red. Care should be taken when there is any chance of ambiguity. Thus if Year born was an N2 field and was followed by an N2 tag number , Infomate would interpret 812 as year born 81 tag 2 and not year born 8 tag 12. But then again wouldn't you?

An example of this statement is:

```
ENTERID  Type  the  ID  (YTTTC)
```

You may also append ! characters to the end of the message part of an ENTERID statement to make the Z88 beep to warn the operator that data entry is required. One ! produces one beep.

ENTERID¦

Format: ENTERID¦ {message}

This is a combination of the ENTERID verb just described, and the ENTER¦ described above. If the operator types in a full ID then the ENTER key need not be pressed. If the ID is to be shortened by not typing any leading zeros, the ENTER key can be used. In all other respects it will follow the rules of the ENTERID verb just described. For example:

```
ENTERID¦          Type  the  ID  (YTTTC)
```

You may also append ! characters to the end of the message part of an ENTERID¦ statement to make the Z88 beep to warn the operator that data entry is required. One ! produces one beep.

MAKERECORD

After the ID information has been entered, and before data can be placed in the file, it is necessary to reserve some space in the file for that data. At the same time it is necessary to check that the ID typed conforms to the schema files ID definition, and that a corresponding ID does not already exist in the file. When these checks have been completed satisfactorily, all data fields in the new record are cleared. The MAKERECORD command handles all of these operations. If there is insufficient space in the file, or if the ID typed already exists, an error message will be posted and may be checked using the IFFAILGOTO statement below. Infomate will also clear any errors from CHECKS etc. that were pending at the time the MAKERECORD statement was started. Obviously the MAKERECORD statement should occur at the end of the entry of all ID fields, and before collecting any data fields. An example of the use of MAKERECORD might be:

```
ENTERID         Enter full ID
MAKERECORD
IFFAILGOTO      ENTERID
```

FINDRECORD

Format: FINDRECORD

In the situation where the data file already exists and is having new data added into existing records, a FINDRECORD statement is used in place of the MAKERECORD just described. This statement will search the file for the record that has that ID, and when found, load it into memory. At that point all previous values for the data fields will be shown and may be changed or added to by further sequence statements. If the ID typed does not conform to the ID definition in the schema file, or if the search fails to find the record in the file, then an appropriate error message will be posted and may be checked using an IFFAILGOTO statement. Infomate will also clear any errors from CHECKS etc. that were pending at the time the FINDRECORD statement was started. An example of this procedure is:

```
ENTERID         Enter full ID
FINDRECORD
IFFAILGOTO      ENTERID
```

NEXTRECORD

Format: NEXTRECORD

This is equivalent to the FINDRECORD statement, but will sequentially move through a file from start to finish in the order that the data is stored. It is intended for automatic processing of an entire file. For example, the following sequence will set all entries for field Fruit_Count to zero for the whole file.

```
SEQUENCE
        NEXTRECORD
        LET  Fruit_Count  =  0
END
```

There is no way that a NEXTRECORD can be started part way through a file, and it will not find the NEXTRECORD after one found by a FINDRECORD. Infomate will also clear any errors from CHECKS etc. that were pending at the time the NEXTRECORD statement was started.

CHECK (character)

Format: CHECK {fld} {checkstring}

This allows checking of Letter fields to ensure that they conform to a specific set of possible values. The checkstring is composed of the possible character sequences, preceded by a sequence of asterisks (*) that has the same length as each of the sequences to be checked. Thus a checkstring of the form *MF will make sure that the field typed is a single character (note one asterisk) which is either the letter M or the letter F. The checkstring ***RYECLVCOXWED will check for one of these three character sets: RYE, CLV, COX, or WED . The CHECK statement is case sensitive, so if you want to allow both "m" and "M" as possible characters, they should both be inserted into the character string. The above example would appear as:

```
CHECK    Species ***RYECLVCOXWED
```

If the checks involve mixtures of single and double characters they can be checked using a single check statement. Thus R, O, Y, GN, GY could be checked using

```
CHECK    ** R O YGNGY
```

Note that the single characters must be preceded by a single blank. Do not try this with groups that contain two blanks because Infomate eliminates all multiple blanks from lines before it looks at them. This might be achieved to check the patterns A, AB, ABC by adding $ characters in place of the blanks. Thus

```
ENTER      Pattern    Patterns?
LET        MEMORY$    =   RIGHT$("$$$" + Pattern,3)
CHECK      MEMORY$    ***$$A$ABABC
```

CHECK (numeric)

Format:	CHECK	{fld1}	{fld2}	{val1}	{val2}
OR:	CHECK	{fld1}	#	{val1}	{val2}

Two numeric checks are provided for here. In the first form the value held in {fld1} is compared to the range defined by {fld2) and the lower and upper limits defined in {val1} and {val2}. i.e.

{fld2} + {val1} ≤ {fld1} ≤ {fld2 + {val2} .

If the value lies outside this range a WARNing message will be posted which may be checked by a following IFFAILGOTO statement.

The second form of the CHECK is an absolute check against the values of {val1} and {val2}. i.e.

(val1} ≤ {fld1} ≤ {val2}

You can think of the # symbol as meaning "number". Examples of a relative and an absolute numeric check are:

```
CHECK    Newweight          Oldweight    -1.0    +3.0
```

which checks Oldweight-1.0 ≤ Newweight ≤ Oldweight +3.0

```
CHECK    Size    #          0.0       99.9
```

which checks that size is between 0.0 and 99.9 units i.e whether it fits into an N4.1 numeric field. Note that these checks do not suffer the numeric difficulties of the TEST statement.

CHECKDONE

Format: CHECKDONE {fld}

This checks if a field already has a value entered into it. If so, it will set a FAIL message which can be checked by an IFFAILGOTO verb following it. This verb would normally be used in an animal weighing situation to ensure that the new weight being recorded for an animal has not been placed in the file already, signifying perhaps a misread tag on either the present animal or on an animal previously weighed. It assumes that the pre-prepared file of animal ID's has blanks placed in the fields that are to have a CHECKDONE performed on them. Note that a zero value (0.0) IS considered to be an already-done field! An example of a check statement is:

```
CHECKDONE       New_Weight
```

IFFAILGOTO

Format: IFFAILGOTO {line}

This statement would usually follow one of the CHECK statements, the MAKERECORD, FINDRECORD or the NEXTRECORD discussed above. The {line} definition is a sequence of characters which Infomate will search for from the start of the sequence region of the schema file. Only sufficient characters to uniquely define the line to be GOTOed need be inserted.

In the example below the operator will be repeatedly asked for the sex of the animal until they respond with M or F. Note that because there is a line starting with ENTER prior to the line required it is necessary to add further characters.

If any difficulty arises with this use a LABEL statement as defined below.

```
ENTER    WT      Weight
ENTER¦   Sex     Sex (MF)
CHECK    Sex     *MF
IFFAILGOTO       ENTER¦
```

Remember that Infomate deletes all spaces at the start of a line, and reduces all multiple spaces to a single space, so that the last line of the previous example could equally well be either of these:

```
IFFAILGOTO       ENTER¦ Sex

IFFAILGOTO       ENTER¦           Sex
```

TEST

Format: TEST {BASICexpression}

This is a rather more general statement to handle checks that cannot be performed using the CHECK statement. The result of the TEST is stored in a special flag that can be tested using the IFFGOTO (IF FALSE) and IFTGOTO (IF TRUE) statements below. It does not affect the FAIL flag set by the CHECKS etc. The TEST flag is also set by the TESTRECORD and LET statements and examples of its use are provided in the IFFGOTO and IFTGOTO statement descriptions below. Typically you would use it to skip parts of your sequence. e.g..

```
TEST     State  =  "DEAD"
IFFGOTO  LABEL  Alive
```

The TEST statement can also be used to set up a test for the end of a REPEAT UNTIL TEST loop.

For example to insert a pause in your schema try this

```
LET      MEMORY  =  TIME(TIC)
REPEAT
         TEST    (TIME(TIC)-MEMORY)  >  500
UNTIL    TEST
```

which will wait until the TIC TIME counter advances by 500 centiseconds (5 seconds) before proceeding.

You should note the warning in the earlier chapter on ARITHMETIC and the MEMORY about accuracy of representing numbers in TEST statements. If care is not taken with how you write TEST statements, they can fail to operate as you intended.

TESTRECORD

Format: TESTRECORD

This statement asks the question "is the ID data that has been entered valid, and if so does that particular record exist". If both are true, then the TEST flag is set and may be tested using IFFGOTO or IFTGOTO below. It is important to note that TESTRECORD does not either load the record into memory if it exists nor does it make a record if it doesn't exist. These functions must be performed using FINDRECORD or MAKERECORD respectively. A typical application of TESTRECORD is to allow branching to a MAKERECORD or a FINDRECORD depending on whether the record exists or not. e.g..

```
ENTERID         ID?
   TESTRECORD
   IFTGOTO      FINDRECORD
MAKERECORD
   IFFAILGOTO   ENTERID
  GOTO          LABEL      1
FINDRECORD
LABEL           1
```

Note that if the ID is invalid, the TESTRECORD will set the TEST flag FALSE and it is left to the MAKERECORD to pick up the "Invalid ID" problem.

IFFGOTO and IFTGOTO

Format: IFFGOTO {line}

OR: IFTGOTO {line}

These two statements are designed to be used after TEST, TESTRECORD or LET statements, and will branch to the first line in the schema that matches the text after the GOTO if the Appropriate condition is correct. An example might be:

```
TEST             (Sex = "M") OR (Sex = "m")
IFTGOTO          LABEL   Male
    ENTER        TEATSCORE        Enter  Teatscore
LABEL   Male
```

Alternatively they can be used on numeric fields to look at ranges of values. For example

```
LET      Obese    = (Weight   >   100.0)
IFFGOTO           LABEL    Not_Obese
    ENTER         ObRSN    Enter reason for obesity
LABEL    Not_Obese
```

This will set the single digit field to a 1 if weight is greater than 100, and to 0 otherwise. If the animal is not obese, then the obesity question is skipped. Note that in fact the result of the test is set to -1 if true, but the minus will be eliminated when it is squeezed into the single character field.

Comments in the IFFAILGOTO statement above regarding the handling of {line} apply equally to this one. These statements allow the path taken through a sequence file to be altered according to the value held in a field. Thus if different values were to be collected depending on the sex of an animal use one of these GOTO's.

Considerable care should be taken with this form of flow control. Attempts should be made to ensure that the entire sequence file is always processed from top to bottom, with short regions skipped if necessary. Try to keep the path through a sequence as simple as possible. Never get your sequence into an endless loop of the form:

```
LABEL    1
etc.
GOTO     LABEL    1
```

IF0GOTO

Format: IF0GOTO {line}

Provides a way of checking that the result of a LET or TEST statement was zero. For example to test if the number of lambs is zero use this code

```
TEST     No_Lambs
    IF0GOTO       LABEL    Skip_Lambs
         statements to enter lamb data goes here
LABEL    Skip_Lambs
```

The comments about {line} in IFFAILGOTO above apply here as well.

GOTO

Format: GOTO {line}

Sometimes it will be necessary to unconditionally switch control to another part of the sequence. This can be done using a GOTO statement. An example could be where different code is executed for males and females by the following construct.

```
TEST     (Sex = "M")
    IFTGOTO       LABEL    Male
         code for females goes here
         GOTO     LABEL    End_of_sex_dependant_bit
LABEL    Male
         code for males goes here
LABEL             End_of_sex_dependant_bit
```

The comments about {line} in IFFAILGOTO above apply here as well.

LABEL

Format: LABEL {labelcharacters}

Often when performing one of the GOTO's statements it is difficult to define a line to GOTO just using the contents of the line itself. This may happen if an IFFAILGOTO has to GOTO a particular ENTER statement in a sequence of ENTER's. To remove this ambiguity it is advisable to use a GOTO LABEL statement. Obviously the characters following the LABEL must be unique and must match in both the GOTO and the LABEL statements. An example is:

```
LABEL    GetID
ENTERID  Enter ID
IFFAILGOTO        LABEL    GetID
```

DISPLAY

Format: DISPLAY {fld} {message}

OR: DISPLAY # {message}

In some situations it may be necessary to show the operator the value of a particular field in the file. This may occur when a value is computed, or if the value is to be extracted from a file that has been created elsewhere. One classic example is a file containing cull status of animals that has been down-loaded from another computer.

The sequence to use here might be:

```
          DISPLAY Cull_state       Cull status is
```

which if Cull_state for that animal was "draft" would put the message

```
          Cull status is draft
```

onto the screen and wait for the operator to press a key before continuing.

If you wish to display a message without also displaying a field value, then use the # form above. e.g..

```
          DISPLAY          #        Silly Boy
```

Displayed messages may be terminated with ! characters to beep the Z88 speaker. Each ! will produce a single beep. These could be useful if the DISPLAY message is issuing a warning to the speaker.

```
          DISPLAY          #        Silly, silly boy!!!
```

It is sometimes desirable to display a message or a field value for a short time and then continue. This can be achieved using the ~ character which builds a one second delay into the display. As soon as the delay is complete, the sequence continues. The delay or pause can be terminated before time by hitting the space bar. You could use the following to display briefly the result of a calculation.

```
          LET     Total_Eggs       = White_Eggs + Choc_Eggs
          DISPLAY Total_Eggs       The Egg Count was ~~
```

The above would show

```
          Egg Count was 4
```

for two seconds and then continue.

You can have any combination of Beeps and Pauses at the end of a message, and the order they occur in is not important. Unfortunately

```
          DISPLAY          #        !~~!~~!!!
```

will not sing "roll out the barrel".

REPEAT UNTIL KEYPRESS

Format:

```
REPEAT   {message}

more statements go here

UNTIL    KEYPRESS
```

This is a very rudimentary looping structure which has been set up to allow certain weigh scales to be used, and to allow multiple values to be entered into the COUNTSIZE statement. It is not intended to be used to collect multiple fields as you might in higher level languages with data arrays. The following example shows the use of a REPEAT UNTIL loop in the situation where scales do not provide a steady reading and the operator must decide when to accept the value being displayed.

```
REPEAT            Weighing
        GETPORT   WT        Weight  is
UNTIL             KEYPRESS
```

Another example of its use is provided in the COUNTSIZE statement below.

The REPEAT   UNTIL loop, and all statements between may be considered to be a single statement for some purposes. If during the course of data entry the operator back-steps through his sequence using the ⇧ key, then on re-entry to the REPEAT loop he/she will have the message {message} displayed on the screen and be asked if it is desired to repeat the loop. If the choice Skip-the-loop is chosen, control will pass to the statement immediately following the UNTIL statement.

REPEAT UNTIL STEADY

Format:

```
REPEAT   {message}

         more statements go here

UNTIL    STEADY  {fld}      {value}
```

This looping structure is intended to capture weights from scales etc. until the readings obtained for field {fld} are steady to within a range of {value} units. Little work has been done on the steadying algorithm at this stage and it may be necessary to build in some LET statements prior to the end of the loop to smooth the data or perform other processing.

REPEAT UNTIL TEST

Refer to the TEST statement above for details of this statement.

COUNTTYPE

Format: COUNTYPE {fld} {typestring}

COUNTTYPE is one of two Infomate statements that allows the collection of data into a number of fields simultaneously. In effect it will form a Histogram over a number of fields, but in fact can be used for a variety of functions. To describe COUNTTYPE it is best to first use an example commonly found in research. The task is to observe a specimen under a microscope and to count cells of four different types. The observer works systematically through a grid pattern displayed on the viewing area and wishes to count each cell type as it is found and identified. The cell types are defined as Flat, Dished, Elongate and Cuboid and it is expected that up to 20 cells of each type could be present in each sample. The data field definitions for this part of the problem would appear as

```
DATA
          Flat          N2
          Dished        N2
          Elongate      N2
          Cuboid        N2
```

The grouping of these fields is important, and they must all be numeric fields. However, they do not in fact have to be same width. In the sequence region the data collection statement could be:

```
COUNTTYPE          Flat        FDEC
```

This means we are going to count the quantities in the 4 fields (because there are 4 letters in the typestring) starting with the field Flat. Each field will be incremented when the operator presses one of the single keys in the sequence FDEC. Thus to increment Flats just press an F , press it again and it will increment again. To decrement a field press the DEL key and then the letter of the field to be decremented. When all counting has been completed press the ENTER key and sequence processing will continue. There are a few limitations to the usage of this statement. Obviously it is not possible to increment letter fields or in fact any of the ID fields, and obviously the definition should not attempt to increment fields that are beyond the end of the last data or scratchpad field.

The statement can also be used for indicating presence of objects rather than counting them. Thus if the fields were species of plants you could indicate that that species was present in the observation plot by pressing the appropriate letter. Your data analysis may then need to account for the possibility of pressing one key twice and getting a 2 in the field.

The letters do not in fact have to be mnemonics for the fields (i.e. F means Flat etc.) but might be labelled keys on the keyboard, or keys in particular positions on the keyboard. For counting while observing under a microscope you might be best to use the keys QWER because of their more easily located position on the keyboard. The operator would just remember which finger to tweak.

COUNTSIZE

Format: COUNTSIZE {fld} {noflds} {firstval} {valincr}

The COUNTSIZE statement is closely related to the COUNTTYPE statement but rather than counting types of things, it counts in a number of size categories. An example might be a fruit sizing experiment where fruit weights are entered through the keyboard and a histogram of weights is formed in a number of fields.. An example in Kiwifruit trials would be to arrange for 6 weight ranges in the following way.

```
DATA
          .
          UNDERS          N2
          W65-84          N2
          W85-104         N2
          W105-124        N2
          W125-144        N2
          OVERS           N2
          .
          etc.
```

The sequence region would run a REPEAT UNTIL KEYPRESS loop in the following way.

```
REPEAT          Sizing
    ENTER       MEMORY    Fruit  weight
    COUNTSIZE   UNDERS    6        45.0      20.0
UNTIL           KEYPRESS
```

This would loop continuously until the ENTER key was pressed twice rapidly. Each time round the loop, the weight entered into the MEMORY "field" would be divided down to see which category it belonged to on the basis that the first category is 45.0 to 45.0+20.0, the second would be 45.0+20.0 to 45.0+2*20.0 etc. Any weights below 45.0 will be counted in the first category, and weights over 165 will be counted in the last category. Obviously this will build up a histogram of the fruit weights in the group being collected. In practice the ENTER statement might be replaced by statement to fetch weights directly from electronic scales or lengths from a caliper.

LET

Format: LET {fld} = {BASICexpression}

The LET statement without doubt is the most powerful statement available in Infomate. It will allow you to perform very complex arithmetic, and gives access to a powerful set of Time and Date functions. More details of the LET statement are provided in the next chapter. In its simplest form it will allow you to copy numeric field values from one field to another. For example:

```
LET      New_weight      =  Old_weight
```

If your scales were imperial scales and you wished the results to be stored as metric values how about:

```
LET      Met_weight      =  Imp_weight    *   0.45
```

Note that you should read the = sign as the phrase "is replaced by". Another scale conversion might be:

```
LET      TempC    =    (TempF -32) * 5 / 9
```

You can see that the possibilities are unlimited here but there is more yet. The BASICexpression on the right hand side of the = sign can also contain some of the standard BASIC language functions like SIN for the Trigonometric sine of a number, SQR to take a square root or use LOG or LN to take the base 10 or natural logs of numbers or fields. A full list of the functions available is given at the end of this section. Some care must be taken with the fields that you use in the expressions. If they are Number fields they can be used by just putting in their names. You should take care in Infomate with statements like

```
LET      Wt   =   Wt   *   0.45
```

because each time that the statement is executed the field Wt will be made even smaller. This can happen if you back-step through a file to correct an earlier field value.

To copy letter field values you can use similar constructs to those for numeric Fields. Thus

```
LET      Son_Sirname        =          Dad_Sirname
```

If you are assigning specific sequences of characters to a letter field they must be enclosed in quotes like this

```
LET      State    =        "Dead"
```

Further examples of letter LET statements are provided in the chapter on the LET statement that follows this section.

If your LET statement must refer to a field value that was used in the previous record, then use the characters (<) after the field name. Thus if the name of the field was Rat_Bag , then Rat_Bag(<) would refer to the old Rat_Bag value placed in the previous record of the file. If you are referring to a string value of a number field on the right hand side of the = sign in a LET expression, put the $ symbol first. Thus use Rat_Bag$(<) .

You should never use a (<) on the left hand side of the = sign in a LET statement. This would allow you to change the past and that can never happen.

GETPORT

Format: GETPORT {fld} {message}

This statement is very similar to the ENTER statement except that it fetches a value from the device defined in the PORTIS line at the start of the schema file. Obviously that device must be connected to the Z88 when the statement is executed, and it must be weighing or measuring the appropriate thing at the time. This may require other statements to be put into the sequence to control this. For a mettler scale the following may be appropriate.

```
PORTIS     METTLER_1
etc.
.
DISPLAY    *          Press ENTER to weigh baked beans
GETPORT    Been_Wt    Beanzo
```

Some devices will not transmit a steady weight and it will be up to the operator to decide when the weight is steady.

This might be done using:

```
PORTIS   OLDSCALES
etc.
.
REPEAT
         GETPORT     Moose_Wt          Weigh da moose
UNTIL    KEYPRESS
```

Or if you are careful use:

```
PORTIS   OLDSCALES
etc.
.
REPEAT
         GETPORT    Moose_Wt          Weigh da moose
UNTIL    STEADY     Moose_Wt          0.2
```

which will wait until Moose_Wt is steady to 0.2 kg .

The Message part of the GETPORT statement may have beeps and pauses added to it in the same way as the Display statement. These should be added at the end of the message using one ! per beep and one ~ per second of pause. The beeps will be sounded before weighing begins, and the pauses will occur after the result from the GETPORT has been displayed. The pauses will allow the operator to see the weight or other measurement before it goes to the file, and can be terminated by pressing the space bar.

If the message part of the GETPORT is eliminated entirely, there will be no message sent to the operator at the start of the GETPORT, nor will the result of the GETPORT be displayed when it has been received through the serial port. This is a convenient way of making the GETPORT transparent when it is linking to analogue to digital converters etc. More details are provided in the serial port chapter earlier.

SETPORT

Format: SETPORT {message}

This statement sends a command held in the message to either the device connected to the Z88 serial port, or to the program that is driving that device. Which of these tasks is performed will be dependent on the driver program selected by the current PORTIS line at the start of the schema. Details of the operation of this statement can be found in the section describing the device drivers later.

A rather trivial example might be to capture weights from a Mettler scale first in oz and then in grams. This could be done using:

```
SETPORT  U  oz
GETPORT  Oz_Wt    Weigh  yer  ouncers
SETPORT  U  g
GETPORT  Gm_Wt    Weigh  yer  grammas
```

Note that the Mettler command U oz changes the scale units to oz's. However you should take care with the above constructions. If the code is backstepped to using the ⇧ key, the GETPORTs will be executed without intervening SETPORTs. A solution to this is to enclose each pair of port instructions in a loop like this:

```
REPEAT
        SETPORT  U  oz
        GETPORT  Oz_Wt     Weigh  yer  owncers
        TEST  1
UNTIL  TEST
```

This will execute the loop once only but will backstep to the REPEAT.

SLEEP

Format: SLEEP

An appropriate statement to put near the end of a long list of statements. This statement literally puts the Z88 to sleep. A trivial example of its use is the night watchman schema which was described earlier. To awaken the Z88 again the operator should press the two SHIFT keys. There are other ways of using the SLEEP statement. You may wish to record occasional animal behaviour observations. In this case the Z88 can be put to SLEEP and will start the question and answer session when you press the two SHIFT keys. Time can be recorded automatically immediately Infomate starts. A further example of this is provided with the ADAM data logger schema described at the end of the manual.

END

Format: END

This is always the last statement in a schema file that is read by Infomate. Any statements following this will be ignored and may be filled with notes or comments if desired. In effect the END statement says to the program finish this cycle of the SEQUENCE , check if there are any errors that have not been noted, if there are none, store the current values of all fields in the file and then go back to the start of the SEQUENCE and start it all over again.

LET STATEMENT REFERENCE

Because of the potential complexity of the LET statement all of its special features are collected here.

LET and TIME

There are a number of special functions written for use with the LET statement that give you access to the real time clock that is incorporated into the Z88. This means that you can automatically record the date or the time that events occur when you collect data. There are 10 built in functions that can be used on the right hand side of a LET statement. These are shown below as they apply to the time 23:23.59 pm on date Wednesday the 31st of December 1991.

Function	Meaning	Format	Example
TIME(HMS)	Hours Mins Secs	N6	235959
TIME(HM)	Hours Minutes	N4	2359
TIME(DOW)	Day Of Week	N1	3
TIME(DOM)	Day Of Month	N2	31
TIME(M)	Month	N2	12
TIME(Y)	Year	N4	1991
TIME(DMY)	Day Month Year	N6	311291
TIME(DM)	Day Month	N4	3112
TIME(DOY)	Day Of Year	N3	365

These functions produce numbers that look like those shown, but take care: you cannot subtract one from the other and get an elapsed time. Elapsed time can be handled by a technique described below. Some of the other functions not listed above can be obtained by one of two techniques, either by chopping of the unwanted bits of the functions above, or by adding the appropriate bits together. Thus to record minutes and seconds only, store hours minutes and seconds (TIME(HMS)) in a 4 character number field. The hours digits will be cut off at storage time. To store year and day of year in a 7 digit number use:

```
LET     DATE    =       TIME(Y) * 1000 + TIME(DOY)
```

However this example will cause a problem at a certain time of the year. If new year occurred between the calculation of the two Functions you would get either the right year and the wrong day, or the right day and the wrong year, depending on your point of view. Now that might not happen very often, but if you were adding an hour function to a minute function it might be a little more likely to occur.

To avoid the midnight trap make all functions other than the first a negative number. Thus use

```
LET     DATE    =       TIME(Y) * 1000 + TIME(-DOY)
```

in which the negative value in the second function will make the second function use the time that was obtained for the first. You can use this trick to advantage with care in that you could store the time that the last data point was collected by using TIME(-HM) . You will then need to collect the time that this point was collected using another TIME(HM) . An example of this is

```
LET     Last_Sample     = TIME(-HM)
.
etc.
.
LET     MEMORY          = TIME(HM)
```

Note that the last statement shown records the time of this sample for the next record. MEMORY itself is not used for this function, but rather we are forcing the TIME module to record the date and time internally. The MEMORY is just a dummy field name because every LET must have a field name on the left. You could equally well use TEST TIME(HM) in place of the final LET.

Elapsed time

One further time function is available to allow the computation of elapsed time. This is the function TIME(TIC) and is the time in 1/100's of a second since the most recent schema file compilation took place. This number will become computationally very big quite quickly and will in fact overflow the Z88 arithmetic after some 5 months. However, if differences are used it will give you a good elapsed time counter. A simple example of its use follows.

This example times a particular operation and stores the result to the nearest second in a field called Elapsed_time.

```
ENTER   MEMORY  Press ENTER to start and stop timer
LET     MEMORY  = TIME(TIC)
   REPEAT
   UNTIL KEYPRESS
LET     Elapsed_time   = INT( (TIME(TIC) - MEMORY)/100 )
```

There are a couple of points to note here. First, MEMORY is used as a storage for time at start, and an empty REPEAT UNTIL KEYPRESS loop is used to stop the timer. Often you can probably use a data entry to stop the timer, rather than the loop. Take care if you back step through your file using the ⇧ key because all of the times will be retaken. Sometimes you will want this to happen and sometimes not. Win-some, lose-some!

Calculation order

When executing LET statements some operations have higher priority than others. Thus if you perform 6+4*3 , the multiplication will be done before the plus. If there is ever any doubt which way things will be performed, put brackets around the bits that you want to perform first. Thus in the above expression you could use 6+(4*3) . Other priorities are less obvious and if you have trouble writing them, put in brackets just in case. It will then be obvious to you later when you try and work out what is going on. However, for the purist, here is the priority order used in LET statements.

Highest priority i.e. Done first

Field values , TIME()

Brackets , unary+ , unary- , NOT

^

* , / , MOD , DIV

+ , -

= , < , > , <= , >= , <>

AND

OR , EOR

Lowest priority i.e. Done last

Functions available

The following functions are supported by the Infomate LET statement:

The first set give number field results.

ABS( )	Absolute value
ACS( )	Arc cosine
ASN( )	Arc sine
ATN( )	Arc tangent
COS( )	Cosine
DEG( )	Converts radians to degrees
DIV	Integer divide. e.g. 7 DIV 2 gives 3
EXP( )	Exponential base e
INT( )	Integer part of
LN( )	Natural logarithm
LOG( )	Logarithm base 10
MOD	Integer remainder. e.g. 7 MOD 2 gives 1
PI	3.14159.....
RAD( )	Converts degrees to radians
RND	Generates a random no from 1 to &FFFFFFFF (4,294,967,295)
RND(1)	Generates a random no from 0 to 1
RND(no)	Generates a random no from 1 to no
SGN( )	Gives +1 if +ve , 0 if 0 , or -1 if -ve
SIN( )	Sine
SQR( )	Square root
TAN( )	Tangent
VAL( )	Value of a string or a letter field.

This set give letter field results.

LEFT$(fld,N)	Left most N characters of fld
RIGHT$(fld,N)	Right most N characters of fld
MID$(fld,N)	All characters from position N onward
MID$(fld,N,M)	M characters from position N onward
STRING$(N,"chars")	N copies of the characters chars
CHR$(fld)	ASCII character whose code is held in fld

DATA FILE FORMATS

In this section we describe details of the Infomate data file structures. This will be followed by some of the alternative formats allowed for, including that for Lotus 123, Microsoft Excel, and of course Pipedream.

The Infomate File Format

Because Infomate needs to Interface with many different data manipulation programs, and to allow it to access any of its own data quickly and in any order, the Infomate authors chose a simple fixed record format that has the simplest form to convert to other formats.

In the figure below we have shown a typical Infomate file. Note the presence of "missing" values in all of the examples, and the use of implied decimals internally in the file

The schema file definition of the Infomate file might appear like this

```
ID
          TAG                 N3
          COLOUR              L1
DATA
          Breed               L2
          Condition           N1
          Old_weight          N31
          New_weight          N4.1
          Wool_Wt             N30
          Wool_Len            N3.0
```

An Infomate Data File created by the above could look like this.

```
  4GRM624524.512080.c/r
 14R R4352     9724.c/r
1240SF943243.2   12.c/r
854G  712712.7134   c/r
899R R    34.910117.c/r
998GRM6 99 9.9   23.c/r
```

Broadly, the Infomate File Format is defined by the schema file in terms of field widths and types. In addition to this the following should be noted.

1. All characters in the file are the ASCII printable characters from ! to ~ plus the space and carriage return ($OD)

2. All records are the same length and are terminated by a carriage return. No line feed is included.

3. There are no TAB characters or other record separators, each field being in the strict position indicated by the schema file.

4. All fields with less than a full field width are filled with spaces from the left.

5. Empty fields are filled with spaces.

6. There is no end-of-file mark, the carriage return on the final record being the last character in the file.

7. Leading zeros are normally converted to blanks, but are acceptable to Infomate.

In many cases the Infomate File Format can be used directly by the data analysis packages, or once simple editing tasks have been performed on them using the word processors on the target computer. Techniques to do this were described earlier.

The Pipedream and Excel File Format

The Infomate file above can be converted to Excel and Pipedream plain text format using the file conversion module described earlier. The format of these two files is shown below:

```
   4  ^  G  ^  RM  ^  6  ^  24.5   ^  24.5  ^  120  ^ 80. c/r
  14  ^  R  ^  _R  ^  4  ^  35.2   ^     _  ^   97  ^ 24. c/r
 124  ^  O  ^  SF  ^  9  ^  43.2   ^  43.2  ^    _  ^ 12. c/r
 854  ^  G  ^  __  ^  7  ^  12.7   ^  12.7  ^  134  ^  _  c/r
 899  ^  R  ^  _R  ^  _  ^     _   ^  34.9  ^  101  ^ 17. c/r
 998  ^  G  ^  RM  ^  6  ^   9.9   ^   9.9  ^    _  ^ 23. c/r
```

You should note that in the above format the space characters are shown as _ symbols and the ^ symbol is used to show the TABs. Each line is terminated by a carriage return. A full description of the formats follows.

1. All Fields are separated by a TAB character.

2. Each record is terminated by a carriage return.

3. Leading blanks and zeros in Numeric fields are eliminated.

4. Missing numeric fields contain a single space character.

5. Letter fields are transferred in their entirety. Blanks are not suppressed.

6. There is no end of file mark.

7. Implied decimal points are re-inserted into the file.

Lotus 123 Format

The Infomate file above can be converted to Lotus 123 /PRINT format using the file conversion module described earlier. The format of these files is shown below:

```
4      ^"G"    ^"RM"    ^   6    ^24.5  ^24.5  ^120   ^ 80.c/r  l/f
14     ^"R"    ^"_R"    ^   4    ^35.2  ^ "_"  ^ 97   ^ 24.c/r  l/f
124    ^"O"    ^"SF"    ^   9    ^43.2  ^43.2  ^"_"   ^ 12.c/r  l/f
854    ^"G"    ^"__"    ^   7    ^12.7  ^12.7  ^134   ^ "_"c/r  l/f
899    ^"R"    ^"_R"    ^"_"     ^ "_"  ^34.9  ^101   ^ 17.c/r  l/f
998    ^"G"    ^"RM"    ^   6    ^ 9.9  ^ 9.9  ^"_"   ^ 23.c/r  l/f
```

When Files are converted to Lotus /PRINT form they take on the following format.

1. All fields are separated by a TAB character.
2. Each record is terminated by a carriage return, followed by a linefeed character.
3. Leading blanks and zeros in numeric fields are eliminated.
4. Letter fields are transferred in their entirety and are surrounded by double quote marks (e.g. "DEAD")
5. Missing numeric fields contain a single space surrounded by double quote marks(" ").
6. There is no end of file mark.
7. Implied decimal points are re-inserted into the file.

SCALES AND OTHER DEVICES

Infomate provides much flexibility in the area of interfacing to weighing equipment and other electronic devices. These are provided by a wide range of manufacturers and all vary considerably in the details of the way that they interface to different computers. To handle this problem Infomate uses specific drivers for each piece of equipment. A driver will handle all of the peculiarities of the device and make it look standard to the main part of the Infomate program. Drivers can be added to the set included here with not too much difficulty, but must be developed by Infomate engineers. If you have specific devices that are not in this set provide as much information about the interfacing specifications as you can to us and we will consider if it can be incorporated, and the cost of doing that. Specialty equipment that has limited market potential may have to be interfaced on a full cost recovery basis, but more commonly used devices may be handled on a part cost recovery basis.

In the following pages the devices currently supported by Infomate are described, including the details of their schema file usage and cabling and operation details. The current list is shown below and it will grow with Infomate.

ACTRONIC_1
ALLFLEX_1
AND_1
DONALD_1
MAFKCI_1
METTLER_1
METTLER_1/MAFKCI_1
METTLER_2
METTLER_3
SARTORIUS_1
SARTORIUS_1/MAFKCI_1
TOLEDO_1
TRU-TEST_1

Driver Name: ACTRONIC_1

Description: Interfaces with the Actronic ADAM (Analogue Data Acquisition Module). This is a portable multi-channel analogue to digital converter with an additional low speed event counter. It has 7 user addressable channels each with input range of +/-2V and resolution of +/-20,000 counts.

With Infomate's time functions and the Z88 alarm function a simple data logger can be implemented.

A customised version of the ADAM interface called InfoLogger that has been optimised for data logging on the Z88 is available in a metal die-cast box. This unit has circuitry to allow the Z88 and the ADAM interface to be powered by an external 12 volt battery.

Driver for: Actronic AS549, and InfoLogger.
Note: The AS549 is a special edition of the AS456 ADAM with a 9600 baud option and modification to its output strings to include a carriage return as a message terminator.

Tested with: Actronic AS549.

Manufacturer: Actronic Systems Ltd.
P.O.Box 9341
Auckland, N.Z.

N.Z. agents: As above.

Default Z88 Port: 9600 Baud, Space Parity, No Xon/Xoff.

Equipment Set-up: See ADAM user manual.

Instructions: The ADAM is a multi-channel device. Its channels need to be initialised. This is done with the PORTIS statement. Channels are initialised using the letter "I" followed by a list of digits for the channels to be used e.g.:

```
PORTIS   ACTRONIC_1      I1245
```

initialises channels 1,2,4 and 5.

The event counter can also be reset to zero using ¦* in the PORTIS statement (the ¦ is the split vertical bar on the Z88 keyboard and inserts a <CR><LF> in the string which is sent to the ADAM). e.g.:

```
PORTIS   ACTRONIC_1  I23¦*
```

intialises channels 2 and 3 and resets the counter.

In the sequence part of the schema each channel needs to be specified before a reading is taken. This is done using the SETPORT command e.g.:

```
SETPORT  5
GETPORT  Voltage   reading  volts!
```

fetches a reading from channel 5. Using a "P" in the SETPORT allows the counter value to be read e.g.:

```
SETPORT  P
GETPORT  Counter reading  counts
```

Using the letters A to G with SETPORT fetches the averaged analogue data (averaged over the last 4 conversions) for channels 1 to 7 respectively.

Using an "*" (asterisk) with SETPORT resets the counter.

Note the warning in the reference section about use of SETPORT / GETPORT pairs when backstepping through a schema.

Data Logging

Set up the ADAM and connect to any sensors required.

To synchronise the readings with the Z88 clock, put the Z88 to sleep using a SLEEP command in the schema file. The Z88 alarm application can then be arranged to wake up the Z88 to take a reading at specified intervals. The ADAM must remain powered up.

To set the alarm, press □ A, choose SET ALARM and use the arrows to set up the options. They should be set as follows. **Bell** to Off (to avoid double readings), **Alarm Type** to Alarm, **Repeat Every** to the value required, and **No. of Times** to Forever. Use ⇧ and ⇩ to cycle through the options available.

The second schema in the examples following is one designed for data logging. Note the options used with the ENTER line: ENTER=<+ ensures that each time the Z88 wakes up it continues with the ID incremented automatically. When you run it you should initially enter the starting ID and the Z88 will go to sleep immediately. The first reading will be taken on the first alarm. To confirm that everything will work correctly when the Z88 wakes up on the first alarm you can wake the Z88 manually with the two SHIFT keys. The readings should flash on the screen and the Z88 fall back to sleep. You may want to discard this reading when processing your data since it will be out of step with all the alarm generated readings.

To stop logging you need to wake the Z88 with the two SHIFT keys and press ESC before the Z88 has finished taking the current set of readings. Note: Pressing ESC when the Z88 has been woken up by the alarm will have no effect because the alarm wakes the Z88 in the "locked out" state which means key-presses are not registered until the Z88 has been switched off and on again.

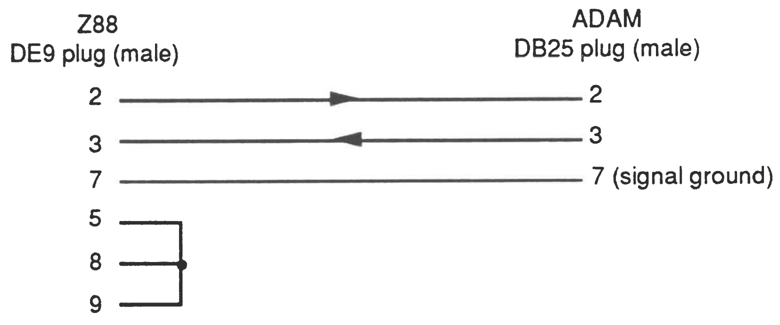
Schema Example:

```
TITLE     ACTRONIC_1  TEST  SCHEMA
FILE      VOLTS.DAT
PORTIS    ACTRONIC_1  I23¦*
ID
          Read_No            N2
DATA
          Voltage_2          N7.2
          Voltage_3          N7.2
          Count              N6
SEQUENCE
          ENTER?<+           Read_No   Type  Read  No.
          MAKERECORD
          SETPORT  2
          GETPORT       Voltage_2      Reading  chan_2
          SETPORT  3
          GETPORT       Voltage_3      Reading  chan_3
          SETPORT  P
          GETPORT       Count          Getting  count
END
```

A schema for data logging.

```
TITLE     ACTRONIC_1  TEST  SCHEMA
FILE      VOLTS.DAT
PORTIS    ACTRONIC_1  I23¦*
ID
      Read_No          N2
DATA
      Rec_Time        N6
      Voltage_2       N7.2
      Voltage_3       N7.2
      Count           N6
SEQUENCE
      ENTER=<+        Read_No        Type  Read  No.
      SLEEP
      MAKERECORD
      LET             Rec_Time  =  TIME(HMS)
      SETPORT  2
      GETPORT         Voltage_2      Reading  chan_2
      SETPORT  3
      GETPORT         Voltage_3      Reading  chan_3
      SETPORT  P
      GETPORT         Count          Getting  count
END
```

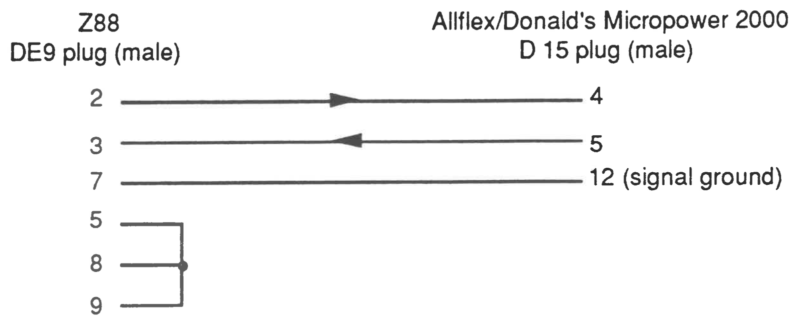
Cable Diagram:



Driver Name: ALLFLEX_1

Description:	Interfaces Infomate with Micropower 2000 series animal weighing scales. These scales were formerly manufactured by Donald Presses but are now made by Allflex.
Driver for:	Micropower 2000 series scales.
Tested with:	Donalds Micropower 2000-XP.
Manufacturer:	Allflex New Zealand Ltd. Private Bag Palmerston North, N.Z.
N.Z. agents:	As above.
Default Z88 Port:	1200 Baud, Space Parity, No Xon/Xoff.
Equipment Set-up:	Modify older models as per special note on next page. Set up the Micropower so that weights are displayed. To achieve this press "MODE" then "4" then "MODE". Set up in "on line" mode. To achieve this press "*", then "MODE" as many times as needed to get the "O-L 00" display and then press "*" to get back to the weigh mode.
Instructions:	Connect up scales and Run your schema. Note: Unfortunately the scales will not respond if the weight is zero or negative so make sure there is something on your scales when you run you schema.
Schema Example:	See the METTLER_1 driver example. Make sure the driver name is spelt correctly.

Cable Diagram:



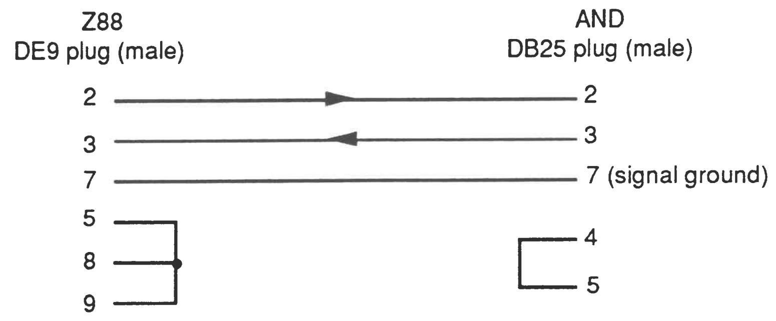
Special Note: Older models (and possibly current ones) need a separate ground wired into the 15 way D socket. This wire was omitted by the manufacturer in spite of being specified by the scale's designer. This modification should be done by someone handy with a soldering iron or else by the manufacturer.

To gain access to the inside of the Micropower 2000 you need to lift the rubber at the edges of the perspex front panel and undo the screws. Pin 12 of the 15 way D communications connector must be connected to the digital ground. This is pin 2 of the 14 way molex connector on the printed circuit board. To ascertain numbering use the fact that the red and white wires on that same molex connector are pins 3 and 4 respectively. Make sure you're not connecting to mains ground, and do not under any circumstances perform this modification yourself if you have any doubts about it at all.

Driver Name: AND_1

Description:	Interfaces Infomate with a series of scales manufactured by A&D Corporation. It relies on these scales sending out a continuous stream of readings through their serial port.
Driver for:	A&D models EW, FX, FY, ER, EP, EK (according to information from the data sheets available).
Tested with:	AND EW-3000B.
Manufacturer:	A&D Company, Ltd Shintaiso Building, No. 5-1052 10-7 Dogenzaka 2-Chome Shibuya-ku Tokyo 150 Japan.
N.Z. agents:	E.C. Gough Ltd, P.O. Box 22073, Christchurch, NZ.
Default Z88 Port:	2400 Baud, Space Parity, No Xon/Xoff.
Equipment Set-up:	Older style scales do not have any means of altering the serial interface set up. If the baud rate is different from the default rate of 2400 baud you will have to change the baudrate using your PORTIS statement e.g. `PORTIS   AND_1[1200SN]` More modern scales e.g. FX/FY series have "Software Parameters" which can be set by the user (consult the scale instruction manual).
Instructions:	Connect up scales and Run your schema.
Schema Example:	See METTLER_1 sample schema.

Cable Diagram:



(according to the data sheets available this particular wiring scheme will work with most AND scales)

Driver Name: DONALD_1

Description:	Identical to ALLFLEX_1 driver. Interfaces Infomate with Micropower 2000 series animal weighing scales. These scales were formerly manufactured by Donald Presses but are now made by Allflex.
Driver for:	Micropower 2000 series scales
Tested with:	Donalds Micropower 2000-XP.
Manufacturer:	These scales are no longer manufactured by Donald Presses but by: Allflex New Zealand Ltd. Private Bag Palmerston North, N.Z.
N.Z. agents:	As above.
Default Z88 Port:	1200 Baud, Space Parity, No Xon/Xoff.
Equipment Set-up:	Modify older models as per special note below. Set up the Micropower so that weights are displayed. To achieve this press "MODE" then "4" then "MODE". Set up in "on line" mode. To achieve this press "*", then "MODE" as many times as needed to get the "O-L 00" display and then press "*" to get back to weigh mode.
Instructions:	Connect up scales and Run your schema. Note: Unfortunately the scales will not respond if the weight is zero or negative so make sure there is something on your scales when you run you schema.
Schema Example:	See the METTLER_1 driver example. Make sure the driver name is spelt correctly.
Cable Diagram:	See ALLFLEX_1 driver.
Special Note:	These scales need a separate ground wired into the 15 way D socket. This wire was omitted by the manufacturer in spite of being specified by the scale's designer. This modification should be done by someone handy with a soldering iron or else by the manufacturer (see ALLFLEX_1 driver for details).

Driver Name: MAFKCI_1

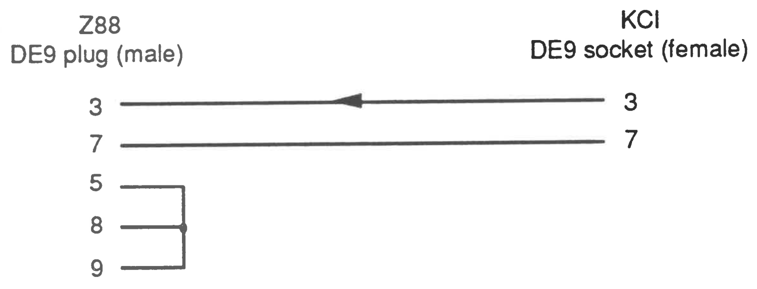
Description:	This driver enables length measurements to be captured using Mitutoyo or PAV electronic calipers. The calipers are connected to the Z88 via the KCI interface produced by MAF Kerikeri.
Driver for:	KCI caliper interface with Mitutoyo Digimatic Series 500, 550, 551 and some PAV Electronic calipers.
Tested with:	KCI and Mitutoyo 500-322.
Manufacturer:	KCI: Environmental Physics Kerikeri Horticultural Research Station, P.O. Box 23, Kerikeri, New Zealand. Calipers: Mitutoyo Corporation Japan. (they only deal through the local agents). PAV Prazisions-Apparatebau Ag Schaanerstrasse 40 FL-9490 Vaduz Liechtenstein.
N.Z. agents:	KCI: as above. Calipers: Metrology Techniques P.O. Box 10-024 Hamilton N.Z.
Default Z88 Port:	9600 Baud, Space Parity, No Xon/Xoff.
Equipment Set-up:	Plug in cables and turn on caliper interface (remember to turn it off after use).
Instructions:	Run your schema file. You are prompted to press the Data/Hold button to initialise the interface. If something is not plugged in or the KCI has a flat battery the Z88 will wait indefinitely for an answer from the interface. If you can't rectify the fault just press the ESC key. Note: If you turn on the KCI after you've tried to Run the schema you may get a "Could Not Initialise ?.." message. That's because turning on the KCI causes a spurious character to be received by the Z88. Just Run the schema again and it should work the second time.

During sequence operation press the Data/Hold button whenever a length measurement is required from the calipers. A message prompting the user to press the button should be included in the schema file as shown in the following sample schema.

Schema Example:

```
TITLE    MAFKCI_1  TEST  SCHEMA
FILE     LENGTHS.DAT
PORTIS   MAFKCI_1
ID
   Piece_No    N2
DATA
   Length      N7.2
SEQUENCE
   ENTER=<+    Piece_No   Type Piece No.
   MAKERECORD
   GETPORT     Length     Press  Hold!
END
```

Cable Diagram:



Driver Name: METTLER_1

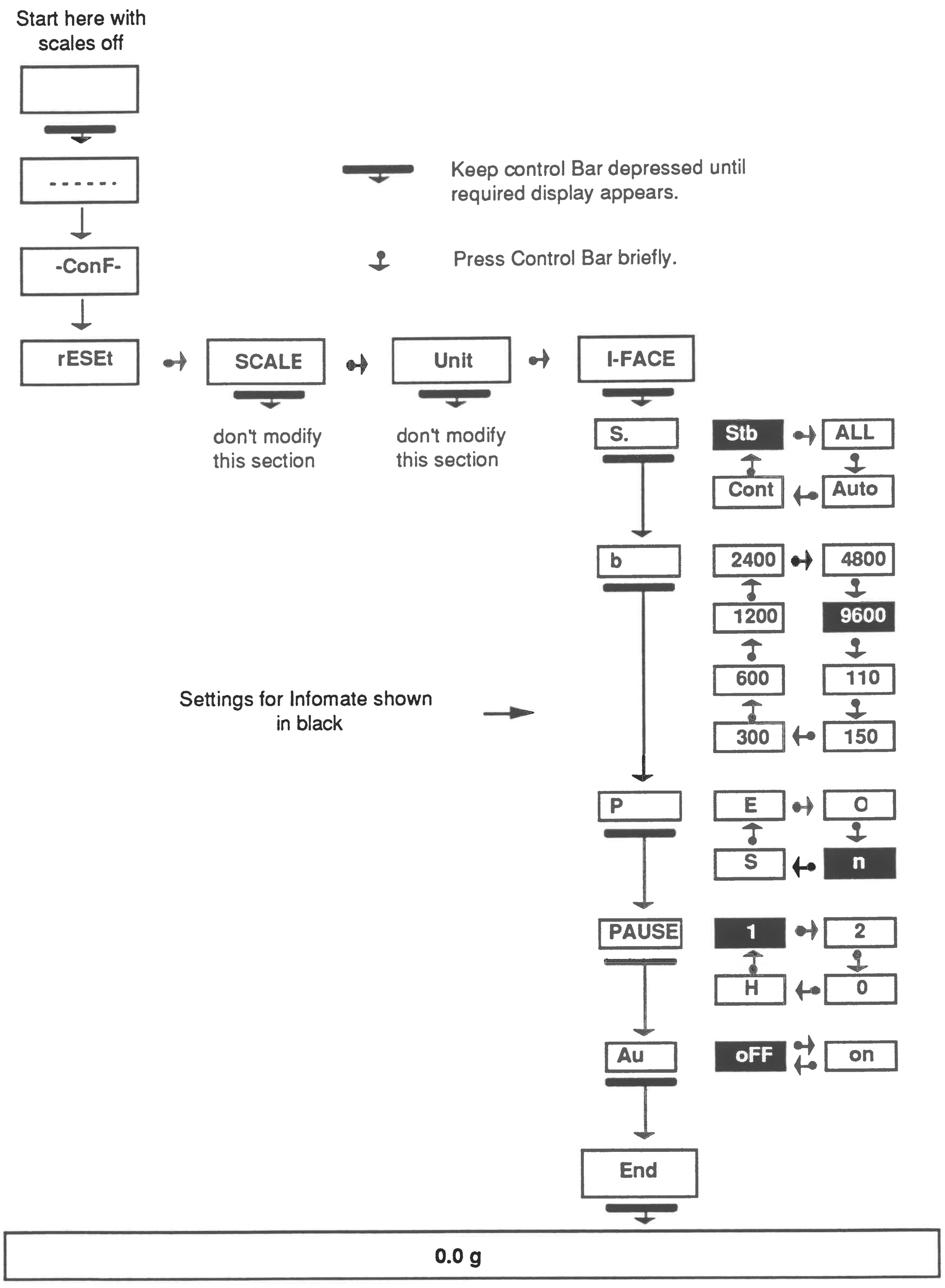
Description:	Interfaces Infomate with Mettler scales. This driver is for the bulk of the modern Mettler balances with RS232 interfaces. That is for those Mettler scales which respond to the S <CR><LF> (Send steady weight) command. There are other Mettler drivers for some of the older types.
Driver for:	PM, SM, AM, AT and BASBAL scales.
Tested with:	Mettler PM6000.
Manufacturer:	Mettler Instrumente AG CH - 8606 Greifensee, Switzerland.
N.Z. agents:	Watson Victor Ltd. P.O. Box 1216, Auckland 1 NZ.
Default Z88 Port:	9600 Baud, Space Parity, No Xon/Xoff.

Equipment Set-up: The Mettler needs to have its configuration file changed from the default factory settings in order to work with this driver. Set the Interface parameters as follows:

S. Data transfer mode to Stb
b. Baudrate to 9600
P. Parity to n
PAUSE to 1
AU to oFF (if an option on your scales)

To achieve this you should follow your scale's instruction manual. A summary of the process is provided in the diagram following (there are slight differences between models but the principle is the same). With the scales off hold down the control bar (sometimes called the tare or zero key/bar) until the word -Conf- is displayed. Release the control bar and give it a few short presses. You are now cycling through the main sections of the configuration file. Stop when you see the word I-FACE-. Now hold down the control bar until s. appears. You are now setting up the data transfer mode. Cycle round the options in the s mode with short presses of the control bar until Stb shows. You are now ready to move on to set the next parameter. To do this hold down the control bar for a long time until b. appears. Cycle through the baudrate options with short key presses. Set the other parameters likewise. When you get to the "End" display give the control bar one last long press and you will be back in weigh mode.

Setting Configuration file for Scales
for use with METTLER_1 Driver


Start here with scales off
- - - - - -
-ConF-
rESEt
SCALE
Unit
I-FACE
don't modify this section
don't modify this section
Keep control Bar depressed until required display appears.
Press Control Bar briefly.
S.
Stb
ALL
Cont
Auto
b
2400
4800
1200
9600
600
110
300
150
Settings for Infomate shown in black
P
E
O
S
n
PAUSE
1
2
H
0
Au
oFF
on
End
0.0 g

Instructions: Connect your scales to the Z88 with the cable provided and Run your schema file.

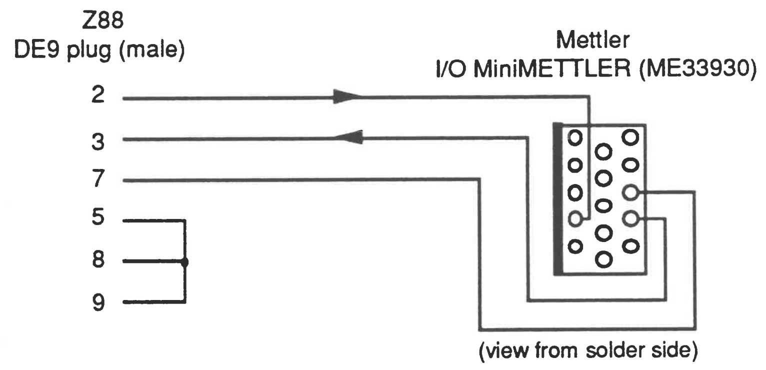
Schema Example:

```
TITLE     METTLER_1  TEST  SCHEMA
FILE      WEIGHTS.DAT
PORTIS    METTLER_1
ID
   Piece_No        N2
DATA
   Weight          N7.2
SEQUENCE
   ENTER?<+        Piece_No        Type  Piece  No.
   MAKERECORD
   GETPORT         Length          Reading  weight!
END
```

Special Notes: If you have documentation of the Mettler interface you will see that you can send various commands to the scales. You could set up the units on the scale using the U command within your PORTIS statement. e.g.. to set the units to ounces use:

```
PORTIS  METTLER_1  Uoz
```

Cable Diagram:

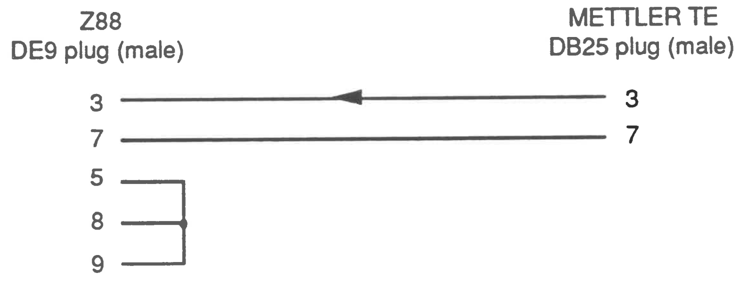


Driver Name: METTLER_2

Description:	Interfaces with Mettler TE scales only. It has a special feature to allow unsteady weights to be recorded.
Driver for:	Mettler TE series scales with Mettler 030 Data output option.
Tested with:	Mettler TE 60.
Manufacturer:	See METTLER_1 driver.
N.Z. agents:	See METTLER_1 driver.
Default Z88 Port:	2400 Baud, Space Parity, No Xon/Xoff.
Equipment Set-up:	Set up the scale in the "Send Continuous Mode".
Instructions:	Connect your scales to the Z88 with the cable provided and Run your schema file. The driver will only enable steady readings to be captured by Infomate unless you set the dynamic reading flag as shown in the schema example following.
Schema Example:	As for METTLER_1 example but with METTLER_2 in the PORTIS statement. To allow unsteady (dynamic) readings to be recorded the PORTIS statement should be:

```
PORTIS  METTLER_2  {D}
```

Cable Diagram:



Driver Name: METTLER_3

Description:	Interfaces with many of the older style Mettler scales with limited RS232 interface capabilities. This driver requires that the scales are in a "continuous print" output mode.
Driver for:	Mettler AE, AC, PC & PE series with RS232 interface modules/options.
Tested with:	PE3600 and AE160.
Manufacturer:	See METTLER_1 driver.
N.Z. agents:	See METTLER_1 driver.
Default Z88 Port:	2400 Baud, Space Parity, No Xon/Xoff.
Equipment Set-up:	Set your scale up in "continuous print mode" (see your scale manual). It is probably in the correct mode already.
Instructions:	Connect your scales to the Z88 with the cable provided and Run your schema file.
Schema Example:	As for METTLER_1 example but with METTLER_3 in the PORTIS statement.
Cable Diagram:	These vary but if the socket on the Mettler is square the wiring should be the same as in the METTLER_1 driver diagram. If it is a 25 pin D connector it will probably be the same as in the METTLER_2 driver.

Driver Name: METTLER_1/MAFKCI_1

Description: A combination of the METTLER_1 and MAFKCI_1 drivers. It enables length and weight measurements to be captured by Infomate using the single serial port. A Mettler Balance and a pair of electronic calipers (via the KCI Caliper interface) are connected to the Z88 with a "port splitter" cable.

Driver for: MAF Kerikeri KCI caliper interface in conjunction with any of the Mettler scales compatible with the METTLER_1 driver. i.e. Mettler PM, SM, AM, AT and BASBAL.

Tested with: KCI and Mettler PM6000.

Manufacturer: See METTLER_1 and MAFKCI_1 drivers.

N.Z. agents: See METTLER_1 and MAFKCI_1 drivers.

Default Z88 Port: 9600 Baud, Space Parity, No Xon/Xoff.

Equipment Set-up: Configure the Mettler scale as described in METTLER_1 driver. The KCI should be set to 9600 baud (you shouldn't have to alter the KCI baudrate because it is supplied set to 9600 baud).

Instructions: Connect up the KCI and scales to the "Port Splitter" cable. The cable is symmetrical so either socket will do.

Since there are two devices, the schema file has to identify the relevant device before getting data. This is done with a SETPORT command before your GETPORT. A single letter, K for calipers or M for Mettler is included after SETPORT. For example if a reading is to be taken from the calipers the schema file would have:

```
SETPORT   K
GETPORT   length      press  hold!
```

Remember to press the Data/Hold button for caliper readings.

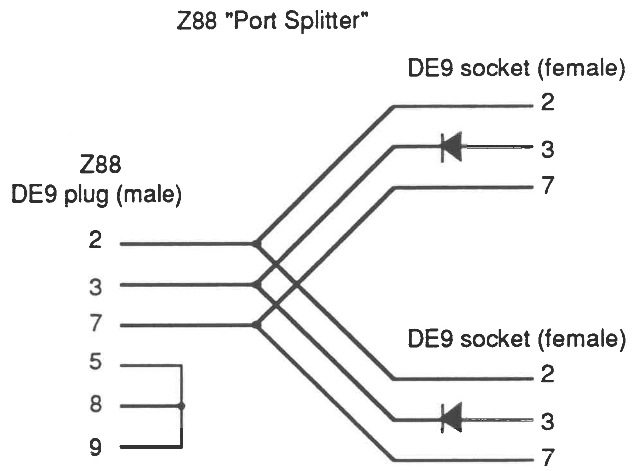
Note the warning in the reference section about use of SETPORT / GETPORT pairs when backstepping through a schema.

When running the schema, the port is initialised and both the Mettler and KCI connections are tested. To test the KCI connection you are asked to press Data/Hold to send a test reading.

Schema Example:

```
TITLE     METTLER_1/KCI_1  TEST  SCHEMA
FILE      PODS.DAT
PORTIS    METTLER_1/MAFKCI_1
ID
   Pod_No           N2
DATA
   Weight           N7.2
   Length           N7.2
SEQUENCE
   ENTER?<+         Pod_No           Type  Pod  No.
   MAKERECORD
   SETPORT  M
   GETPORT          Weight           Getting  weight!
   SETPORT  K
   GETPORT          Length           Press  Hold
END
```

Cable Diagram:



Mettler cable as described in METTLER_1 driver.

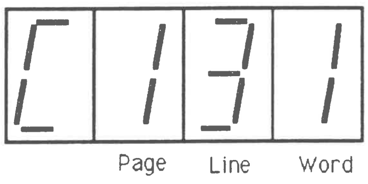
KCI cable as described in MAFKCI_1 driver.

Driver Name: SARTORIUS_1

Description:	Interfaces to Sartorius Scales with the MP8 series and the new MC 1 series interfaces. N.B. It may be difficult to ascertain what type of Sartorius scale you have. The MP8-X number doesn't refer to the model number but the microprocessor. The label on the back with the serial number may help.
Driver for:	As above.
Tested with:	Sartorius 1702 and LC620P.
Manufacturer:	Sartorius AG P.O.Box 3243 3400 Goettingen Germany.
N.Z. agents:	Wilton Instruments (a Salmond Smith Biolab Group) P.O. Box 31044 Lower Hutt N.Z.
Default Z88 Port:	9600 Baud, Space Parity, No Xon/Xoff.
Equipment Set-up:	MP8 series: Refer to the scale's manual for exact details. The Balance Operating Program needs to be modified as follows: Code: Data output: C211 external print command without stability Code: Baudrate: C227 9600 Baud Code: Parity Bit C232 Space Parity

To access the balance operating program:
With the balance turned of (Standby state) hold down the tare control while momentarily pressing the ON/OFF key. Upon completion of the automatic self-test, release the tare control before "CH5" is displayed. The status of the balance operating program will be indicated in the weight display. "L" stands for the list mode. In this mode, you can check the code settings, but you cannot program new codes. To change a program code you must first unlock the menu access switch to access the menu. Your scales manual will tell you how to do this. For most models this involves finding the appropriate switch cover, removing it and flicking the switch. Some models require a special plug which is inserted in the serial port which unlocks the scale program. This plug is just a 25 pin D

connector with pin 24 connected to pin 21 and pin 19 connected to pin 7. For these models it would be a good idea to change the code to "Program lock OFF" (411) using the method described below. Having unlocked the program the display will now indicate "C" which stands for the change mode, meaning you can now change codes.



After the balance operating program has been called, the display will show a continuous numerical sequence from 0-5 representing the "page" selection. You want page 2 "Data Output". When the 2 appears press the tare control. You are now cycling through the "lines" of the data output page. Press the tare key when the line number you want to change is displayed. Having done this, the numbers for the "word" will be displayed. A small triangle or circle indicates the current setting. If you need to change it, press the tare key when the code you want is displayed. Repeat the procedure for each line which needs to be changed. To return to the weighing program press the tare control each time a 0 appears in the numerical sequence (word, line, page)

MC1 Series
The Menu Code needs to be modified as follows:
Code: Data output:
C611 Print on request regardless of stability.
Code: Baudrate:
C517 9600 Baud
Code: Parity Bit:
C522 Space Parity

To change the Menu Code:
- Turn the balance or scale off and then back on again.
While all segments are displayed, briefly press the tare control.
- If "-L-" is displayed, change to the "-C-" mode using the menu access switch
Select the desired code number as follows:
- press F1 to increase it or
- F2 to decrease it.

Select the left, middle or right place as follows:
- press the key to go toward the left
- press the key to go toward the right

To confirm your code selection press the control labelled **T**.
Adjust the menu access switch back to the original setting - readout: - L-.
To leave the menu, press **CF**

Instructions: Connect your scales to the Z88 with the cable provided and Run your schema file.

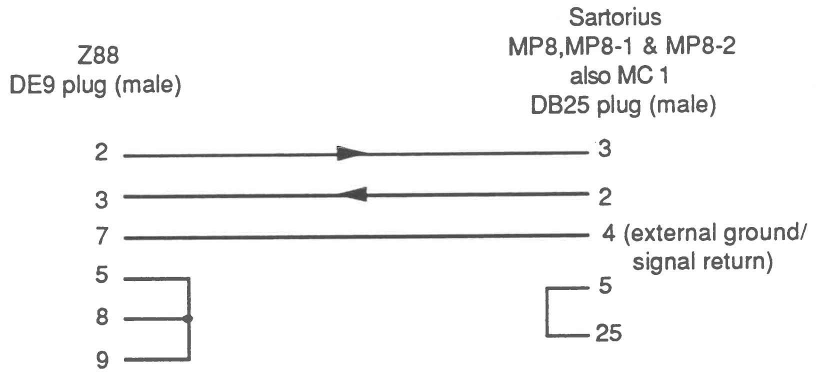
Schema Example: See the example for the METTLER_1 driver.

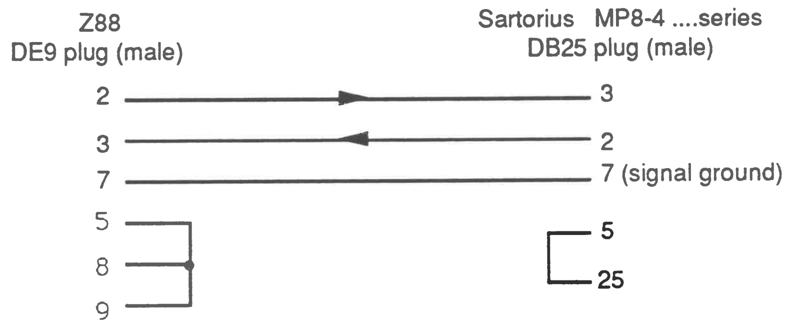
There is a facility to send single letter commands to the balance using either the initialising PORTIS line or SETPORT in your schema e.g.. to initialise your Sartorius to weigh in Carats you would use this statement:

```
PORTIS    SARTORIUS_1  C
```

(see your Sartorius manual for other commands)

Cable Diagram: Different scales have different signal grounds make sure you have the correct cable.





Driver Name: SARTORIUS_1/MAFKCI_1

Description:	A combination of the SARTORIUS_1 and MAFKCI_1 drivers. It enables length and weight measurements to be captured by Infomate using the single serial port. A Sartorius Balance and a pair of electronic calipers (via the KCI caliper interface) are connected to the Z88 with a "port splitter" cable.
Driver for:	MAF Kerikeri KCI caliper interface in conjunction with any of the Sartorius scales compatible with the SARTORIUS_1 driver.
Tested with:	Sartorius 1702 and Mitutoyo 500-322.
Manufacturer:	KCI see MAFKCI_1 driver. Sartorius see SARTORIUS_1 driver.
N.Z. agents:	As above.
Default Z88 Port:	9600 Baud, Space Parity, No Xon/Xoff.
Equipment Set-up:	See MAFKCI_1 and SARTORIUS_1 drivers.

Instructions:

Connect up the KCI and scales to the "Port Splitter" cable. The cable is symmetrical so either socket will do. Since there are two devices, the schema file has to identify the relevant device before getting data. This is done with a SETPORT before your GETPORT. A single letter, K for calipers or S for Sartorius is included after SETPORT. For example if a reading is to be taken from the calipers the schema file would have:

```
SETPORT  K
GETPORT  length    press  hold!
```

Remember to press the Data/Hold button for caliper readings. When running the schema, the port is initialised and both the Sartorius and KCI connections are tested. To test the KCI connection you are required to press Data/Hold to send a test reading.

Note the warning in the reference section about use of SETPORT / GETPORT pairs when backstepping through a schema.

Schema Example:	See METTLER_1/MAFKCI_1 driver example.
Cable Diagram:	See METTLER_1/MAFKCI_1 driver for "port splitter" adapter. For Z88 to KCI see MAFKCI_1 driver For Z88 to Sartorius see SARTORIUS_1 driver.

Driver Name: TOLEDO_1

Description:	Interfaces Infomate to Toledo 1938 series weighing scales. These scales provide quite sophisticated features for parts counting and net or gross weight capture.
Driver for:	TOLEDO 1938.
Tested with:	Toledo 1938 - 0001.
Manufacturer:	Toledo Scale, Industrial Products 350 W. Wilson Bridge Rd. Worthington, Ohio 43085, USA.
N.Z. agents:	Watson Victor Ltd. P.O. Box 1216, Auckland 1 NZ.
Default Z88 Port:	9600 Baud, No Parity, No Xon/Xoff.

Equipment Set-up: To set up your scales you will need to access Set-up mode. First remove the top screw that secures the end cap on the right side of the display housing and loosen the lower screw so the end cap can be rotated out of the way allowing access to the inside of the enclosure. Then press the white "ON/OFF" key (located on the front below the right side of the display screen) to turn the scale power ON. After the power-up sequence is completed, press and release the white Set-up mode push-button located on the PCB inside the end of the display.

When the Set-up mode is accessed, the first series of soft-switches will be displayed as [10 0]. The "10" indicates the first series of switches (11 through 16). Pressing the "TARE" key will change the "0" (indicating OFF) to a "1" (indicating ON). When the "1" is displayed, press the "PRINT" key to accept the displayed setting and advance to the first soft-switch selection. If the "10" series of soft-switches is to be by-passed, leave the display showing [10 0] . and just press the " PRINT" key to advance to the "20" series of soft-switches. Pressing the "PRINT" key without making changes will advance to the next selection, and pressing the "ZERO" key will backup to the previous selection.

The five sections of soft-switches in the set-up mode have the following functions:

10	SCALE SETUP.
20	LEGAL FOR TRADE AND CALIBRATION SETUP.
30	COUNTING SETUP.
40	PRINTER OUTPUT SETUP.
50	PRINT FORMAT SETUP.

Presumably you have taken care of setting up soft-switch series 10, 20 and 30. Consult your scale manual if you need to inspect or modify any of those.

Before setting up your scales, you need to decide what data you need to send to Infomate from the scales. The Toledo 138 scales can give you Gross weight, Tare weight, Net weight, Piece count or a combination of these. The most common of these is the net weight. In this case you enable printing of the net weight and Infomate will get the number correctly. However, if you are going to get more than one number for each weighing, e.g.. Tare weight and Net weight you will have to work out how to extract the relevant statistics from the string returned from the scales using the BASIC string functions. A list of these formats is shown later.

Below is the definition of the "40" and "50" series soft-switches. The normal settings for INFOMATE are shown in bold

<u>40 PRINTER OUTPUT SETUP SOFT-SWITCHES</u>

0 = By-pass Section 40 soft-switches 41-43.
1 = Access Section 40 soft-switches 41-43.

41 **PRINTER BAUD RATE**

0 = 9600 Baud.
1 = 300 Baud.

42 **CHECKSUM**

0 = Do not send checksum.
1 = Send checksum.

43 **REMOTE INPUT**

0 = Remote Input is disabled.
1 = Remote Input is enabled.

<u>50 PRINT FORMAT SETUP SOFT-SWITCHES</u>

0 = By-pass Section 50 soft-switches 51-58.
1 = Access Section 50 soft-switches 51-58.

51 **SINGLE OR MULTI-LINE PRINTING**

0 = Print/send data on multiple lines.
1 = Print/send data on single lines. <- Must be set.

52 **PRINT GROSS WEIGHT**

0 = Do not print/send Gross Weight data.
1 = Print/send Gross Weight data. <- Optional setting.

53 **PRINT TARE WEIGHT**

0 = Do not print/send Tare Weight data.
1 = Print/send Tare Weight data. <- Optional setting.

54 **PRINT NET WEIGHT**
0 = Do not print/send Net Weight data.
1 = Print/send Net Weight data. <- Normal setting.
WARNING: See notes below about taring zero weights.

55 **PRINT DOUBLE WIDTH NET WEIGHT**
0 = Print Normal Width Net Weights.
1 = Print Double Width Net Weights. <- No point Setting.

56 **PRINT APW** (Average piece weight)
0 = Do not print/send APW data.
1 = Print/send APW data. <- Optional setting.

57 **PRINT PIECES** (To enable this, set soft-switches 31-34)
0 = Do not print/send Number of Pieces.
1 = Print/send Number of Pieces <- Optional setting.

58 **PRINT DOUBLE WIDTH PIECES**
0 = Print Normal Width Number of Pieces
1 = Print Double Width Num. of Pieces <- No point Setting.

Note: You are limited to 30 characters for the string returned to Infomate from the Toledo Scale so you cannot have all the options enabled. To help you work out what is possible, the format for each type of output is given below:

GROSS	Six digits (including decimal point), a space, and then the two letters "lb" or "kg"
TARE	Six digits (including decimal point), a space, followed by the two letters "lb" or "kg"
NET	Six digits (including decimal point), a space, followed by the two letters "lb" or "kg", a space and then the letters "NET".
APW	Eight digits (including decimal point), followed by two letters "lb" or "kg", space and then the three letters "APW".
COUNT	A one to six digit count followed by a space and then the three letters "PCS".

Also note that each double width selection (cf. switches 55 and 58), above adds two characters to the stream of output characters.

To exit set-up mode press the PRINT key until you get to 99 END OF SETUP MODE, and then press the Set-up push-button to return to normal run mode. If at any stage through the setting of the scales you wish to terminate the operation and keep the settings you have made so far, press the C key on the scales, and then press the Set-up push-button to return to normal run mode.

NOTE: There is a minor difficulty with the Toledo scales if it is to be used with an empty pan. i.e. without a container for the material being weighed. The scales cannot be tared to the zero value that appears when the scales are first powered on, and nor can weights be sent to the Z88 or any other computer unless a tare has been completed. Reasons for this are unclear, but to avoid it try one of these solutions. 1). Always use a container for your material to be weighed, 2). After powering up the Toledo, clip a strip of magnetic material to the front of the scales to raise the weight indication to a value greater than zero. Now tare the scales and weighing can continue. Whenever the scales must be switched on again, remove the magnetic strip before you turn on the scales. Alternatively, 3). turn the scales on while lifting the scales pan lightly with your finger. This creates a slight negative weight which will be zeroed out when it becomes steady. You must lift with a **steady** force that is **less than 2% of the scale capacity**. Practice makes this operation easier.

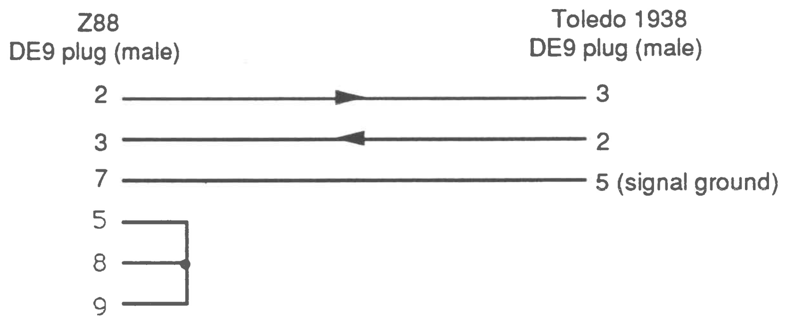
Instructions: Connect your scales to the Z88 with the cable provided and Run your schema file.

Schema Example: See the METTLER_1 driver example. Make sure the driver name is spelt correctly.

If you wish to extract a second weight e.g. APW from the string of characters returned from the scales you can use a MID$() sequence like this

```
GETPORT    MEMORY$          Reading  Scales
LET        Net_Weight   =   VAL(MEMORY$)
LET        APW  =           VAL(MID$(MEMORY$,14,8))
```

Cable Diagram:



Driver Name: TRU-TEST_1

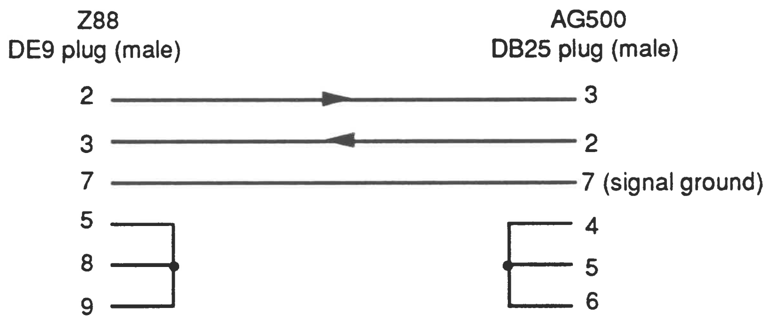
Description:	Interfaces Infomate to AG500 series animal weighing scales. All data capture is done on the Z88.
Driver for:	AG500-02 and AG500-03 versions 1, 2 & 3.
Tested with:	AG500-02 version 2, AG500-03 version 3.
Manufacturer:	TRU_TEST Distributors Ltd P.O. Box 51-078 Pakuranga Auckland, N.Z.
Default Z88 Port:	9600 Baud, Space Parity, No Xon/Xoff.
Equipment Set-up:	Press the "SET" and "Baud" keys and set to 9600 baud. If your scales have AWR (automatic weight recording) set this to OFF.

Instructions: Connect your scales to the Z88 with the cable provided and Run your schema. The driver has a feature to allow you to transmit the code of any key on the AG500 console using the SETPORT instruction. Your scale's manual gives the codes for the keys. For example by sending the code for the WGT key (RECORD key on version 3 models) the AG500 can be made to store the weights as well as Infomate (this has not been tried by the author). For example the RECORD key on an AG500-03 version 3 is at coordinates 3,9 (numbering starts at the top left hand corner with 0). So to get infomate to press this key "remotely" you would use:

```
SETPORT  39
```

Schema Example: See the METTLER_1 driver example. Make sure the driver name is spelt correctly.

Cable Diagram:

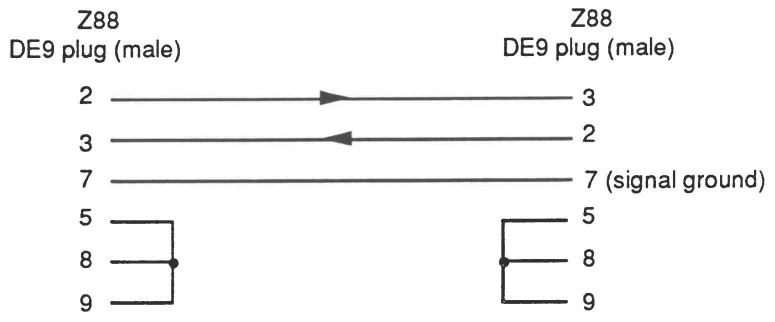


WIRING TO OTHER COMPUTERS

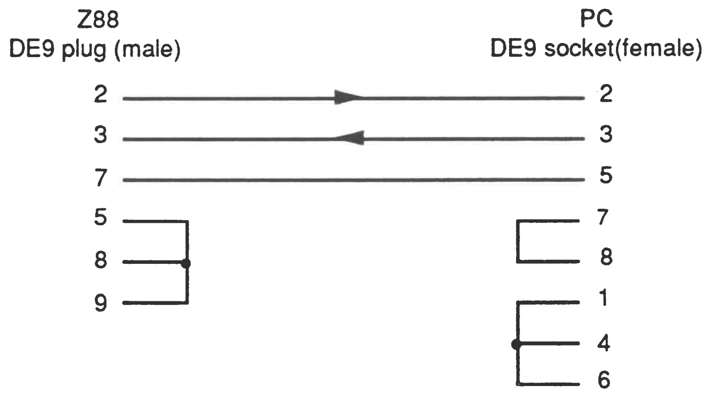
Below are descriptions of some of the cables required to link the Z88 to other computers.

Z88 to Z88 cable

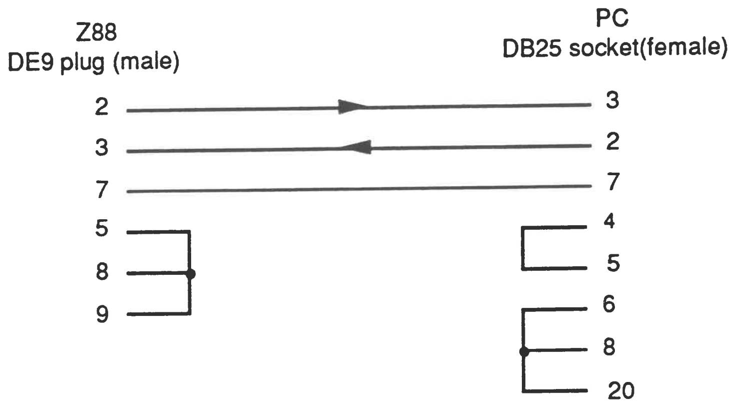
The following cable can be used to connect two Z88s together to allow files to be transferred from one to the other.



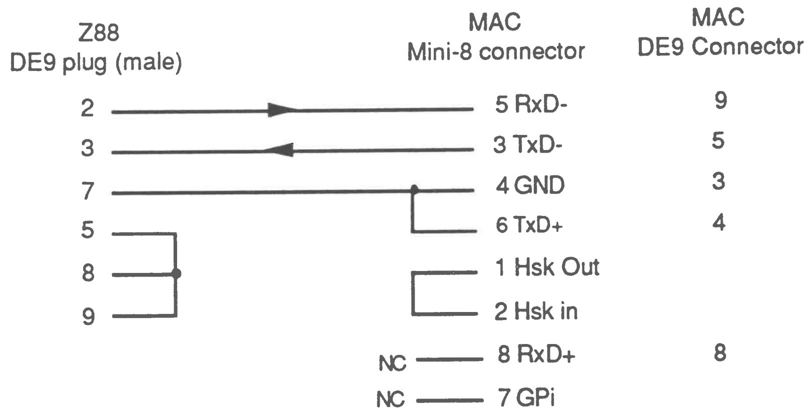
Z88 to PC AT cable

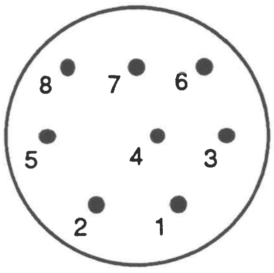


Z88 to PC XT cable



Z88 to Macintosh D connector and Mini-DIN cables





View from Solder Side

ERRORS AND THEIR POSSIBLE CAUSES.

The following is a list of errors that could occur at any stage of your interaction with Infomate. For user convenience they are listed in alphabetical order. They may have other messages appended to them such as " in line xxx", or in some cases may have other messages placed in front of them.

-ve root	You have tried to take the square root of a negative number in a LET or TEST statement.
... On Driver ...	A problem talking to the equipment on the port. Consult the driver notes at the end of the manual.
Accuracy Lost	Your LET or TEST statement has a trig function in it with a very large angle as an argument. Try subtracting a few multiples of 2*PI from the angles.
Already Done	The field specified in a CHECKDONE statement has a value in it other than a series of space characters implying that this record has already been updated. In an animal weight capture schema this would mean that the animal has already been weighed, or at least an animal you thought had that ear tag had been weighed.
Arguments	A function call (FN..) in a LET or TEST statement has too many or too few arguments. This can also be caused by bad bracketing .
Array	Your LET or TEST statement refers to a BASIC array of the form NAME(...) . This is not normal LET statement usage.
Bad Call	A LET or TEST statement is accessing a function incorrectly, or has the letters PROC imbedded when they are not a valid field name.
Bad File Name	You have specified a file name that is not valid for the Z88. For example FILE.DATA as a name contains to many characters after the dot.
Bad Format	A field format definition is not valid. Check with the start of the reference section for the allowed formats for each type of field (ID, DATA, or SCRATCHPAD).

Bad Groups	The string of characters in a CHECK statement implies that there are more fields than actually exist in your definition, or the number of * characters is not followed by complete groups of that number of characters. This could be a problem of the compiler reducing multiple blanks to a single blanks. Check the CHECK statement definition.
Bad HEX	A LET or TEST statement refers to an invalid hexadecimal number. Hex numbers must start with an & and comprise only the letters A to F and the digits 0 to 9.
Bad Parameters	The parity, baudrate or xon/xoff specifications in a PORTIS or LOG PRINTER statement does not conform to the accepted set of options. Check the definitions in the corresponding statement.
Bad Response	The equipment connected to the serial port did not respond as it should have. Is it connected correctly, and is it turned on. Perhaps you have specified the wrong driver.
Can't Open Data File	When Infomate has attempted to read a record from the data file, or write a new record to the data file, it has failed. Perhaps the file has been secretly deleted by the operator, or a move to a new device or directory has been made.
Check Failed	A CHECK statement looking at character patterns has failed to find the character string you entered in the list of allowed strings.
Could Not Write SCHEMA.CMP	The internal file SCHEMA.CMP could not be written onto the current device/ directory perhaps due to lack of space.
Could not Initialise ...	Infomate called a driver to initialise it and failed. Is the serial port connected to the scales or device, and is it switched on and in a correct operating mode. Consult the notes for the individual drivers at the end of the manual.
Data Fields Don't Match Schema	This probably means that the data file was not created by this schema, or for this schema. When the file is loaded it was found not to conform to the field specifications for the current schema. A common cause for this error is when fields are added or subtracted from a schema, or their length modified. When this happens you will need to create a new data file using the New Data File Create command from the main menu.
Division by Zero	Your LET or TEST statement has attempted to divide by zero.

Driver not Available	The driver for the serial port of the Z88 specified in the PORTIS statement does not exist in this version of Infomate. You may have an earlier version, or perhaps misspelled, or did not use capital letters in the driver name.
Driver not Specified	An attempt has been made to test a driver and either there is no schema file compiled at present, or the currently compiled schema does not contain a PORTIS statement.
End of Data	Not really an error. You have been using a NEXTRECORD to process the data in a file and the end of the file has been reached.
EPROM Full	The file that you wish to put on EPROM will not fit.
Exp Range	Your LET or TEST statement has referred to an EXP function and the value inside the brackets is greater than about 88. The result of this is too big for the Z88.
File > 2000 Records	Infomate can only handle up to 2000 records at this point in time.
File in Use	You are trying to use a file that another application is currently using. If there are no other applications pending, then an application has been killed without closing a file. In this situation the only option open to you is a soft reset. Sorry.
File Not Found	The data file specified does not exist. It may need to be loaded into the current directory, or if you are to use an empty file it may need to be created using the New File Create command from the main menu.
File Problem: Already Exists	This is an internal error for Infomate and should not occur. An attempt has been made to rename a file to a name used by another file.
File Problem: Bad Filename	You have specified a file name that is not valid for the Z88. For example the filename FILE.DATA has too many characters after the dot.
File Problem: Channel	This is an internal error for Infomate and should not occur. An attempt has been made to read from a non existent file.
File Problem: End of File	This is an internal error for Infomate and should not occur. An attempt has been made to read beyond the end of a file.

File Problem: File not Found	Infomate has tried to do something to a file that is now missing. You perhaps erased it while Infomate was not looking. A common culprit here is the internal file SCHEMA.CMP which is a compressed form of the current schema file.
File Problem: In Use	You are trying to use a file that another application is currently using. If there are no other applications pending, then an application has been killed without closing a file. In this situation the only option open to you is a soft reset. Sorry.
File SCHEMA.CMP is missing	The internal file that holds the compressed form of your schema has been deleted, or you have changed the device or directory that Infomate is currently running in. Check this by going into the Filer and try to find the current schema and SCHEMA.CMP files.
File Type Mismatch	You have probably specified a directory or device name rather than a filename.
ID Already Exists	Infomate requires that every data record has a unique ID to distinguish it from all other records. You have created a record with the same ID as one that already exists in the file. This may mean that your ID definition does not contain enough fields to provide truly unique ID's.
ID has been Changed	You have modified the data in an ID field after the record has been initialised by a MAKERECORD, FINDRECORD or NEXTRECORD. This means that the ID would have to go back into a different place in the file. This is not allowed by Infomate.
ID Not Found	Infomate has searched through the data file for the ID entered and cannot find it. Check spelling, digit transposition, or that you are using the right file.
ID Not Valid	You have entered an ID that does not conform to the schema file ID region definition. You have either put letters where numbers are expected or vice-versa.
ID Too Large	There are too many characters in an ID definition. The number of letters*2 plus the number of digits*3 must be less than 19.
Incorrect Log File Open	The currently open log file on EPROM has a different name to that specified in the current schema. You should first check that you have the right schema, and if OK close the EPROM log file using the EPROM Catalogue command from the main menu.

Infomate has unexpectedly Quit	This is an internal error and should not happen. We ask that you record the numbers that are displayed and communicate them too us as soon as possible so that we can correct the error and prevent others from the same fate that now besets you. Sorry.
Invalid Filename	The filename in a FILE or LOG statement of a schema contains a colon or slash character, or a LOG statement contains a dot. These characters are not allowed as they imply devices, directories or filename extensions are being used. Check with the specifications for these filenames under the FILE or LOG statement definitions.
Invalid Increment	The Increment value in a COUNTSIZE cannot be zero.
Invalid Statement	Infomate is expecting an ID, FILE, PORTIS, LOG or FILE statement. It may be misspelled, or use lower-case letters rather than capitals.
Invalid Weight	The weight read back from the serial port equipment is invalid. Is the baud-rate correct, and is the equipment configured in the right way. Is it connected properly.
LET Syntax	The LET statement must have a blank on either side of the = sign to work correctly.
LOG File still Open	You cannot save data to EPROM if an EPROM LOGFile remains open. You must first close the file by responding to the questions at the end of an EPROM Catalogue command from the main menu.
Log File Remains Open	You cannot save data to EPROM if an EPROM LOGFile remains open. You must first close the file by responding to the questions at the end of an EPROM Catalogue command from the main menu.
Log Range	You cannot take logs of numbers less than or equal to zero.

Looping at ...	You have backstepped with the up arrow to a REPEAT statement, and Infomate wants to know if you want to enter the REPEAT loop or skip over it. REPEAT UNTIL loops may be thought of as a single item, like an ENTER, and thus may be skipped over completely.
Message too Long	The response from the serial port equipment was longer than 30 characters, and this is not expected. It could be an error in the baud-rate. or in the configuration of the connected equipment.
Missing "	A string LET or TEST statement uses a string definition that is not terminated by a " character.
Missing)	A LET or TEST statement has a bracket missing.
Missing ,	A LET or TEST statement should have a comma, perhaps to separate arguments of a function call.
Missing End	The schema file currently being read does not have an END statement.
Missing File Statement	All infomate schemas must contain a FILE statement near the front so that data collected can be filed somewhere.
Missing Message	The ENTER statement does not have a message to be displayed to the operator. Note that the format is ENTER Field Message.
Missing TITLE statement	The schema file currently being read does not have a TITLE statement which should be the first line of the schema.
Mistake	A LET or TEST statement makes no sense to Infomate.
No Comms: Plug or Config?	The equipment connected to the serial port did not answer the query from Infomate. Check the baudrate, the configuration, the connection, and that power is applied.
No Data Fields	All schemas must have at least one data field. If they don't have data fields, then what is the point of collecting data?
No Data File Specified	You cannot run many of the Infomate modules until you have compiled a schema file to "tell" Infomate the name of the data file, and what to expect in the file.
No EPROM in Slot 3	If there is something in slot 3 then it is not a data EPROM, or it is faulty, or it is not inserted correctly.

No ID Fields	All schemas must have at least one ID field to identify different records.
No MAKE/FIND/NEXTRECORD	All sequence areas must contain at least one of the statements MAKERECORD, FINDRECORD, or NEXTRECORD.
No Schema File Prepared	You cannot run many of the Infomate modules until you have compiled a schema file to "tell" Infomate what to expect in the file.
No SETPORT function	The current driver does not support SETPORT commands.
No Space for ID's	This probably means that the file that exists on the RAMdisk is too big for Infomate. i.e. it is larger than 2000 records.
No Space on EPROM	The file that you wish to put on EPROM will not fit.
No Such Field	The field specified in the sequence line cannot be found in the field definitions at the start of the schema. Check spelling and capitalisation of letters.
No Such FN/PROC	A LET or TEST statement includes a reference to something starting with FN that is not part of a field name. You have probably misspelled the name of a variable.
No Such Line	One of the GOTO statements has specified a line that Infomate cannot match in the schema. Check spelling and use of capitals etc.
No such variable	A LET or TEST statement has referred to something that is not a field, a standard function, nor a valid Infomate variable. Spelling mistake?
No Tare Function	The current driver cannot be tared.
Not a DATA Field	You cannot use COUNTYPE or COUNTSIZE on ID fields.
Number of Fields	The COUNTYPE or COUNTSIZE lines imply use of fields beyond the end of the record or if you are using SCRATCHPAD, beyond the end of the scratchpad area.
Out of Range	The weight or other variable being measured by the equipment on the serial port is outside the range catered for by that equipment.

Out of Workspace	If this occurs then Infomate has run out of space to work on your task and has had to give up. It is difficult to guess how to solve this problem which will only occur during data collection. If you get it immediately you start collecting data you must severely reduce the size of your schema somehow. If it happens after a reasonably long period of capture, try occasionally going into the View/Edit module so that the data capture module is started afresh.
Outside Range	A CHECK statement has detected a field value that is outside the range allowed for it by the CHECK. You can ignore the warning, or go back to a point in your schema to correct the problem.
PORT not defined	You have used a SETPORT or a GETPORT line without defining the port using a PORTIS line at the start of the schema.
Port in Use	You cannot have a LOG PRINTER statement and a PORTIS statement in the same schema as this implies use of one port for two diverse functions. Try LOGging to EPROM instead.
Record too Long	The total number of characters in all fields exceeds Infomate's capacity. This could be the actual total which to allow the editor line to fit on the screen must be less than 86, or the total with blanks inserted between adjacent field where the total characters plus the number of fields must be less than 92.
REPEAT Duplicated	Infomate only allows one REPEAT UNTIL loop at a time. You have probably forgotten to insert the UNTIL part at the end of the loop, and have started another REPEAT loop.
REPEAT without UNTIL	You started a REPEAT UNTIL loop but did not finish it.
Schema too Long	There are too many lines in the schema file. Check the current maximum in the Infomate Limitations section.
Sorry, not implemented	You have inadvertently referred to a BBC BASIC function that does not exist on the Z88. This is probably a spelling mistake.
String too long	You have added some strings together in a TEST or LET statement and the total length of the string is longer than 255 characters.
Syntax error	Infomate started to understand what you were trying to say in a LET or TEST and then got a little confused. Is it a spelling mistake, or a misplaced bracket or comma?

Tare Failed	The driver test failed to get a satisfactory tare from the weighing equipment connected to the serial port.
TIME Syntax	In a LET statement you have used a TIME function that does not exist, or there is no closing bracket in the function. Check spelling and use of capital letters.
Too big	You have created a number that is too big for Infomate. The biggest number it can handle is about $+/-10^{38}$.
Too Many Constants	The compiler cannot handle that many constants. Constants are used to hold the numbers in a CHECK, COUNTSIZE or UNTIL STEADY line.
Too Many Fields	There are too many field definitions in your schema.
Too Many Steps	There are too many sequence lines in your schema file.
Too Many Strings	The compiler cannot handle that many strings. Strings are used to hold messages, filenames etc.
Type mismatch	The fields, function calls and constants in a LET or TEST are mixtures of letters and numbers. You have something like "letters" + 69 rather than "letters" + "69" .
Unknown Reply	The current driver received a response from the connected equipment that does not make sense. Is it connected correctly? Is the baud-rate set correctly?
Unknown Verb	When examining the SEQUENCE region of a schema, Infomate has found a Verb (the first word of a sequence line) that it does not understand. Check spelling, and that capital letters are being used.
UNTIL Syntax	There are only three types of UNTIL. They are UNTIL STEADY, UNTIL KEYPRESS, and UNTIL TEST. Check spelling and capital letters.
UNTIL without REPEAT	You cannot have an UNTIL in a schema until after a REPEAT statement. Keep REPEAT and UNTIL statements in strict order.
Waiting	Infomate is waiting for the equipment connected to the serial port to respond to a GETPORT. This may mean you have to press a GO button on the equipment.

```
TITLE        {message}
FILE         {filename.ext}
LOG          {EPROMlogfilename}
LOG          PRINTER[optbaudstring]
PORTIS       {drivername}[optbaudstring]          {optINITstring}
ID
             {IDfieldname}        {IDfieldtype}{IDfieldwidth}
DATA
             {DATAfieldname}      {DATAfieldtype}{DATAfieldformat}
SCRATCHPAD
             {SCRfieldname}       {SCRfieldtype}{SCRfieldformat}
SEQUENCE
ENTER            {fld}          {message}
ENTER!           {fld}          {message}
ENTER?           {fld}          {message}
ENTER?<          {fld}          {message}
ENTER?<+         {fld}          {message}
ENTER=<          {fld}          {message}
ENTER=<+         {fld}          {message}
ENTERID          {message}
ENTERID!         {message}
MAKERECORD
FINDRECORD
NEXTRECORD
CHECK            {fld}          {checkstring}
CHECK            {fld1}         {fld2}       {val1}       {val2}
CHECK            {fld1}         #            {val1}       {val2}
CHECKDONE        {fld}
IFFAILGOTO       {line}
TEST             {BASICexpression}
TESTRECORD
IFFGOTO          {line}
IFTGOTO          {line}
IF0GOTO          {line}
GOTO             {line}
LABEL            {labelcharacters}
DISPLAY          {fld}          {message}
DISPLAY          #              {message}
REPEAT           {message}
UNTIL            KEYPRESS
REPEAT           {message}
UNTIL            STEADY         {fld}        {value}
REPEAT           {message}
UNTIL            TEST
COUNTYPE         {fld}          {typestring}
COUNTSIZE        {fld}          {noflds}       {firstval}            {valincr}
LET              {fld}          =              {BASICexpression}
GETPORT          {fld}          {optmessage}
SETPORT          {message}
SLEEP
END
```

THE INFOMATE LIBRARY.

This disk contains PCLink, RangerLink, a Z88 BASIC, EPROM file recovery program called RECOVER.BAS, as well as numerous Infomate demonstration schema files.

The following schema files make up the Infomate Library:-

1. WEIGHKY.SCH	This is a simple keyboard entry weighing program.
2. WEIGHMT.SCH	This is an extension of 1, which uses the Mettler scales to collect the weights.
3. CALIPER.SCH	A short schema file showing how you can use Infomate to capture measurements from Mitutoyo or PAV electronic callipers.
4. PROD.SCH	Another short schema which demonstrates the counting feature of Infomate.
5. TOGGLE.SCH	A schema file designed for two people weighing herbage samples collected by one mower and two grass catchers.
6. HISTOG.SCH	A short schema file which demonstrates Infomate's countsize function.
7. STRATUMHT.SCH	A longer schema file which was designed to assess a forest for tree volume, and records the trees diameter and height.
8. AIRPORT.SCH	This schema uses Infomate's timer to log airport passenger traffic.
9. AIRPORT-.SCH	A modification to 8 which keeps better time.
10. ANIMAL.SCH	This schema works with a file called animal.dat to record new weights, comparing them with the old weights
11. ANIMALV2.SCH	This schema is a modification to 10, to enable running min, mean, and max to be computed.
12. ANIMALTR.SCH	A demonstration of Infomate's automatic processing capabilities, this file moves all the new weights into old weights ready for the next run.
13. DMGREEN.SC	This is a set of three schemas
14. DMSAMPLE.SCH	that were used to analyse
15. DMDRY.SCH	grass samples.
16. MTBENTRY.SCH	Another set of three schemas which
17. MTBTIMER.SCH	were used in a
18. MTBCALC.SCH	mountain bike race.
19. RHKEY.SCH	A program which prompts for wet and dry bulb temperature and then calculates relative humidity.
20. RH1.SCH	A quite large schema which records temperature, relative humidity, wind angle, wind speed etc.
21. ED.SCH	A short schema which allows you to edit past records.

All the schema files are in PipeDream format, and so can be transfered to your Z88 and compiled with Infomate straight away. You should try to understand as many of these schema files as possible even if you have no use for the particular application which it was written for, as they all include examples of many common tasks which you may use in your own schema files, and in many cases you can probably modify one of them to meet your needs.

Good Luck, and **don't forget to read the Manual!**