

LocoMail

LocoMail adds 'Mail Merge' and arithmetic facilities to LocoScript 2 on your Amstrad PCW8256 or PCW8512

LocoMail will:

- Produce a mailshot by merging a standard letter with a mailing list
- Automatically adjust the final layout to accommodate variable information
- Produce letters for each name or data record, or for just a selection
- Include special text depending on the data to be merged
- Edit the merged letter before printing
- Save the result to disc if required
- Use LocoScript data files, LocoFile data files or ASCII data files from a CP/M program
- Prompt for information to be typed at the keyboard
- Perform arithmetic within LocoScript documents
- Let you produce sophisticated LocoMail 'programs'

LocoMail

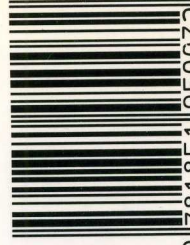
Mail Merge for LocoScript 2



LOCOMOTIVE
SOFTWARE

APR 89

ISBN 1 85195 007 9



9 781851 950072

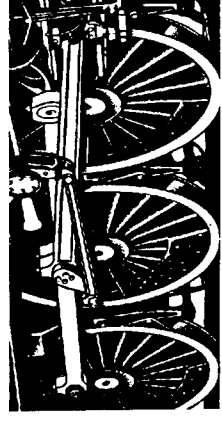
LOCOMOTIVE
SOFTWARE

User Guide

LocoMail

Locomotive Software's
Mail Merge for LocoScript 2

USER GUIDE



LOCOMOTIVE
SOFTWARE

Neither the whole, nor any part of the information contained in this User Guide may be adapted or reproduced in any material form except with the prior written approval of Locomotive Software Limited.

Whilst every effort has been made to verify that this software works as described, it is not possible to test any program of this complexity under all possible conditions or circumstances. Therefore LocoMail is supplied 'as is' without warranty of any kind either express or implied.

The particulars given in this User Guide are given by Locomotive Software in good faith. However, LocoMail is subject to continuous development and improvement, and it is acknowledged that there may be errors or omissions in this User Guide. In particular, the examples shown in this User Guide may differ in detail from the results of carrying out the same actions with the supplied documents. If you experience any difficulty with the use of this product, please write to:

Locomotive Software Ltd
Dorking Business Park
Station Road
Dorking
Surrey RH4 1YL

Locomotive Software reserves the right to revise this User Guide without prior warning. Notice of any such revisions will be included in 'Script', the LocoScript newsletter available from Locomotive Software.

Written and Published by Locomotive Software
Printed by Ashford Colour Press, Gosport, Hants

New edition, published February 1988 (Reprinted with corrections November 1989)
ISBN 1 85195 007 9

Some of the examples in this User Guide are based on LocoMail applications originally developed for Locomotive Software by Ed Phipps.

LOCOMOTIVE, LOCOSCRIPT and LOCO are registered trademarks of Locomotive Software Ltd
LocoMail, LocoFile, LocoScript 2 and LocoSpell are trademarks of Locomotive Software Ltd
PCW8256, PCW8512, PCW9512 are trademarks of AMSTRAD plc
WordStar is a trademark of MicroPro International Corporation

Introduction

Locomotive Software's LocoMail works with the LocoScript 2 word processing software to help you produce different versions of essentially the same document.

For example, you might use it when you want to add personal details to a standard letter or contract, or to fill in sales details on company invoices. All you need to provide is a master version of the letter, contract, or invoice (complete with the 'blanks' that you want filled), and the details that you want LocoMail to put into these 'blanks'.

LocoMail fills up the 'blanks' in the master version of the document – the slot for the date, the slot for the address, the price of a product, etc – in the same way as you might fill up a form. But as it works, it adjusts the way the result is laid out, so that at the end you get a perfectly finished document – just as if you had prepared it specially.

You can also make LocoMail do a lot more – you can get it to automatically produce versions only for people who meet certain conditions, you can automatically include or exclude different bits of text in your letter, depending on who you're sending the letter to, or you can get it to keep running totals of the prices and VAT amounts of all the goods you include on the invoice, then print these totals. In fact, the range of jobs you can carry out is almost endless!

LocoMail has two ways of working – known as Merge mode and Fill mode. In Merge mode, LocoMail works through a copy of the master version of the document, taking information from another file and slotting this information into the 'blanks' in the master document. In other words, it works rather like you might when copying details from a card index into a letter you are writing.

In Fill mode, LocoMail works through the master document, prompting you to type in all the details that you want to slot into each 'blank' – just as the headings on a form prompt you for the details.

These two ways of working are designed for different types of tasks:

- Merge mode is designed for producing multiple copies of a document, where the details you want to slot into the 'blanks' are stored on disc, for example in a datafile set up for use with the LocoFile database program. For instance, you might want to send an announcement of a new product to all your previous customers. LocoMail in Merge mode will take the names and addresses you want from your customer address list and produce individually 'personalised' letters. LocoMail will also allow you to add extra details from the keyboard, such as today's date.
- Fill mode is designed for producing one-off copies of your master document, or for the occasions where none of the information that you want to insert into the 'blanks' has been stored on disc. For example, you might want to send the details of a product in response to an enquiry from a potential customer: all you need to do is type the name and address into your standard letter.

As you see LocoMail in action in the tutorial examples of this User Guide, then use it to tackle your own particular jobs, you will soon recognise which of these two modes is best suited to a particular task.

You should be reasonably familiar with using LocoScript 2 before using LocoMail.

About this User Guide

The User Guide is in four parts:

Part I: Getting Started - tells you how to get LocoMail working on your PCW, then gives a number of tutorial sessions showing you how to produce a variety of 'personalised' letters using LocoMail in its Merge and Fill modes, and how to create a master document and an address file. We recommend you to work through all these tutorial sessions carefully before moving on to Part II and attempting to create your own LocoMail documents.

Part II: Using LocoMail - describes the fundamentals of how LocoMail works, the rules you must follow when creating your own data files and master documents, and how to test the files to make sure they work properly. You will need to be familiar with these fundamental actions before you attempt any of the more advanced uses of LocoMail.

Part III: Advanced LocoMail - describes LocoMail's advanced features and how you can use them to produce masters that will tackle more complicated tasks - for example, controlling which parts of the document are included, VAT calculations, keeping running totals, automatically renumbering clauses in a contract, etc. In this part, LocoMail is treated very much as a type of 'programming language'. Experience of programming would be useful here, but it is certainly not essential.

Part IV: Applications - shows how the LocoMail features of Parts II and III can be used to tackle a range of different applications:

- producing newsletters, fees reminders, lists of membership, etc. for a busy club
- generating invoices, producing sales and debtor lists, etc. from your sales records
- responding to simple enquiries for information (a list of houses in a particular price range, etc.)
- producing address labels on different types of labels stationery

All these documents are supplied on your LocoMail disc. Studying and using these documents should help you to apply LocoMail to your own particular needs.

At the end of the User Guide there are a number of appendices, covering everything from a full technical description of LocoMail to a troubleshooting guide for you to refer to if you have any problems while working with LocoMail. There is also a range of 'building blocks' in Appendix III. These are short, self-contained groups of LocoMail instructions that carry out specific tasks. You can often copy the ones you need straight into your own master documents.

We strongly recommend that you work through this User Guide in the way we have described above. Trying to use LocoMail just by 'trial and error' will almost certainly be making the job harder for yourself!

Note: The datafiles used in the examples in this User Guide are all special LocoMail datafiles or datafiles from database programs that have been dressed up as LocoMail datafiles. Information on using *LocoFile* datafiles with LocoMail is given in the LocoFile User Guide.

Conventions

The following conventions are used in this User Guide:

- ? " ! etc. - This style of text is used to indicate something to type or something displayed on the screen, in order to distinguish it from descriptive text.
- description* - Italic text is used for descriptions of the type of information that is required, rather than the information itself. For example, *number* could be 1 or 2 or 3 etc. If the description of a single item is more than one word long, the words are joined by hyphens: for example *item-name*.
- [*option*] - Slanting brackets round an item are used to show that this item is optional. Normally, including the item and omitting the item produce different effects. The brackets themselves should always be omitted.
- [RETURN], etc - The Return key, etc., not the letters RETURN

IMPORTANT: Data Protection Act 1984

Information about private individuals, that you process on your computer, may have to be registered under the Data Protection Act 1984.

Information that is only used in connection with personal, family or household matters or held only for recreational purposes may be exempt, but you will need to confirm this for your own particular case.

Guidelines giving full details are available from HM Stationery Offices or from:

The Office of the Data Protection Registrar
Springfield House
Water Lane
Wilmslow
Cheshire
SK9 5AX

LocoMail User Guide

8803

Contents

Part I: Getting Started	1
Installing LocoMail	5
Tutorial	
Part II: Using LocoMail	
1. Introduction to LocoMail	53
2. Setting up a data file	57
2.1 Setting up a data file specially	
2.1.1 Items and item-names	
2.1.2 Deciding on the separators	
2.1.3 Terminators and End-of-record markers	
2.1.4 Creating the data file	
2.1.5 Special cases	
2.2 Converting an existing data file	
2.2.1 A simple data file	
2.2.2 A more complicated data file	
2.3 Testing the data file	
2.3.1 Errors in the data file	
3. Planning and preparing a master document	77
3.1 Planning the master	
3.2 Putting in the LocoMail codes for marking blanks	
3.3 LocoMail instructions	
3.3.1 Instructions that take information from a data file	
3.3.2 Instructions that take information from the keyboard	
3.3.3 Instructions that set up information within the master document	
3.4 Improving the master document	
3.5 Testing	
3.5.1 Errors in the master	
3.5.2 Dealing with errors	

LocoMail User Guide

4. Using LocoMail in Merge mode 99

- 4.1 Getting started
- 4.2 Filling in the details
- 4.3 At the end of the pass

5. Using LocoMail in Fill mode 107

- 5.1 Getting started
- 5.2 Filling in the details
- 5.3 At the end of the pass
- 5.4 Using Fill mode to test master documents

The Exit menu 113

Part III: Advanced LocoMail

6. Making better use of your data files 119

- 6.1 Selecting specific records from the data file
 - 6.1.1 Conditional instructions
 - 6.1.2 Conditions that can be tested
 - 6.1.3 Example: Producing a selective mailshot
- 6.2 Controlling access to the data file
 - 6.2.1 Advancing to the next data record
 - 6.2.2 Keeping the same record for the next pass

7. Making master documents more versatile 133

- 7.1 Omitting sections of a document
- 7.2 Including alternative pieces of text

8. Manipulating text and numbers 141

- 8.1 Numeric and non-numeric information
- 8.2 Counting
- 8.3 Doing arithmetic
- 8.4 Formatting the results of arithmetic
- 8.5 Combining items of information

9. Using groups of LocoMail instructions 157

- 9.1 Setting up a program unit
- 9.2 Using the program unit once
- 9.3 Repeating program units
- 9.4 Controlling the number of repeats
 - 9.4.1 Counting the number of repeats
 - 9.4.2 When you don't know how many repeats
- 9.5 More complicated cases
- 9.5 An alternative approach

10. Nesting instructions 179

11. Designing master documents 183

- 11.1 Analysing the task
- 11.2 Setting out the master documents
- 11.3 Testing your master documents

Part IV: Applications

12. Club membership system 191

- 12.1 The data file
- 12.2 Printing member's details
- 12.3 Finding particular members
- 12.4 Customised newsletters
- 12.5 Membership fees reminders
- 12.6 Updating the membership details

13. Sales records and invoicing system 209

- 13.1 The data file
- 13.2 Adding records to the data file
- 13.3 Producing invoices
- 13.4 Updating the data file
- 13.5 Reminders for overdue invoices
- 13.6 List of debtors
- 13.7 Quarterly VAT totals

Part I

Getting Started

LocoMail for LocoScript 2 is made up of:

- The LocoMail program LOCOMAIL.JOY (supplied on the LocoMail Master disc)
- This User Guide
- A range of example documents, available on disc.

If you have just bought the LocoMail program to use on your PCW, you need to start by working through the instructions given overleaf for 'Installing LocoMail' and 'Making a LocoMail example documents disc'. We then suggest you work through the Tutorial sessions before going on to study the main part of this User Guide.

Note: The LocoMail program you have bought is for use with LocoScript 2. If you are still using LocoScript 1, the version of LocoScript supplied with your PCW8256/8512, then you need to buy LocoScript 2 before you can use LocoMail. Contact Locomotive Software for details.

If you already have LocoMail and have simply bought this User Guide to find out more about using this program, then there is no need to install LocoMail again or to study the Tutorial. Simply study the chapters of the main part of this guide that you require.

This applies both to people who have a PCW9512 (on which LocoMail is supplied as standard) and to people who originally bought LocoMail to work with LocoScript 1.

However, we would suggest that you buy the 'LocoMail Examples disc' (available from Locomotive Software). This disc contains copies of all the example documents used in this User Guide. In particular, it contains copies of the LocoMail applications described in Part IV and of the LocoMail 'Building Blocks' we suggest you use.

If you have already bought a LocoMail Examples disc then, before you use it, you should make a working copy of this disc. Full instructions on how to do this are given on the information sheet accompanying the Examples disc.

14. Enquiries system

- 14.1 The data file
- 14.2 Producing the response

235

15. Continuous labels

- 15.1 The data file
- 15.2 Setting up a master for 1-across labels
- 15.3 A master document for n-across labels

243

Appendices

I: Technical description of LocoMail

251

II: The LocoMail Master disc

259

III: Building blocks

263

IV: Troubleshooting Guide

273

End User Licence Agreement

Index

Installing LocoMail

IMPORTANT: The installation instructions given below apply only to owners of PCW8256 or 8512 machines who have just purchased the LocoMail program. If you have a PCW9512 or you are already using LocoMail with LocoScript 2, you don't need to follow these instructions.

The LocoMail Master disc contains a number of files:

- Side 1 of the disc contains the LocoScript 2 version of the LocoMail software, and all the example documents described in Parts I to IV of this User Guide. The complete list of files is given in Appendix II.
- Side 2 of the disc contains an Installation program.

To install LocoMail on your system, you need to run the special Installation program - either the one supplied on the LocoMail Master disc or one from another Master disc if this bears a later version number. The rule is always to use the Installation program with the latest version number on it.

The Installation program will take your current LocoScript 2 Start-of-day disc and any additional Start-up discs you have and produce new versions of these discs with LocoMail added to the range of LocoScript programs. (A Start-up disc is a disc that can be used to store extra files for loading to Drive M at Start-up. You may not have any Start-up discs at present.) Where appropriate, your LocoScript programs are automatically upgraded to work with each other.

You will probably need just one spare blank disc for this process.

When you use the new Start-of-day disc at Start-up or to reset your machine, the LocoMail software will be loaded with the LocoScript 2 software and the LocoMail facilities will be available as you use LocoScript.

To use the Installation program, switch on or reset your PCW. Then follow the instructions for 'Installing the new software' are given in the *Update Information booklet bearing the same LocoScript 2 version number as the Installation program you are using*. Be careful to check that these version numbers match.

The only special things to note about installing LocoMail are:

- When the program displays the list of LocoScript add-ons, use ☐ and ☐ to move the cursor (the highlighted oblong) to LocoMail and press ☐. This puts a marker beside LocoMail, showing that this add-on is selected.
- Adding LocoMail to your system does not reduce the size of Drive M that you will have to work with. So there's no need to reduce the range of add-ons you have on your Start-of-day disc, even on a PCW8256 with 256k of memory.
- There are no special support files to add to your Start-up discs, so you can ignore the section on adding extra support files unless you want to take the opportunity to store some of your other support files (printer files, dictionaries etc.) more conveniently. Similarly, if you are already using LocoScript v 2.20 or later, your font files (MATRIX.#xx files) should already be updated - allowing you to ignore the section on updating these files.

- You have now added the LocoMail software to your Start-of-day disc. Whenever you use this disc to start up or to reset your machine, all the LocoMail facilities will automatically be available.

Making a LocoMail example documents disc

To work through the examples in this User Guide, you need to prepare a working disc containing copies of the supplied example documents. As there are so many of these example documents, the easiest way of copying the documents is simply to copy the LocoMail Master disc onto a separate blank disc.

The steps are as follows:

- 1 Write-protect the LocoMail Master disc by opening the small write-protect holes in the top corners of the disc. (There is a picture on page 3 of the LocoScript 2 User Guide showing these holes.)
- 2 Load LocoScript. (For details, see page 19 of the LocoScript 2 User Guide.)
- 3 Press ☐ and select the **Copy disc** option from the menu that appears. What you are going to do is copy a 180k disc, so when this option appears on the screen, select it and press ☐.
- 4 A sequence of messages will then appear on the screen, telling you what to do. (See page 88 of the LocoScript 2 User Guide for details.)
- 5 If you have a single-drive PCW, insert your LocoMail Master disc in Drive A with Side 1 to the left, when a message appears on the screen asking you to insert the 'Source disc'. Press ☐. When a message appears telling you to insert the 'Destination disc', insert your new blank disc with Side 1 to the left and press ☐. When you have finished copying Side 1, a message will ask whether you want to copy another disc. Select **Return to disc manager** and press ☐.
- 6 If you have a two-drive PCW, insert your LocoMail Master disc into Drive B, with Side 1 to the left. Insert the blank disc in Drive A, again with Side 1 to the left. Press ☐. When you have finished copying Side 1, a message will ask whether you want to copy another disc. Select **Return to disc manager** and press ☐.
- 7 Take the disc(s) out of the drive(s) and put the LocoMail Master disc away. Label the Destination disc 'LocoMail example documents'.
- 8 Put the LocoMail example documents disc back in Drive A and press ☐.
- 9 Use the **Erase file** option in the F3 File menu to erase the files in group 0 of your LocoMail example documents disc - you don't need any of these files on your example documents disc. (For details of erasing files, see page 81 of the LocoScript 2 User Guide.)
- 10 The example documents have now been copied, and preparation is complete. Put your LocoMail Master disc safely to one side - you will only need it again if your Start-of-day disc should be damaged, or if you ever need to make further copies of the example documents. You are now ready to start using LocoMail as described in the rest of this User Guide.

Introduction to LocoMail

If you have never used a computer to fill in forms or prepare batches of letters ('mailshots') before, then there are several aspects of LocoMail that you may not have met:

- how you work with a master version of a document, complete with 'blanks' to be filled by LocoMail
- how to set up a list of information to fill the 'blanks'
- how you can be prompted to type in information for the 'blanks' when necessary
- how the document is tidied up as the 'blanks' are filled in

The best way to get used to this style of working is by seeing LocoMail in action and through practice - that is what this part of the User Guide provides through a series of tutorial sessions.

On your 'LocoMail example documents disc', in the group called TUTORIAL, there are a number of 'master documents' (that is, master copies of letters, forms, etc.) and an address list. One of the documents is a standard letter, to be sent to the people whose names and addresses are stored in the address list; others are slightly more complicated documents which need some extra details to be filled in at the keyboard. In this tutorial, we are going to look at and use these documents, prepare a master document of our own and prepare an address list for use with the mailshot letter.

Each session of the tutorial illustrates different aspects of using LocoMail. Rather than working straight through the complete tutorial, we suggest you practise using LocoMail in the way described in the first tutorial session until you feel confident, then go on to the second session and practise that, then go on to the third - and so on.

If anything happens while you are working through the tutorial that isn't explained in the text, turn to Appendix IV at the back of this User Guide. This is the troubleshooting section. You should be able to find out what has happened either from there or from your LocoScript 2 User Guide or from your PCW User Guide.

- To use LocoMail in the following sessions, you have to load LocoScript from the disc on which you installed LocoMail.

If your computer is switched off, switch on and insert the 'Start-of-day disc (with LocoMail)' you created during the Installation procedure. LocoScript 2 and LocoMail should then both be loaded, and the top of the Disc Manager Screen should display the normal range of LocoScript options with the two additional LocoMail options, **F=Fill** and **M=Merge**:

Session 1. Producing a mailshot

This session will show you how LocoMail can produce a number of versions of a standard letter, by 'merging' the letter with a list of names and addresses. What LocoMail does is to take a name and address from the list and slot it into 'blanks' in the letter – giving you a 'personalised' letter. It then repeats the process, slotting in the next name and address from the list into another copy of the letter, and so on. So what you end up with is a batch of letters, with one for every person on the list – this batch of letters is commonly known as a 'mailshot'.

The two documents that you merge are known as the 'master document' and the 'data file'. The master document is the document that you want to slot the details into – in this case, a standard letter. The data file is the document that holds all the details that you will put in the different versions of the master – in this case, names and addresses.

In the first part of this tutorial session, we will see LocoMail in action, merging the letter with the address list (ie. the master document with the data file) to produce the maillshot. We will also look at the contents of the master document and the data file in a little more detail, to see how LocoMail knows where to insert the information and how it knows what information to insert.

The second part of this session shows you how you can merge the same address list data file with a different master document to create the address labels for the mailshot.

Before you start the session, make sure that you have A4 (portrait) paper in the printer, since this is the paper that the MAILSHOT.EG master document is set up for.

Merging the documents

The two documents you are going to use to create the mailshot are a standard letter, stored in MAILSHOT.EG in the TUTORIAL group, and an address list which is stored in ADDRESS.LST. To merge these two, you use the LocoMail **M=Merge** option which you can see on the second information line of the Disc Manager Screen. You also need to tell LocoMail that MAILSHOT.EG contains the master document you want to use, and that ADDRESS.LST holds the information that is to be inserted. Do this as follows.

First move the File cursor until it is on MAILSHOT. EG (the master document), then press **M** to select the **M=Merge** option. An Alert message appears in the middle of the screen, telling you to pick out the document holding the data you want to merge with this master:

Merge documents:
select merge data document
then press ENTER
or CAN to abandon

[illegible]

If your machine is already on, finish any job you are currently working on and display the Disc Manager Screen (illustrated above). Check that the two LocoMail options **F=Fill** and **M=Merge** are shown on the second line of the screen – if they aren't, reset your machine using the **SHIFT**, **EXTRA** and **EXIT** keys and the Start-of-day disc that you created during the Installation procedure.

Note: If the two LocoMail options still don't appear on the Disc Manager Screen, the LocoMail software has not been installed correctly on your Start-of-day disc. In this case, work through the steps in "Installing LocoMail" again. Be sure to follow the steps very carefully.

When the full range of options are displayed, you can remove your Start-of-day disc (if it is still in the drive).

Each of the following tutorial sessions also requires one or more of the LocoMail example documents, which you copied to your 'LocoMail example documents disc' during installation. With the Disc Manager Screen showing, insert this disc and press 77. The examples we will be using in the Tutorial are the ones seen in the TUTORIAL group.

You are now ready to begin the tutorial.

Note: It is assumed throughout that a printer is attached to your PCW. If no printer is fitted, then LocoMail will operate in the same way but the options offered as it finishes filing each document will be more limited.

The data file you want to use is ADDRESS.LST, so move the File cursor to it and press **[ENTER]**. A menu will appear, allowing you to check that you have selected the correct master document and data file to 'merge'.

Merge documents	
Data:	ADDRESS.LST
Group:	TUTORIAL
Drive:	A
Name:	MAILSHOT.EG
Group:	TUTORIAL
Drive:	A
✓ High quality Draft quality	
Number of copies:	1
► Manual Automatic	

The names on the address list without any intervention on your part. The Manual option prepares one letter then stops and asks you what you want to do with this letter. Only then does it go on to prepare the next letter.

In this part of the tutorial, we are going to merge the documents automatically, so move the Menu cursor down until it highlights **Automatic** and press **[ENTER]**.

When LocoMail has finished filling the document, it is ready to be printed. But, before printing can start, the usual checks have to be carried out, comparing the printer and paper details in the document set-up with the printer and paper LocoScript is currently set up for.

These mailshot letters are intended to be printed on A4 paper in the PCW8256/8512's built-in printer. If this is what your system is set up to use, printing can start immediately though if this is the first document you've printed since switching on, you will first see a summary of the current print details so that you can check that these are as you expected). If your system is currently set up for a different printer or a different type of paper, you will see Alert messages telling you of those differences. The printer you use to print the mailshot is not crucial and you can accept the 'Use Current' option here (if you have a PCW9512, you will have to do this!) but we suggest that you only use either A4 or 11" continuous stationery to print the letters on. If necessary, you may have to accept the Use Intended option here and use A4 stationery in your printer.

Once printing begins, notice how LocoMail immediately starts to fill in another version of the letter.

LocoMail finishes preparing this next version of the document before it has finished printing the first one, so the following message appears on the screen:

LocoMail will print the document when the printer becomes idle
► Printer now idle: retry
Cancel operation

Leave the cursor over the **retry** option. As soon as the current version has finished printing, LocoMail will immediately take this message down and start printing the next version. (It doesn't need to go through the checking procedure again.) All you have to do at this stage is to keep the printer supplied with paper. Each finished version of the letter will be printed on a new sheet of paper (or on a new page, if you are using continuous stationery).

Note: If you need to abandon the Automatic Merge at any stage, press **[STOP]**, then **[STOP]** again, then select **Abandon LocoMail** from the menu that appears.

Each time LocoMail progresses from top to bottom of the master, filling all the 'blanks' as it goes (ie, putting in the name, the address and then the name again), it completes a 'pass' through the master. Each 'pass' results in a complete document. There are four names and addresses in the supplied ADDRESS.LST file, so assuming that you don't **Abandon** or **Cancel** at any stage, you will end up with four versions of the same letter. (Of course, each letter will be addressed to a different person):

The Widget Company
New Industrial Estate
Little Meadows
Worcester

Mr J G Smith
589 New St
Wintleham
Cheshire
CS4 8QT

21st May 1986

Dear Mr J G Smith

Please note that we are opening a new direct sales counter to provide a better service in Widget supplies. This will be located at:

52 High St
Westington
Teekensbury

Please call in and try our new service.

Yours sincerely

J Wendling
Sales Director

When LocoMail has produced a version of the letter for each person in the ADDRESS.LST file, it returns to the Disc Manager Screen.

Look at the printed versions. You can see that each version contains a different name and address, and these details have been inserted tidily, however long or short they are.

That is all there is to producing a mailshot from an existing master document and data file. The next stage of this tutorial session looks at the master document and the data file to see exactly what makes LocoMail insert the names and addresses.

Your screen should display:

```

Name: J. C. Smith
Address: Country
Language:

Mr J C Smith
589 New St
Wintlesham
Cheshire
CS4 8TU; England
English

Mr F M Tombs
The Cabinet
Elvington Rd
Driffield
East Sussex
BN1 1ST; England
English

Dr H Quintin
45 Green St
New Bondville
Glasgow
G4 5JH; Scotland
English

Mr P J McPherson
4 Black St
The Estate

```

As you see, ADDRESS.LST is divided up into a number of pages. (Each bar across the screen is an End-of-page line, marking where one page ends and another begins.) The first page has the words **Name**, **Address**, **Country** and **Language**; the other pages each contain a 'real' name, address, country and language. You will probably recognise these names and addresses as the ones used in the mailshot letters you have just printed.

The sections into which this data file is divided are called 'Records'. In this case, each record is a separate page, with the end of each record marked by a ↵ character. When you merge a master document with a data file, LocoMail uses each record for a separate version of the master. Each record in our address list contains all the details for one person, so you should end up with a separate letter for each person in the address list – exactly what you got earlier in this session.

The first section in the data file is called the 'Record Pattern' – because it defines the pattern each record has. The purpose of the record pattern is to tell LocoMail how to pick out a Name, an Address, a Country and a Language from each of the other records. You can do this just by looking at each page, because you know what each of these types of information looks like: LocoMail, however, needs to be told where one item of information ends and the next begins (ie. what separates the items of information), and what each item of information is called. In this instance, the record pattern is:

```

Name:
Address:Country
Language:

```

What this tells LocoMail is:

- that each record (page, in this case) of the data file should be divided up into a Name, an Address, a Country and a Language
- that the information before the first ↵ carriage return (ie. on the first line) in each record is a Name
- that everything between the first ↵ and the following semi-colon is an Address
- that everything between the semi-colon and the next ↵ is a Country
- that everything else in that record is a Language

The end of the record pattern is marked by the End-of-page character ↵.

If you look at the other pages of the data file, you will see that the first line is indeed always a Name; the Addresses vary in length but always start on the second line and are immediately followed by a semi-colon; the rest of the line after the semi-colon is the name of a Country; and the remaining information on the page is the name of a Language.

The individual details in the 'real' records (eg. J G Smith, or England) are called *items*, and the words in the record pattern (the first record) that are used to describe these items (eg. Name or Country) are called *item-names*. The characters used to separate the items (the semi-colon, and the ↵ carriage return) are called *separators*. These definitions become important when you come to prepare your own master documents and data files.

Note: You probably noticed that the MAILSHOT.EG master document we used with this address list didn't use either the Country information or the Language information from the data file. Data files can always contain more information than is actually required to fill any particular master document – this allows you to use the same data file with a number of different master documents which don't all require the same information.

It also means you can merge a master document with any data file as long as the record pattern of the data file contains the *item-names* used in the master. For example, a master that contains the LocoMail instructions (+Mail) name (-Mail) and (+Mail) language (-Mail) can be merged with any data file that includes name and language in its record pattern.

You also probably noticed that LocoMail inserted the name from the data file twice in each version of the master (once above the address and once in the Dear... salutation). The details LocoMail is currently using from the data file can be inserted any number of times in a version – and in any order. You aren't restricted in any way by the order in which the details appear in the data file.

• When you have finished looking at the records in the data file, and how they are split up into items by separators, press [EXIT] to leave the LocoScript Editor and select **Abandon Edit** when offered (because you haven't made any changes to the data file). The next stage in the tutorial describes how you can use this data file with another master document.

Using the same address list to create labels

The next stage in producing a mailshot is often to produce address labels for the envelopes. This can be done very simply because you can now use the same address list data file with a master document that's set up to produce labels.

Once again, you use the **M=Merge** option to merge two documents, and the data file will be the ADDRESS.LST document; this time, the master document you will use is the LABEL.EG document in the TUTORIAL group.

LABEL.EG is a very simple master document, which just consists of three lines. If you want to look at this document before you merge it with the address list, simply edit it as before. The following three lines will appear:

```
(+Mail)Name (-Mail) ␣
(+Mail)Address (-Mail) ␣
(+Mail)Country (-Mail)
```

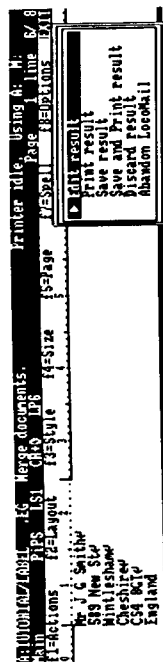
These LocoMail instructions tell you that when you merge this document with the address list, LocoMail will produce labels with a name on the first line, an address on the next line (using as many lines as it needs), and the country on the next line.

Note: You cannot select the LocoMail options while you are editing a document, so when you have looked at these instructions, leave the LocoScript Editor by pressing **[EXIT]** and selecting **Abandon Edit** to return to the Disc Manager Screen.

To use LocoMail's Manual merge, we essentially carry out the same steps as before. Move the File cursor until it is on LABEL.EG, and press **M** to select the **M=Merge** option. Then move the File cursor until it is over the ADDRESS.LST document and press **[ENTER]**. When the menu appears, check that you've selected the right documents and that the printer output quality is set to Draft quality. However, this time you are going to merge the documents manually, so that LocoMail produces one version of the master at a time – stopping after each version and asking you what you want to do with it, before carrying on with the next version. This is the default setting on the menu, so simply press **[ENTER]**.

For a few moments, the LABEL.EG master document is displayed on the screen exactly as if you were editing it. LocoMail then starts to process the document, filling in the 'blanks' just as it did with the MAILSHOT.EG letter. When it has inserted the name, address and country from the first 'real' record in the data file, your screen will appear like this:

Merge documents	
Data:	ADDRESS .LST
Group:	TUTORIAL
Drive:	A
Name:	LABEL .EG
Group:	TUTORIAL
Drive:	A
High quality	
✓ Draft quality	
Number of copies:	1
☒ Manual ☐ Automatic	



You can see the finished document (the label) on the screen, together with a range of options so that you can decide what you want to do with it. We describe what each of these options means later in this tutorial (in Session 3). For now, select **Print result** if you want to print the label out or **Discard result** if you don't.

If you select **Print result**, then as before you may see messages on the screen alerting you to any differences in how your printer is set up and how the master document is set up. In particular, you will see a message telling you that the paper types don't match. The LABEL.EG master document has been set up with a special paper type, called 1½" Labels, where the page height is only 1½".

Document and current printer do not match
Paper types differ

Current is: A4 Portrait
Intended: 1½" Labels ? Continuous

Use the current paper
Change to paper intended for document
Cancel operation

To see the labels being printed as if you had 1½" labels stationery in the printer, select the **Change to paper intended for document** option from this menu and press **[ENTER]**. The labels that you choose to print will then appear every 1½" down the paper.

Whether you choose to print or discard the current label, LocoMail will start the next version of the master document straight away – this time, inserting the name, address and country from the second set of details in the data file.

As you look at the finished labels, notice that although the Country information was on the same line as the address in the data file, in the finished label it appears on the line below – as dictated by the position of the **(+Mail)Country (-Mail)** in the LABEL.EG master. Also, there is no sign of the semi-colon that was between the Address and the Country in the data file.

• *LocoMail continues the passes through the master, producing a label from each pass, until it has worked through to the end of the data file (ie. four labels). When you have practised using LocoMail's Manual merge enough, printing some versions and discarding others, select **Abandon LocoMail** from the Exit menu and return to the Disc Manager Screen. If you want to repeat the merge, check that the File cursor is on the LABEL. EG master and press **M**. The steps you need to carry out are the same as described before (page 14).*

Session summary

In this first tutorial session, you have seen:

- how LocoMail takes pieces of information from one document (the data file) and inserts them into another (the master document)
- how to 'merge' details stored in a data file with a master document (both automatically and manually) using the LocoMail **M=Merge** option
- how items of information are stored in data files in a series of 'records'
- how the first record in the data file is the 'pattern' that tells LocoMail how to divide up the data file into records and how to pick information out of the records in the file
- how LocoMail recognises 'blanks' in a master document by the presence of (+Mail) and (-Mail) codes, and how you can recognise them because they are automatically highlighted
- how the *item-name* between the (+Mail) and (-Mail) codes in the master document tells LocoMail what information to take from the data file
- that in an 'automatic' merge, LocoMail prints every version of the master, but in a 'manual' merge you can choose individually for each version whether to save, print, edit or throw it away

Session 2. Adding details from the keyboard

The details from an address list can be inserted into 'blanks' in a range of master documents, but of course the information you can insert in this way is limited to whatever you have stored in the data file. What if you want to include some extra information in each letter that isn't stored in the data file – for example, today's date?

This session will show you how you can have a different sort of instruction marking a 'blank' in the master document – one which tells LocoMail that the information is to be typed in. When you merge the master with the data file, LocoMail will stop at this 'blank', wait for you to type in the details, then slot what you type into the 'blank'.

You will also see how LocoMail can remember the details that you type, and use them again further down the document. It can even store what you type and carry it over from one pass to the next!

The tutorial session has been split into three parts, to introduce you to the different types of LocoMail instructions that are used here. In each part, you will use a different master document, but you will always be merging the master with the ADDRESS.LST data file that you used in Session 1. In each case, we recommend carrying out a Manual merge (as you did when creating labels in the first session), so that you can readily abandon each document when you are satisfied that you understand what is happening.

I: The basic operation

The master document that you are going to use in this part of the tutorial is a standard letter such as a company might send out to its customers, and it is stored in MERGEDOC.EG1 in the TUTORIAL group of the LocoMail example documents' disc.

As we are going to use information from the data file, we begin by merging the master with the address list – so move the File cursor until it is on MERGEDOC.EG1, press **M** to select the **M=Merge** option, then move the File cursor until it is over ADDRESS.LST and press **ENTER**. When the menu appears, check that you've selected the correct documents, and that the printer output quality is set to 'Draft quality'. The Manual merge will already be selected (as it is the default setting) so simply press **ENTER**.

The first item of information which you see being slotted into the MERGEDOC.EG1 document is the name (which you may recognise as being the name from the first 'real' record in the data file); then you will see the address being inserted, correctly spaced out over a number of lines. All this is just as you have seen in Session 1.

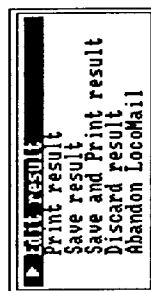
If you require any further information about Widget Company products, or details of our 'extended service' plan for your product, please do not hesitate to contact us. Please quote reference number JW/SU2205140. The Customer Support Manager for your area is (4UL)Mr Brown (4UL), at (4UL) (Mail) Local Branch Address (4Mail)22 Winston Parade (4Mail)22 Wintlesham (4Mail)22 Cheshire (4Mail)22 Yours sincerely (4UL)

The lines of the address aren't lined up properly at this stage – but don't worry about that. Just press **[ENTER]** to signal that you've finished typing. The prompt is then removed and the three-line address is inserted tidily into the document.

The reason that this works is that LocoMail always waits for you to press **[ENTER]**, and then takes everything you've typed (including tabs, carriage returns, etc.) – and slots it into the document. The result is exactly as if you had originally prepared the document with the same sequence of tabs, carriage returns, etc. in it.

At the end of the first pass

LocoMail now moves on to the end of the document – there are no more 'blanks' to be filled, either from the keyboard or with information fetched from the data file. Because you're merging documents manually, LocoMail then offers the Exit menu, so that you can choose what you want to do with this version of the master. For the time being, either select **Print result** if you want to print a copy of this filled letter, or select **Discard result** if you don't:



Note: If you don't see the Exit menu, this is because LocoMail has put you straight into the **Edit result** option, because you pressed **[ENTER]** more times than necessary while you were filling in the 'blanks' in MERGEDOC.EG1. To return to this Exit menu from editing, simply press **[EXIT]**.

Whether you chose **Discard** or **Print result**, LocoMail now starts the next pass through the master. The process of preparing the second version is exactly like the first: LocoMail begins by taking the name and address details from the data file (you might recognise the name and address as being from the second record in the address list data file). You will then be prompted for extra information – first for a new customer reference number, and then today's date, etc. Each time LocoMail stops and prompts, simply type in the information and press **[ENTER]**. Of course, you don't have to type in the same replies as on the first pass – so you can enter a different Product name, or a different Customer contact, etc. At the end of the pass, the Exit menu appears again.

Continue as long as you wish producing versions of the master document, then printing them or discarding them as required. If you run out of names and addresses in the data file, you can start the merge off again by carrying out the steps given on page 17.

When you have practised typing information into 'blanks' enough, return to the Disc Manager Screen by selecting **Abandon LocoMail** from the Exit menu when it is displayed. We will look at the contents of the MERGEDOC.EG1 master document, to see what is making LocoMail stop at some of the 'blanks', produce a prompt for details and wait for the details to be typed in.

Looking at the master document

As always the way to see the instructions that have been used is to look at the master document using the LocoScript Editor. So with the File cursor on MERGEDOC.EG1, press **E** then **[ENTER]**. The document should now appear as below:

```

1 (Mail) (4UL) Mr Brown (4UL), at (4UL)
2 (Mail) Local Branch Address (4Mail)22 Winston Parade (4Mail)22
3 Wintlesham (4Mail)22 Cheshire (4Mail)22
4 Yours sincerely (4UL)
5
6 Your ref : (Mail) (4UL) Customer reference (4Mail) (4UL)
7 (Mail) (4UL) Customer reference (4Mail) (4UL)
8 Dear (Mail) (4UL) (4Mail) (4UL)
9
10 Many thanks for purchasing a (Mail) (4UL) product. Purchased (4Mail), and
11 returning your guarantee registration card, I am sure that this
12 product will give you many years of trouble-free service. (4UL)
13
14 If you require any further information about Widget Company
15 products, or details of our extended service plan for your
16 product, please do not hesitate to contact us. Please quote
17 reference number (Mail) (4UL) Customer reference (4Mail) (4UL)
18 (4UL) on your area is (4UL) Mr Brown (4UL), at (4UL)
19 (Mail) (4UL) Local Branch Address (4Mail) (4UL)
20 Yours sincerely (4UL)
21 (4UL) Hendleings
  
```

The letter on the screen looks like the master document you used in the first session of the tutorial – except it has rather more blanks. Moreover, while the instructions which insert the name and address are exactly as in the previous version, the other instructions are examples of a different type of LocoMail instruction. They all have the same structure – they all begin with a question mark and contain a semi-colon followed by text:

(+Mail) ? ; Customer reference (-Mail) (This one appears twice)
 (+Mail) ? ; Today's date (-Mail)
 (+Mail) ? ; Product purchased (-Mail)
 (+Mail) ? ; Customer's contact (-Mail)
 (+Mail) ? ; Local Branch Address (-Mail)

The ? character after the (+Mail) code tells LocoMail to stop and wait for you to type something in. To tell you that you have to type something (and what you should type) LocoMail needs to display a 'prompt'. You will probably recognise that the prompt LocoMail used in each case was the text shown here after the semi-colon. In other words, the 'blank' shown as:

(+Mail) ? ; Customer reference (-Mail)

tells LocoMail "Stop here, display the prompt Customer reference and wait for information from the keyboard", and similarly the 'blank' shown as:

(+Mail) ? ; Today's date (-Mail)

tells LocoMail "Stop here, display the prompt Today's date and wait for information from the keyboard".

When you have finished looking at these instructions in the MERGEDOC.EG1 document, you can leave the LocoScript Editor and return to the Disc Manager Screen by pressing **EXIT**, selecting an option and pressing **ENTER**. (Select the **Abandon Edit** option, as you haven't made any changes to MERGEDOC.EG1.)

II: Re-using information

As you produced the different versions of the letter in the previous example, you probably found it annoying to type in the customer reference twice for each letter, and to enter today's date for each version. It would be so much easier if you only had to type these details once and LocoMail automatically remembered them for use again later, either in the same document or in the next pass.

There are LocoMail instructions to do exactly these things, and you can see them in action by merging the master document called MERGEDOC.EG2 with your address list. MERGEDOC.EG2 is the same letter as before, but some of the 'blanks' are marked with different LocoMail instructions to make your job (typing in the information) easier.

Once again, we will start by merging the master document with the ADDRESS.LST data file before we look at the LocoMail instructions it uses. Move the File cursor to MERGEDOC.EG2 and press **M** to select LocoMail's **M=Merge** option. Then move the File cursor to ADDRESS.LST and press **ENTER**. Check that you have chosen the correct documents, and that the printer quality is set to 'Draft quality' and press **ENTER**.

LocoMail then starts the first pass through the master, filling in the name and address details as before. You are then prompted for the Customer reference, Today's date, and the Product purchased just as before.

However, you will see an important difference at the next 'blank': instead of stopping and waiting for you to type in the customer reference again, LocoMail automatically inserts whatever you typed in response to the earlier **Customer reference** prompt and goes straight on to the 'blank' for the **Customer contact**. The instructions now being used for the Customer reference 'blanks' have told LocoMail to remember the details you typed earlier and use them later in the same document.

The rest of the pass is exactly as before. When the first pass is complete, LocoMail offers the Exit menu. If you want to print a copy of the letter, select **Print result**; otherwise, select **Discard result**. LocoMail then begins the second pass, inserting the name and address from the second record in the data file.

You will then be prompted to enter a new customer reference number for this new letter - so although LocoMail remembered the customer reference you typed and used it later in the same pass, the details were forgotten at the end of the pass (this is just what you want because you need a new reference number for each letter). Type a new reference number and press **ENTER**. LocoMail will then move on to the 'blank' for the date.

However, instead of prompting for today's date again, LocoMail inserts a date without any action on your part. The date it puts in is the one you typed on the first pass. The instruction marking the blank for the date is obviously different from those marking the customer reference number, because unlike the customer reference information (which was forgotten between passes), LocoMail has remembered the information you typed and carried it over between passes.

The remainder of the 'blanks' in the document are filled as on the first pass (ie. you are prompted for product, contact and local branch address, and the new customer reference is automatically slotted in when necessary). At the end of the pass, select either **Print result** or **Discard result** and continue producing versions.

On each subsequent pass, you will need to enter a new customer reference number. The date, however, will be automatically inserted into all passes, until you return to the Disc Manager Screen at the end of the merge.

We will now look at MERGEDOC.EG2 to see how LocoMail remembered the customer reference and date details. We will also see how LocoMail knew to keep the date for all versions but to ask for a new customer reference number each time. So move the File cursor until it is over MERGEDOC.EG2 and press **E** then **ENTER**.

The next important instruction to look at is the one for the date:
 (+Mail) ! ? date : Today's date (-Mail)

(+Mail)? reference : Customer reference (-Mail)

both in structure and in how it works (in this case whatever you type will be remembered under the *item-name* **date**) but with one important difference – the exclamation mark at the beginning. When LocoMail comes to this point as it works through the master, the exclamation mark tells LocoMail to check in its memory **first**, to see if you have already stored anything under this name (**date**, in this case) – if there is, LocoMail will use that information and ignore the ?; if there isn't it will carry out the instruction as if the ! wasn't there.

On the first pass through the document, there is nothing stored as **date**. So LocoMail stops and waits (because of the **?**), prompting you with the text from after the semi-colon (**Today's date**). Whatever you type before pressing **[ENTER]** will be stored as **date**. On the second and subsequent passes, there is something stored as **date**, so LocoMail simply ignores the **?** part of the instruction and inserts whatever is stored under **date**.

What you see in this master document is the operation of three different types of LocoMail instruction for taking information from the keyboard:

- the simplest type of instruction:
(+Mail) ? ; prompt (-Mail)
which makes LocoMail stop, prompt for information, insert it then move on, forgetting what you typed.
- instructions including an *item-name*:
(+Mail) ? item-name ; prompt (-Mail)
which tell LocoMail to store what you type as well, so that it can appear later in the same document.
- instructions including an *item-name* and an exclamation mark:
(+Mail) ! ? item-name ; prompt (-Mail)
which tell LocoMail to store the information and carry it over from one pass to the next.

Any information which is stored under an *item-name* can then be inserted into a blank marked with the simple instruction:

- (+Mail) item-name (-Mail)**
- When you have finished looking at the instructions in the MERGEDOC.EG2 master document, you can return to the Disc Manager Screen by pressing **EXT.** As you haven't made any changes to the master, select **Abandon edit** from the menu that's offered.

The instructions that remembered what you typed:

(+Mail) ? reference; Customer reference (-Mail)

The only difference between this instruction and the corresponding one in MERGEDOC.EG1 (which you looked at earlier in this session) is the word following the question mark (**reference**). As before, the question mark tells LocoMail to stop and wait for you to type in the information, and the text after the semi-colon is used as a prompt. The word after the question mark and before the semi-colon tells LocoMail to store what you type under an *item-name* – just like the *item-names* we saw being used for the items of information in the data file in Session 1. In this case it is remembered under the *item-name reference*. The reason for storing it is so that you can use it later on without having to re-type it.

If you look down the document, you can see that the 'blank' in the document where the customer reference details are to be inserted a second time is marked by the instruction:

This is essentially the same type of instruction as (+Mail) Name (-Mail) used to insert the name details from the data file – it tells LocoMail to insert an item of information it already knows, and which is stored under this *item-name*.

III: Typing in all the details at the beginning

The final part of this tutorial session shows you how LocoMail can ask you to type in all the information needed to fill in that version of the master at the start of the pass, by using yet another sort of LocoMail instruction. A major advantage of doing things this way is that you type in all the information at once, usually in a more sensible order than the information appears in the document.

The master document you are going to use in this example contains the same letter as you have filled in the previous examples. It is stored in MERGEDOC.EG3. We will begin by merging this master with the ADDRESS.LST data file, then examine the master in more detail. So move the File cursor to MERGEDOC.EG3 and, press **M**; then move the cursor to ADDRESS.LST and press **ENTER**. Check that you have selected the right documents, the printer quality is set to 'Draft quality' and press **ENTER** to select Manual merge (set by default).

This time LocoMail stops right at the beginning of the document and displays:

Customer reference?

Type the customer reference just as you have done in the previous examples, pressing **ENTER** when finished. LocoMail immediately offers the next prompt, for **Product purchased**, and so on. Type in the details of the product, the customer contact, the local branch address and the date exactly as you have been doing earlier in this session (adding styles such as bold, underlining, etc. if you wish), and finishing each response with **ENTER**.

As you enter each reply, notice how both the prompt and details disappear from the top of the screen. This time, LocoMail isn't inserting what you type straight into the finished document, as it did in the previous examples. All that remains in the document after each prompt disappears is a **←** (a blank line).

When you have supplied all these details and pressed **ENTER** after the last line of the local branch address, LocoMail then moves down through the document, and inserts each piece of information at the appropriate point(s). You don't need to reply to any more prompts – all you have to do is decide what you want to do with the finished version when the Exit menu is displayed.

Select **Discard result** or **Print result** as before: LocoMail then begins the second pass.

Once again, you are prompted for the customer reference, product information, customer contact and the local branch address, but not for today's date. This is carried over from the first pass.

Continue filling in versions of the master: when you have filled enough, select **Abandon LocoMail** from the Exit menu to return to the Disc Manager Screen.

We shall now edit MERGEDOC.EG3 to see the type of LocoMail instruction that you've been using.

Looking at the master document

Move the File cursor to MERGEDOC.EG3 and press **E** followed by **ENTER**.

```

Reference: Customer reference (Mail)
Product: Product purchased (Mail)
Contact: Customer contact (Mail)
Branch: Local branch address (Mail)
Date: Today's date (Mail)
The Widget Company
New Industrial Estate
Little Headway
Morester

Your ref: (Mail) Reference (Mail)
Date (Mail)
Dear (Mail) Name (Mail)

Many thanks for purchasing a (Mail) product (Mail), and returning your
warranty registration card. I am sure that this product will
give you many years of trouble-free service.

If you require any further information about Widget Company
products, or details of our 'extended service' plan for your
product, please do not hesitate to contact us, please quote
reference number (Mail) Reference (Mail). The Customer Support Manager for
your area is (Mail) Contact (Mail), at:

(Mail) Branch (Mail)
Yours sincerely
J Headings
Sales Director

```

As you can see, the document begins with a group of five LocoMail instructions, which again contain question marks but are of a slightly different form to those you've seen so far.

Four of the instructions are similar:

```

(+Mail) reference=?; Customer reference? (-Mail)
(+Mail) product=?; Product purchased? (-Mail)
(+Mail) contact=?; Customer contact? (-Mail)
(+Mail) branch=?; Local branch address? (-Mail)

```

As before, the ? in each of these instructions tells LocoMail to prompt for the details (using the text after the semi-colon as the prompt), and wait for your reply. Similarly, the presence of an *item-name* in the instruction tells LocoMail to store the information you type under this *item-name*. However, the fact that the instruction starts *item-name*= tells LocoMail that it isn't to insert the details you type into the document at this point. Further down the document, wherever each of these details are to be inserted we use instructions like:

```

(+Mail) reference (-Mail)
and (+Mail) product (-Mail)

```

The other instruction in this group is the one requesting the date:

(+Mail)!date=?;Today's date (-Mail)

Once again, the exclamation mark is used to tell LocoMail "ask for the date only if you haven't got one". Otherwise, this instruction is just like the other four in appearance and action, storing the details without inserting them – wherever the date is to appear in the finished document we use the instruction:

(+Mail)date (-Mail)

As you look at the rest of the master, you can see that it is a lot less cluttered than the previous examples. This is because each 'blank' that is to be filled is marked only by the simple **(+Mail) item-name (-Mail)** type of instruction, which inserts the details you stored at the beginning of the master. The text of the letter is considerably easier to read than the previous examples. Each of these **(+Mail) item-name (-Mail)** instructions could include comment text, but when the names are suitably descriptive this is generally not necessary.

As you continue working with LocoMail, you will see blocks of instructions made up of **(+Mail) item-name=? ; prompt (-Mail)** instructions used more and more.

Note: When LocoMail worked through this initial block of five instructions, the carriage return at the end of each line ended up in each finished letter. Obviously, this could be inconvenient – what you really want is a series of LocoMail instructions which leave nothing inserted in the document. You could do this by omitting the **↵**s between the **(-Mail)** and **(+Mail)** codes, and running the five instructions one after another:

**(+Mail)reference=?; Customer reference?
(-Mail)(+Mail)product=?; Product purchased?
(-Mail)(+Mail)...**

but this doesn't make your master document very clear. However, finishing one instruction with **(-Mail)** and starting the next one with **(+Mail)** isn't the only way you can tell LocoMail that you have got two LocoMail instructions following straight on from one another. Instead, you can use a **↵** – giving you each instruction on a separate line as follows:

**(+Mail)reference=?;Customer reference?↵
product=?; Product purchased?↵
...(-Mail)**

(Notice that the sequence of five instructions still starts with a **(+Mail)** code and ends with a **(-Mail)** code.)

Carriage returns that are used in place of **(-Mail) (+Mail)** combinations like this are not inserted into the finished document.

(Note: To see the effect of replacing the **(-Mail) (+Mail)** combination with **↵**s, when you have finished with MERGEDOC.EG3 and returned to the Disc Manager Screen, you can merge the MERGEDOC.EG4 document in the TUTORIAL group with the ADDRESS.LST data file to see it being filled. Notice how none of the **↵**s between the instructions are inserted into the finished version this time.)

• When you've finished looking at the instructions in MERGEDOC.EG3 that will store information without inserting it, return to the Disc Manager Screen by pressing **[EXIT]** and selecting **Abandon edit** (as you haven't made any changes to the document).

That completes the second session of the tutorial. In the three examples, you have seen a lot of different types of LocoMail instruction, and you might be feeling that there is a lot to learn. But don't worry – you will find that as you use the supplied masters you'll soon be able to work out what each instruction you come across means.

Session summary

In this session, you have seen:

- how LocoMail fills certain types of blank with details that you type.
- how the simplest of these blanks are marked **(+Mail) ?; prompt (-Mail)** and the information you type is inserted, then immediately forgotten
- that LocoMail can remember what you type, and use it later in the same document if the instructions contain an *item-name*
- that LocoMail can carry over the details you type from one pass to the next if the instruction both contains an *item-name* and starts with a **!**
- that you can make a master document easier to use by grouping the information-gathering instructions at the start of the master document, using instructions of the type *item-name=?*

Session 3 – Using LocoMail's Fill mode

So far in the tutorial you have just been using LocoMail's Merge mode – which enables you to insert details from a data file into your master document. The other LocoMail option you can see on the second information line of the Disc Manager Screen is **F=Fill**. This allows you to fill in a master document without using a data file; instead, all the details are typed in from the keyboard. Filling in a document in this way is particularly useful if you only want to produce at most a handful of filled versions of the master, when it might not be worth involving a data file (as you'd have to throw away versions you don't want); alternatively, it might be that none of the details to be inserted are stored in a data file yet, or you only want to use information from the keyboard anyway. As we'll see later, the **F=Fill** option is also useful for testing new master documents.

The steps needed to start filling a document are very simple – you don't have to specify a data file, and you don't have to choose between different types of Fill. You just have to specify the print quality you want, and the number of copies.

This session has been split into two parts – in the first part, we will fill in a simple document where all the instructions take information from the keyboard; in the second part, we will use the MERGEDOC.EG1 master that you merged in Session 2, to see how you can use the Fill option to type details into blanks where the information is meant to come from a data file.

I: A simple master

To demonstrate exactly how you would fill in a letter or form using LocoMail in Fill mode, we will fill in the FILLDOC.EG master document in the TUTORIAL group, then look at its contents. FILLDOC.EG is a simple letter, such as might be sent in response to an enquiry. All the 'blanks' are to be filled from the keyboard.

Place the File cursor over FILLDOC.EG and press **F** to select the **F=Fill** option. You first see a Selection menu so that you can check you picked the right document and set the print quality options. For the purposes of this tutorial, set the options to 'Draft quality' and single copy (if they are not already set). When all the details are as you require, press **[ENTER]**.

Note: Selecting **F=Fill** in this way is all you need to do when testing the master documents you create – as we'll see in Session 4.

For a few moments, the screen will show the FILLDOC.EG document just as if you were editing it. LocoMail then starts to process the document.

Filling in the blanks

The first 'blank' to be filled as LocoMail works through the document in Fill mode is shown by the prompt: **Enter enquirer's name?**

Typing in details in LocoMail's Fill mode follows exactly the same procedure as typing in the Merge mode – you simply type the text you require (complete with word-processing codes, 'styling' codes, **[RETURN]** for new lines, etc.) and when you have finished, press **[ENTER]**. So type in a name and press **[ENTER]**. LocoMail inserts the details you've typed and moves on.

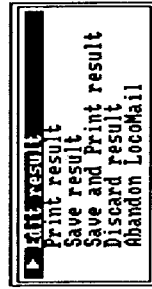
The next blank in the letter is marked by:

Product of interest?

Once again, type the name of a product. If you want, underline the product name by first typing a **(+UL)** code, then the product name, then a **(-UL)** code before pressing **[ENTER]**. (How to do this was described in Session 2.) LocoMail moves on through the document, tidying up (relaying) the paragraph as it goes.

The final blank in FILLDOC.EG requests the price of the product. Type in a price in any format you want (eg. **£17.99**, or **Seventeen pounds and 99p**) and press **[ENTER]**.

LocoMail then moves through to the end of the document, completing the first pass. The screen displays the finished document, together with an Exit menu – offering the same six options at the end of each pass as in a Manual merge.



(If you don't see the Exit menu, LocoMail has put you straight into editing the document, because you pressed **[ENTER]** more times than it expected as the document was filled. To get back to the Exit menu from editing, simply press **[EXIT]**.)

In Session 2, we ignored most of these options and just concentrated on the **Print result** and **Discard result** options. Their effect is just as before. This time, we will look at the effect of the other options.

Edit result allows you to make any changes you want to the filled document on the screen, just as you might edit any other LocoScript document. You can then delete or cut text, type or paste in new text, insert codes, etc. When you've finished making changes to the version, press **[EXIT]** and you'll be returned to the Exit menu. (The changes you make only affect this filled version of the master – the master document itself is unaffected.)

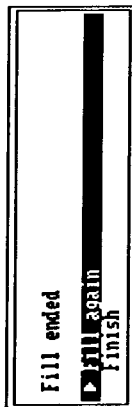
Save result allows you to save the current version on disc. Once you've selected **Save result**, the Disc Manager Screen displays a message telling you to choose the drive and group that you want to store the version in. Use the cursor keys (together with **[SHIFT]** if necessary) to select the drive and group that you want to use, then press **[ENTER]**. Type in a name for the document (or accept the suggested name offered on the selection menu) and press **[ENTER]**.

Note: The saved version of the document occupies disc space – a valuable commodity, so you should think whether you really want to keep a filled version as well as the master document before saving.

Save and Print result is for when you want to save the current filled version of the master and also keep a printed copy. The finished version is first saved on disc as described for **Save result**, and then it is printed just as if you had selected **Print result**.

The final option on the Exit menu is the **Abandon LocoMail** option. Selecting **Abandon LocoMail** immediately leaves LocoMail and returns you straight to the Disc Manager Screen. The filled version of the master currently on screen will be discarded. (Of course any document that is currently being printed will be completed.)

Choose an option from this menu in the usual way (just moving the Menu cursor and pressing **ENTER**). Once LocoMail has saved the current version, discarded it or started printing it (as appropriate), it then gives you the choice of filling in another version of the master document, or finishing with LocoMail:



Selecting **Fill again** from this menu will give you a fresh unfilled version of the letter on the screen, ready for you to work through again. Selecting **Finish** will take you out of LocoMail and straight back to the Disc Manager Screen. For the time being, keep selecting **Fill again**, as this will allow you to prepare further versions of the letter and try out each of the options on the Exit menu. You don't need to wait for any printing to be completed – LocoMail can fill in the next version, while printing the current one.

Keep producing filled versions and try out the effect of the other Exit options.

• When you have used the **Fill** mode on this simple master document enough (ie. you've tried out each of the options on the Exit menu), leave LocoMail either by selecting **Finish** from the short **Fill again/Finish** menu or selecting **Abandon LocoMail** from the Exit menu. In both cases you will be returned to the Disc Manager Screen.

Looking at the master document

To see the instructions that have been used we need to look at the master document, so move the File cursor until it is over **FILLDOC.EG** and press **E** then **ENTER**.

(Mail) ? ; Enter enquirer's name (-Mail)
 (Mail) ? ; Product of interest? (-Mail)
 (Mail) ? ; Price of product? (-Mail)
 Thank you for your letter requesting details of our company services. I see that the (Mail) (Mail) (Mail) is of particular interest, and consequently I have taken this opportunity to send you the full technical specification of this product. You can see that this is a quality product, designed for even the toughest jobs around the house.
 At the moment, this product is available from us at the special price of only (Mail) (Mail) (Mail). If you would like to confirm an order at this once-in-a-lifetime price, please complete the attached order form and return it to me. I will see that it is dealt with by return of post.
 Yours sincerely
 A J Hendleings
 Sales Administrator

The three 'blanks' you filled in are marked:

(+Mail) ? ; Enter enquirer's name (-Mail)
 (+Mail) ? ; Product of interest? (-Mail)
 (+Mail) ? ; Price of product? (-Mail)

This is the same type of instruction that you saw when looking at the MERGEDOC.EG1 master document in Session 2. In that session, you were using the Merge option – here you have used the Fill option. Nevertheless, the instructions are the same and they work in exactly the same way both times – making LocoMail stop, prompt for details and then insert what you typed.

In fact, all instructions that fetch information from the keyboard work in exactly the same way in both LocoMail's Merge and Fill modes.

• When you have finished looking at the **FILLDOC.EG** master, return to the Disc Manager Screen by pressing **EXIT** then selecting **Abandon edit** (since you haven't made any changes).

II: Filling in 'Merge' blanks

In this part of the session, we will see how the Fill option can be used to type in information that is intended to come from a data file. The document we will use is the MERGEDOC.EG1 document we used in Session 2 to take names and addresses from a data file.

To fill this document, place the File cursor over **MERGEDOC.EG1** and press **F** to select the **F=Fill** option. Set the options on the Selection menu to 'Draft quality' and single copy (if they are not already set). When all the details are as you require, press **ENTER**.

For a few moments, the screen will show the MERGEDOC.EG1 document just as it did under the LocoScript Editor at the start of Session 2. LocoMail then starts to process the document.

The first 'blank' to be filled as LocoMail works through the document in Fill mode is the **(+Mail)Name (-Mail)** blank. You will remember that when working in Merge mode, this instruction told LocoMail to take the name from the data file. But this time, as you are working in Fill mode, LocoMail treats all instructions as if they require information from the keyboard. So it stops and waits for you to type in a name.

So that you will know what to type, LocoMail again supplies a prompt. This prompt would normally consist of any text which followed a semi-colon in the instruction (just as it did when you were typing dates, products, etc. in the previous example). In this case, however, there isn't any text to use – so LocoMail does the next best thing by displaying the *item-name* in capitals. So, what you will see on the screen is:

NAME?

Type in a person's name, just as you did when typing in information in the previous session, and press **[ENTER]**. LocoMail inserts the name into the document and moves on to the next blank.

The second blank in the document you are filling is marked by the instruction **(+Mail)Address (-Mail)** in the original MERGEDOC.EG1 document. This is the same type of instruction as **(+Mail)Name (-Mail)**, so LocoMail will treat it just the same, giving you the prompt:

ADDRESS?

Type in the address of the intended recipient. Use the **[RETURN]** key to spread the address over the required number of lines, just as you did when typing a 'local branch address' in Session 2. Only press **[ENTER]** when you've finished the address details.

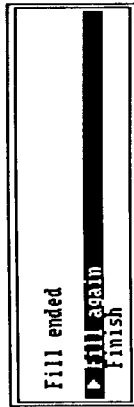
LocoMail then moves on to the blank in MERGEDOC.EG1 where the customer reference is to be inserted. The screen will show the prompt **Customer reference** just as it did when working in Merge mode. Type in the details and press **[ENTER]**. Then type in Today's date as before.

The next place in the MERGEDOC.EG1 master where information is to be inserted is the second Name blank, is on **(+Mail)Name (-Mail)** line.

At this point, LocoMail will automatically insert the name you typed in response to the **NAME?** prompt above – ie. it has stored what you typed under **name** and re-used it.

LocoMail carries on through the document, stopping at each blank and waiting for you to type the details. All the remaining instructions work in the same way as when you used a Manual merge to fill the document, and so the way you're prompted and what LocoMail does with the details you enter are the same as in Session 2.

When the last blank in the document has been filled, LocoMail has finished the first pass and so it displays the Exit menu. When you choose an option from this menu and the appropriate action has been taken, the short Fill again/Finish menu will appear:



Selecting **Fill again** from this menu gives you a fresh unfilled version of the letter on the screen, ready for you to work through again. Selecting **Finish** takes you out of LocoMail and straight back to the Disc Manager Screen.

- *When you have used the Fill mode enough, leave LocoMail (either by selecting Finish from the short Fill again/Finish menu, or selecting Abandon LocoMail from the Exit menu). In both cases you will be returned to the Disc Manager Screen.*

Session summary

In this third session, you have seen:

- how you can use the LocoMail **F=Fill** mode to fill in blanks in a master document with information that you type
- how LocoMail prompts for information that is intended to come from a data file, by using the comment part of the instruction. If there is no comment, then it displays *item-name*?
- the details of all the Exit menu options
- that at the end of each pass in Fill mode, you can choose whether to fill another version, or finish

Session 4. Preparing a master document

So far in this tutorial, we have just used ready-prepared master documents – complete with all the LocoMail instructions necessary to create the different versions of the document. When you want to use LocoMail for your own jobs, you'll have to create the particular master document you want to work with (and possibly create your own address list data file too). This session of the tutorial looks at how you create a master document.

Creating a master document from scratch (ie, typing in all the text of the document and including the LocoMail instructions as you go) is not difficult. However, you will often find it much easier to start from a LocoScript document that matches the sort of finished document you will want LocoMail to prepare for you – for example, a letter that tells Mr Smith where to get his washing machine fixed – and then convert this to be your LocoMail master document by adding LocoMail instructions to it.

We have put just such a document in with the LocoMail examples, and what we are going to do in this part of the tutorial is convert this document into a simple master document which will take all the details from the keyboard. You will see how to insert LocoMail instructions – in particular, the (+Mail) and (-Mail) codes – at the appropriate points in the document. After using your master, you will then see how to make some improvements that will make the master easier to use.

The sample document we are going to work with is stored in MASTER.EG.

Working out where to use LocoMail instructions

The first step in preparing a master document is to work out what parts of the existing LocoScript document you will want to change from version to version. These are the places that you will need to put LocoMail instructions.

The easiest way to do this is to go through a print-out of the existing LocoScript document, marking the parts you will need to change. MASTER.EG is set up for A4 paper, so put some A4 paper in your printer, select MASTER.EG with the File cursor and press P (for Print). When the selection menu appears, check that you've chosen the right document, set the printer quality to 'Draft' (to speed up the printing) and press **ENTER**.

MASTER.EG will then be printed (following the usual checks of the current printer and paper type and the ones set in the document).

The print-out you get should look like this:

The Widget Company
New Industrial Estate
Little Meadows
Worcester

11 July 1988

Dear Mr Smith,

Thank you for your letter of 5 July. I am extremely sorry to learn that the SuperVac is not working correctly. I can assure you that our customers have reported very few problems with this product.

May we suggest in the first instance that you take it back to our York branch for servicing. Ask to speak to the Service Manager. He has been advised of the problems you have been experiencing.

Yours sincerely

Joan Smith
Customer Services Manager

Starting from the top, the first item to mark is the date **11 July 1988**, because this will always need to be set to the day you actually send the letter.

Next on the list is the customer's name, **Mr Smith**, because this will be different for each letter you send. Also on the list will be the date the customer's letter was sent (**5 July**) and the name of the product causing the trouble (the **SuperVac**).

The final piece of the letter to mark is the name of the company's branch (**York**) that the customer is recommended to take the product to for servicing. This gives us a marked up letter looking like this:

The Widget Company
New Industrial Estate
Little Meadows
Worcester

11 July 1988

Dear (Mr Smith)

Thank you for your letter of **5 July**. I am extremely sorry to learn that the **SuperVac** is not working correctly. I can assure you that our customers have reported very few problems with this product.

May we suggest in the first instance that you take it back to our **York** branch for servicing. Ask to speak to the Service Manager. He has been advised of the problems you have been experiencing.

Yours sincerely

Joan Smith
Customer Services Manager

Tutorial

(a) (5)
 (a) (5)
 (a) (5)
 (a) (5)

Dear Mr. Smith,⁴

Thank you for your letter of 5 July. I am extremely sorry to learn that the Superluc is not working correctly. I can assure you that our customers have reported very few problems with this product.⁴

4/ May we suggest in the first instance that you take it back to our York branch for servicing. Ask to speak to the Service Manager. He has been advised of the problems you have been experiencing.4/

Joan Smith
Customer Services Manager

Yours sincerely,

Joan Smith
Customer Services Manager

...s date

...s date

and finally enter the **(-Mail)** code to complete the instruction. To enter the **(-Mail)** code, press **[F]** followed by **M** (or press **[F]**, wait until the Clear menu appears then select **Mail** and press **[ENTER]**) – just like other codes that start with a -.

38

code, followed by a ? ; and finish with a (-mail) code. There should be no (+mail) and (-mail) codes in the document other than the ones starting and finishing your LocoMail instructions. If you get into any difficulties, abandon this version of the document and start over again with a fresh copy of MASTER. EG.

Adding some extra features

Once you have got this master document working properly with the simplest LocoMail instructions, you might like to try enhancing it. With the Disc Manager Screen displayed, edit the MASTER.001 document. Change the LocoMail instruction that asks for the date to:

(+Mail) ! ? date ; Today's date (-Mail)

As you will remember from the second session of this tutorial, this type of instruction tells LocoMail to look whether a date has already been entered before prompting for one. The result is that on the first pass through the master document, you will see the prompt **Today's date**, but on subsequent passes in the same batch the date will be inserted automatically.

There are other enhancements you can make to this document. For example, you can change the instruction that asks for details of the product so that it becomes:

(+Mail) ? product ; Product (-Mail)

and replace the word 'it' in the last paragraph by:

your (+Mail) product (-Mail)

The first of these two instructions tells Loc

to store it under the *item-name* **product**; the second tells it to slot in the details it has stored under **product**. Your master document will then look like this:

(Mail) } The Night Courier
 (Mail) } New Industrial Estate
 (Mail) } Little Wymondley
 (Mail) } Newcastle
 (Mail) }
 (Mail) } (Mail) } (date) } (only's date) } (Mail) }
 (Mail) }
 Dear (Mail) } (Customer's name) } (Mail) }
 Thank you for your letter of (Mail) } (date of customer's letter) } (Mail) } . I
 am extremely sorry to learn that the (Mail) } (product) } (Mail) } is not
 working correctly. I can assure you that our customers have
 reported very few problems with this product. }
 May we suggest in the first instance that you take your (Mail) } (product) } (Mail) }
 back to our (Mail) } (local branch) } (Mail) } branch for servicing. Ask to speak
 to the Service Manager. He has been advised of the problems you
 have been experiencing. }
 Yours sincerely }
 () }
 () }
 Joan Smith }
 Customer Services Manager }

Try filling the document when you've made these enhancements, and see how your job of filling in the details has become easier.

In making these enhancements, we had to think of some names to describe the items we wanted to store the information under. Note how the *item-names* we've suggested are single words (**date** and **product**). *Item-names* are always single words: they also have other characteristics:

- characters in an *item-name* can only be drawn from the letters **A...Z** (or **a...z**), the numbers **0...9**, accented letters and the underline character **_**
- the *item-name* never includes any spaces, characters like **, . () /** or hyphens
- *item-names* all start with a letter (not a number).

The underline character can be used in an *item-name* to join two-word descriptions into single-word *item-names*, eg. **christian_name** for a christian name.

So, for example, the following are valid *item-names*:

```
telephone    unit_cost    née
while these are unacceptable:
```

tel.no	unit cost	laccross
full stop character	space in name	starts with a digit

Note: You might also notice from this that although we have been using *item_name* to represent the names you use, you couldn't actually use the name *item_name* because of the hyphen. You could, however, use *item_name* or *itemname*.

And finally

As a final step, you could now try converting a copy of one of your own LocoScript documents into a master document by replacing the special details it contains with LocoMail instructions similar to the ones used here. Not only will this be very good practice, it could also give you a very useful master document!

Session summary

In the fourth session of the tutorial, you have seen:

- how to work from an existing LocoScript document to create a master LocoMail document
- how to put LocoMail instructions into a document – in particular, how to put in (**+Mail**) and (**-Mail**) codes.

Session 5. Setting up a LocoMail data file

In the first session of this tutorial, we produced a maishot by using the letter stored in MAILSHOT.EG as the master document and merging this with names and addresses taken from the ADDRESS.LST data file. In the same session, we also used this address list with LABEL.EG to produce labels for the envelopes for these letters.

As the final stage of the tutorial, we shall show you how to set up a LocoMail datafile from scratch. What we shall do is create a list of names and addresses of our own to merge with the MAILSHOTEG and LABEL-EG documents, ie. one that will suit the requirements of these master documents.

Note: If you have the LocoFile database program on your system, then we would always recommend using LocoFile to store your data rather than the techniques described here. However, it is still worthwhile to study this session of the tutorial because it will help you when you come to insert data into LocoFile from other types of datafile.

Identifying the data that must be in the data file

As the data file you're going to create is to be used with the MAILSHOT.EG and LABEL.EG master documents, you need to know what information each master document takes from the data file. The first thing to do, therefore, is to examine these documents to see what LocoMail instructions they contain. Starting from the Disc Manager Screen, place the File cursor on MAILSHOT.EG, press **␣** and then **[ENTER]** when you have checked that the details on the Selection menu are correct. When the document appears on the screen, display the codes. Press **[␣]**, then **[F]** to set the Codes option and then press **[ENTER]**

Mailgm
 Mailgm
 Mailgm
 Mailgm
 (Mail) Name (Mail)
 (Mail) Address (Mail)
 21st May 1988
 Dear (Mail) Name (Mail)
 Please note that we are opening a new direct sales counter to provide a better service in Midget supplies. This will be located at:
 52 High St
 Westlinton
 Westsberg
 Please call in and try our new service.
 Yours sincerely
 Handling
 Sales Director

44

The two Loco Mail instructions this document contains are:

(+Mail) Name (-Mail) (this appears twice)
(+Mail) Address (-Mail)

Now do the same for LABEL. EG. You will see that this contains the instructions:

(+Mail) Name (-Mail)
(+Mail) Address (-Mail)
(+Mail) Country (-Mail)

So this means that each record in our data file has to contain a **Name**, an **Address** and a **Country**. (It can also contain more information than this if you want, but that needn't actually concern us here.)

Planning the data file

When you start planning a new data file, the first job is to decide:

- which order you are going to put the pieces of information in each data record
- what you are going to use as the signal that tells LocoMail where each piece of information ends

Working out these things will help you to see what record pattern fits this arrangement. In fact, when you are setting up a data file from scratch it is often easiest to think in terms of working out the record pattern first.

Deciding on the order for the information is usually very easy, because it is almost always best to choose the most natural order. In this case, that is **Name Address Country**.

Deciding on the signals to use is more complex. They have to be either characters like `,` `.` `(` `)` `+` `!` or the carriage return character `\r`. However, what you choose for a particular signal must be something that never appears in the piece of information it will mark the end of – otherwise LocoMail will cut this information short.

In the data file we are preparing here, we shall have:

Names like: Tim Smith, J. B. Harrison and John Baker-Smythe

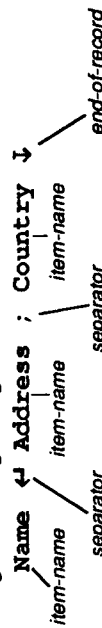
Addresses like:
**1, The Mews
Cottage Row,
Newville,
Glos**

and Countries like: England, Sri Lanka and USA

The signal (or 'Separator') we shall use after a Name is $\leftarrow$, because real names never have carriage returns in them. However, we can't use $\leftarrow$ to mark the end of an Address because addresses are typically written over a number of lines and so contain a number of $\leftarrow$ s of their own. Instead, we shall use ; as we don't expect any of our real addresses to contain a semi-colon.

As the Country is the last item of information in the record we shall follow it with a ↵ (an End-of-page character). ↵ is one of the signals we can use to mark both the end of the record pattern and the end of each data record.

Putting this all together gives us:



This is our record pattern and it tells LocoMail to take everything up to the first ↵ as a Name, everything between the first ↵ and the next ; as an Address and everything else in the record (page) as a Country.

We shall now see what this record pattern looks like when we put it in a data file and what the data records have to look like to match the pattern.

Creating the data file

To create the data file, we simply create a new document in the usual way (ie. as described in the LocoScript 2 User Guide or the PCW9512 User Guide). So with the Disc Manager Screen displayed, place the File cursor over the drive and group in which you want to create your data file and press C (for Create). LocoScript will present a short menu – give the new document a name that you will be able to recognise (eg. DATAFILE.001, but any name will do) and press [ENTER]. A new document will be created and will appear on the screen.

Note: If the new document is not empty (ie. because it is created using a template which contains text), then delete all text until you have a completely blank document.

• Typing in the record pattern

When you have the new blank document on the screen, you start by typing in the Record Pattern. This has to come first because LocoMail has to read this before it can pick out information from the rest of the data file. So type:

Name ; Address ; Country

then hold down the [ALT] key and press [RETURN]. (This puts in an End-of-page character – you can use the F5=Page menu to put in this character if you prefer.)

Your screen should now show the following first page:

```

Name;Address;Country

```

Now follow the record pattern with the data.

• Typing in the data records

To start the data file off, type:

```

John Baker-Smythe
1, The Mews,
Cottage Row,
Newville,
Glos,England
Tim Jones
24 Front Street,
Llanweller,Wales

```

Dictated by the record pattern

By now, you should have the following on the screen:

```

Name;Address;Country
John Baker-Smythe
1, The Mews,
Cottage Row,
Newville,
Glos,England
Tim Jones
24 Front Street,
Llanweller,Wales

```

Now type in some other names and addresses. Remember to type each name on the first line of a page, to put a ; immediately after each address and to follow that directly by the country. Don't press [RETURN] before you type in the country. Finish the country by putting in an End-of-page character ([ALT] and [RETURN]).

Note: Take care not to type in any spare spaces within or after any item of information in your data file – if you do, LocoMail will insert them into the master document along with the information, giving you unsightly blank spaces in the finished documents.

Testing the new data file

When you have finished typing in all the Names and Addresses, save your new address list and then try using it with MAILSHOT.EG in Merge mode. Move the File cursor to MAILSHOT.EG and press M, then move the File cursor to DATAFILE.001 and press [ENTER]. Check that all the details on the Selection menu are correct, and then press [ENTER] again. (This is exactly the same sequence of steps as you took to merge ADDRESS.LST with this master document in Session 1 of this tutorial.)

If you have prepared the data file carefully, LocoMail should now start filling the MAILSHOT.EG with your new Names and Addresses. As each version is completed you will be offered the options on the Exit menu, which we described in Session 3. Continue selecting **Print result** or **Discard result** until you have seen LocoMail create a version for every record in your data file.

If, instead:

- an Alert message appears on the screen, or
- LocoMail fills the letter but gets the information in the wrong places, or
- LocoMail only puts one line of the address in the address slot, or
- LocoMail misses out complete records (ie. you don't get a finished version for everyone in your list)

then you have made a mistake in typing the record pattern or the data. You need to edit your new address list.

Begin by checking the first page (the record pattern). This must be exactly as shown:

```

Name
-----
Address;Country
-----

```

ie. **Name**, **Address** and **Country** must be spelt correctly and there must be a ↵ after the word **Name**, just a ; between **Address** and **Country** and a ↵ after the word **Country**.

Then check that you have each real name on the first line of a new page, that you have just ; between each address and the country, and that you have an ↵ at the end of each country. Also check that none of your addresses include a ; before the one marking the end of the address. Once you have these right, your data file will work correctly with MAILSHOT.EG.

If you wish, you can also try using this new address list with the LABEL.EG master document. This will give you labels to go with the letters you produce with MAILSHOT.EG - just as it did when you used ADDRESS.LST with this master document.

Postscript

Planning a record pattern can seem quite complicated when you are new to using LocoMail. If you have an existing data file that works with your master document, then it is often best to simply copy its record pattern for use in the new file.

Instead of what we did in this tutorial session, you could have copied the record pattern from ADDRESS.LST, ie:

```

Name ↵
Address ; Country ↵
Language ↵

```

and put that at the top of your new data file. You then type in your own information, following the rules laid down in the record pattern - ie. ↵ after a Name, ; after an Address, ↵ after a Country, and ↵ after a Language.

It looks from this that you would have to type in the Language for each 'real' data record, even though this isn't information you need. In fact, you don't have to type in anything more than the information you want. You don't even have to type a Country or an Address if you don't want. Just type ↵ after the last item you want to include in any data record. The remaining items will automatically be set to null - that is, no characters at all.

For example, you could have a data file like this:

```

Name
-----
Address;Country
-----
Mr J G Smith
589 New St
Hertfordshire
SG4 8CT
English
Mr F W Toalss
The Gablers
Livington Mde
Draffwalle
Edinburgh
Dr H Quentine
i England
English
Mr D D Whetherville
W
Welsh

```

This is the data file ADDRESS2.LST in the TUTORIAL group. Study it and the record pattern and work out what information you think it contains. Then merge it with LABEL.EG in the normal way and see if you were right!

Session summary

In this part of the tutorial, you have seen:

- how to create a simple data file containing addresses, for use with the LABEL.EG and MAILSHOT.EG master documents

Part II: Using LocoMail

LocoMail gives you a very powerful way of using information stored on disc to prepare finished documents. The full range of facilities allows you to select just the records you need from a datafile, to include arithmetic in your documents, to specify optional sections of text, to carry out repeated actions (eg. building up an invoice), and much more besides.

However, the main purpose of LocoMail is simply to make it quicker and easier to prepare multiple versions of essentially the same document, pulling in information from a datafile. The sort of thing you might commonly use it for would be preparing mailshots or company invoices – any job where you are essentially slotting information that you have on disc into a 'standard' document. This information can be stored either in LocoFile datafiles (ie. datafiles set up by the LocoFile database program), or in datafiles prepared by CP/M database or accounts programs or in special 'LocoMail' datafiles that you set up for yourself.

To get LocoMail to do these tasks for you, you need to

- decide on the document that you want to produce multiple versions of
- work out where the information that you want to insert in the document will come from
- prepare the data you want to insert from disc, ready for use with LocoMail; if necessary, creating datafiles specially
- create the master document, complete with LocoMail instructions.

This part of the User Guide is designed to help you carry out these fundamental tasks. It introduces the way LocoMail works and describes:

- how you go about planning and creating the master document
- the LocoMail instructions that mark 'blanks' in the master
- how to prepare LocoMail datafiles, both from scratch and from existing datafiles produced by CP/M programs, and
- using LocoMail in its Fill and Merge modes.

Most of these stages were introduced in the tutorial – though without going into any real explanation about what you were doing. This part of the User Guide describes all the stages more fully and should act more as your reference on using LocoMail for these standard tasks. Part III looks at the more advanced facilities and uses of LocoMail.

- If anything happens to your work that isn't explained in the text, turn to Appendix IV – the Troubleshooting section. This describes problems you might encounter within LocoMail. Any more general problems will be described in your LocoScript 2 User Guide.

Introduction to LocoMail

LocoMail is primarily designed to take pieces of information from one file – the data file – and slot these details into 'blanks' in another – the master document. You might use it, for example, to send a standard letter (like a change-of-address slip, or a business letter) to all your friends or customers. In this case the data file you use would contain all the names and addresses, and the master document would be the letter you want to send, with LocoMail instructions marking the places where you want to slot in each name and address. LocoMail merges the names and addresses from your address list into the standard letter, producing a separate document for every name on the list.

As well as taking the details directly from a data file, you might want to include some other details which you don't have in your data file – like today's date, for example. These can be typed at the keyboard into each version.

In this chapter, we introduce the fundamental ideas behind LocoMail. Understanding these fundamentals will help you to use LocoMail to best advantage.

How LocoMail works

LocoMail works with LocoScript in merging data into master documents. LocoMail's part of the job is to create as many copies of the master document as are needed and to obey the special instructions that mark 'blanks' in the document. Typically the instructions tell LocoMail to pull in the piece of text needed at the spot marked by the instruction – ie. to fill in the 'blank'. LocoScript's part of this job is to ensure that the resulting document is correctly laid out and to print and/or save it as required. The way this works is as follows.

LocoMail takes a copy of the master document then works right through it, carrying out the instructions, fetching and inserting bits of text as it goes. Everything else in the document apart from these instructions is left untouched. As LocoMail works through the document, LocoScript works alongside, relaying paragraphs to tidy up after the pieces of text have been inserted and interpreting the word-processing codes. By the time LocoMail reaches the end of the document you are left with a finished document, correctly laid out and with specific details added in place of all the LocoMail instructions. You can then save the document or print it (a task carried out by LocoScript).

Much of what LocoMail does is fetching and inserting bits of information. There are two main sources for this information:

- items of information stored on disc in a 'data file'. For example, the information stored on disc could be names and addresses in an address list.
- information typed in at the keyboard.

The datafile must be carefully set up so that LocoMail can always pick out the right information. Datafiles set up for use with the LocoFile database program can usually be used as they are, but datafiles generated by other database programs need preparation before they can be used by LocoMail.

The sort of information you can insert is anything that you might put in a normal LocoScript document – text of any length (from a character up to a page or more), any of LocoScript's characters, tabs, carriage returns, word-processing codes, etc. When LocoMail inserts these into the 'blank' the result is just as if you had typed them directly into the document yourself.

LocoMail has two main ways of working through a master document: Merge mode and Fill mode. These two ways of working are designed for different tasks.

Merge mode is designed for mailshots – letters you want to send to large numbers of people on your address list. For example, you might want to send details of the annual dinner to all members of your club. LocoMail used in Merge mode will take the names and addresses you want from your address list and produce individually personalised letters.

Fill mode is designed for letters and documents that you want to prepare a few at a time, when you just want to type in the details. For example, you might receive inquiries from time to time about a particular service your company offers, or you might use essentially the same form of agreement for all the contracts you have with your clients. Either of these would be well served by a single LocoScript document, personalised with the help of LocoMail in its Fill mode.

If any of the information you want LocoMail to use is stored in a data file, then you must work in Merge mode; if none of the information is on disc (or it is impractical to get at) you work in Fill mode.

In using LocoMail, you prepare a number of 'personalised' versions all from the same master document. LocoMail produces the different versions you require by repeatedly working through and filling in copies of this master. Each time it works through the master from start to finish is called a 'pass', and each 'pass' gives you a separate complete document. If you decide to print this version, it will start printing on a separate sheet of paper (exactly as happens when LocoScript prints other documents).

Note: Each filled version you produce is held on Drive M before it is saved or printed and so you need to check that there is sufficient free space on Drive M for this before you fill or merge a master document. In most cases, the filled version will be about the same size as the original master, but if the master contains a 'loop' (using instructions described in Part III), the filled version can be very much larger than the original document.

When LocoMail takes its information from a data file, it expects to use one set of details per version (eg. one person's name and address), then automatically move on at the end of the pass to the next set of details. Each separate set of details in a data file is called a *record*, and the number of passes that LocoMail makes through a master document matches the number of records in the data file. There are, however, special instructions that can be used to modify this and use more than one record in a pass, or keep the same record for more than one pass.

Each record comprises one or more *items* of information. An important feature of LocoMail is that each item is referred to by an *item-name*. To enable LocoMail to pick out information correctly, you specify how the data file is divided into records, how these records are divided into items, and also the *item-name* associated with each item of data. This specification is given in the *record pattern*, and it is put either at the beginning of the data file or, occasionally, in a separate file (if it is to be applied to more than one data file).

Information which is typed in can also be stored under *item-names*. Assigning *item-names* to information means that any stored information (whether from the data file, entered at the keyboard or set up within the master document) can be used in the master document simply by giving the relevant *item-name*.

Particular information stored under an *item-name* can normally only be used in the current pass. The information may be used any number of times, anywhere in the document. At the end of each pass, the information associated with the *item-names* in the data file changes as LocoMail moves on to the next record. Information that was typed in or set up within the master document, and stored under *item-names*, is forgotten. (However, special steps can be taken to make the information available for other passes.) The information stored under a particular *item-name* can be changed as often as you want during a pass, provided that it was either entered at the keyboard or set up from within the master document. Information associated with an *item-name* in the data file cannot be changed.

The most common LocoMail instructions simply slot in pieces of information into 'blanks' in the master document. These include instructions that store the information for use later in the same document, or for use in every document. These instructions can cover a wide range of the jobs you might want to do with LocoMail. There are also a large number of instructions that do much more complex jobs – testing conditions to see what text should be included, discarding versions that aren't required, repeating batches of LocoMail instructions, doing arithmetic with numeric information, etc.

The more complex LocoMail instructions can be used together, to form 'programs' of instructions – even offering the same sort of facilities as a simple BASIC program. This combination of the flexibility of programming-type facilities with the LocoScript word processor gives you very powerful document production facilities indeed.

- The rest of this part of the User Guide looks at the data files, the master documents and the simpler LocoMail instructions that will fetch, insert, store and re-use items of information. These will allow you to produce 'mailshots' and the like by merging data files and master documents. In addition, we describe how you can use LocoMail to produce one-off copies of forms by filling in all the details from the keyboard.

Part III looks at the more complex LocoMail instructions and how you can use them in your masters to perform the more complicated tasks that LocoMail is capable of – selective mailshots, 'clever' masters where the content of the finished version changes from one pass to the next, invoices with calculations, etc. It also describes how to design your masters to get the best from using LocoMail.

Setting up a data file

Data files are essentially lists of information. This information will typically be data that you will want to use over and over again. You might have this data stored in a LocoScript document or in a LocoFile datafile.

The other sort of information you'll want to access from LocoMail is the data that's generated, for example, by your order processing program or your accounts program. LocoMail is able to help you here because it is also able to take information from simple text files (ASCII files) created by any CP/M program.

If you have the LocoFile database program, then you will probably want to store your data as LocoFile datafiles (as described in the LocoFile User Guide). If you don't have LocoFile, then you will need either to set up a LocoMail data file for your data or to get LocoMail to take data from your existing data files. Neither of these tasks is difficult, and in this chapter we will show you how you do both.

All data files are arranged with a fundamentally 'regular' structure – one set of details following another, and each set of details containing the same type of information (although not necessarily the same amount of information, as we shall see).

The key to using the information from a data file is being able to tell LocoMail about this structure. You need to be able to tell it where one set of details ends and the next begins, and, within each set of details, where one item of information ends and another begins. You also need to tell it what each item of information is, so that you can pick out the ones you want.

In a LocoFile datafile, the information is automatically laid out in a way that LocoMail can understand. In other data files, you need to mark the end of one piece of information and the start of another, and the division between separate sets of details, using a system of special markers. You then need to tell LocoMail what these markers are and what they mean.

In describing how to do this, it helps to introduce some special terms that are used to describe the structure of data files. As we briefly mentioned in Chapter 1, each separate set of details is known as a *record*. Each piece of information within a *record* is known as an *item*. The marker that is used to mark where one item ends and another begins is called a *separator*, and as well as a *separator* between each item in the record you generally have another marker to pick out where one record finishes and another starts – an *end-of-record* marker.

To allow LocoMail to recognise the different items of information in each record, and insert them in a master when necessary, it needs a special record called the *record pattern* – either at the beginning of the data file, or in a separate document it inspects first. The record pattern consists of a series of *item-names*, which are

used to refer to the equivalent item of information in every record, with each *item-name* followed by the corresponding *separator* and the last *item-name* followed by a *terminator* which tells LocoMail it has reached the end of the record pattern.

Once LocoMail has read through this record pattern, it knows what item-names will be used to refer to the items in the data file and what characters will be used as separators between them. It also knows how many items make up a record. So it can tell when it has reached the end of a data record, either because it meets an end-of-record marker, or because it has counted the right number of items and separators.

When you use LocoMail instructions to insert a piece of information from the data file, all you have to do is ask for the piece of information by its *item-name*. LocoMail picks out the piece of information with this *item-name* from the record being read in the current 'pass' through the document.

So that it can distinguish between *item-names*, *separators* and *terminators* in the record pattern, LocoMail is quite specific about what is acceptable as an *item-name* and which characters (or combination of characters) can be used as the *separators*, *terminators* and *end-of-record* markers. The rules are given in Section 2.1.

If you have not encountered anything similar to records, record patterns, items, etc. before, they probably seem a little confusing at first. But don't worry – they become much clearer once you've seen them in action.

This chapter describes how to set up a data file that can be used with one or more of your master documents – either by preparing the data file specially (Section 2.1) or by adding a record pattern to an existing data file (Section 2.2). Typically, the record pattern for these files will be at the beginning of the data file, but you may want to store it in a separate document when:

- you have a number of data files that require the same record pattern – data files that you have generated using the same accounts program, say
- the data file has been generated by a different program and you cannot edit it to add a pattern to it
- you have a number of master documents, wanting to use the same information, but describing it in different ways – for example, using *item-names* like *first_name* and *surname* in one case and *full_name* in another. (There is another way of getting round this problem – by using 'String Concatenation'. This is described in Part III, Chapter 8.)

Section 2.3 describes how to go about testing your data file to make sure that it gives you the results you want. Testing requires a master document which works properly; creating and testing the master is described in Chapter 3.

- *Preparing a LocoMail data file was covered in Session 5 of the tutorial. You should have worked through this session before attempting to create a LocoMail data file of your own.*

2.1 Setting up a data file specially

This section describes the steps involved in setting up from scratch a data file that you will be able to use with either one individual master document or with a group of master documents. We recommend that you plan and create your data files in the following order:

- decide on the items of information that are going to be included in each record, and the *item-names* you're going to give them
- choose the *separators* to go between the items
- choose the *terminator* to mark the end of the record pattern

When you have considered these points and created your record pattern, producing the remainder of the data file (ie. all the data records) with the 'real' data is then simply a matter of following this pattern.

2.1.1 Items and item-names

The first thing to consider when setting up your data file is what items of information you will want in each record, and what *item-name* to assign to each item in the record pattern.

LocoMail has a number of rules which you must follow when selecting item-names:

- they can only be made up of letters (A...Z), accented letters, the German Scharfes S (ß), numbers (0...9) and the underline character (_).
- they must start with a letter, an accented letter or ß
- they must not include spaces, characters like + , . £ / () etc., or hyphens
- only the first 16 characters of the item-name are important to LocoMail, so there must be a difference between similar item-names within the first 16 characters.
- lower case letters (a...z) are considered to be the same as upper case letters (A...Z) by LocoMail (ie. *date* is the same as *DATE*).
- the words *NOT*, *AND* and *OR* are special and must not be used as item-names

You should always try to choose item-names that describe the corresponding piece of information. The need for this will become much more apparent when you come to create a master document to use the data file, as described in Chapter 3.

If a two-word item-name seems more natural, join the two words with an underline character: for example, join the words *Full1* and *name* into the 'single' word *Full1_name*.

Note: You might notice from this that although we use the term *item-name* to represent the names you use, you couldn't actually use *item-name* because of the hyphen. You could, however, use *item_name* or *itemname*.

If you plan to use the data file with a number of different master documents, make sure that you consider all the uses you want to make of the different items of information. To make the data file as flexible as possible, you might have to split up information into a number of separate items and give item-names to each of these. You will have to include all the relevant items in the data file, even though not all of these will be used by every one of your master documents.

For example, you might be planning an address list data file that will allow you to produce labels such as:

Mr and Mrs H G Taylor
21 Acacia Avenue
Bartonwood

and yet still use the same record in the data file to produce a letter beginning:

Dear Henry and Liz, Robert and David

So to have all the different items of information available to the masters, the *item-names* in your record pattern would need to include:

Title	-	which in the record for the Taylors would be Mr and Mrs
Initials	-	which in the record would be H G
Names	-	which in the record would be Henry and Liz
Children	-	which in the record would be Robert and David
Surname	-	which in the record would be Taylor
Address	-	which in the record would be 21 Acacia Avenue Bartonwood

If you want to include extra pieces of information in the data file (ie. information that you don't necessarily see an immediate use for, but feel would be useful), remember to choose *item-names* for these as well.

Note: In some cases, you may have the master documents already prepared. When you are creating a data file to use with existing masters, you will need to look at the masters and note down every item-name used in the LocoMail instructions. You must then make sure that none of the item-names you decide on for your record pattern clash with the item-names used for different information in the master(s). Be careful, also, to note the spelling of each item-name; pay particular attention to plurals and to multiple words joined by underline characters. Don't put **Initial** in the record pattern, for example, when the item-name in the master document is **Initials** and don't write **Postcode** when the item-name used in the document is **Post_code**. However, swapping between capital and lower case letters doesn't matter.

When you've chosen the *item-name* to represent each item of 'real' information, you must then list these names in a suitable order. You don't have to keep to any particular order: in the above example, you could arrange the names in the record pattern as **Surname** then **Initials** then **Title** then **Address** then **Names** then **Children** if you wanted to, but this wouldn't make the data records very readable! Usually, whatever order seems the most obvious and sensible to you will be the best one.

2.1.2 Deciding on the separators

After you have considered which items of information will be included in the data file, and decided on the *item-name* for each piece of information, you have to work out what you are going to use to mark where one item ends and the next item begins. These markers are called *separators* – because they separate the items of information.

Because LocoMail needs to tell the difference between *item-names* and *separators* within the record pattern, you can't use letters or numbers as separators, but you can use any other character you can print – characters like +, ., /, (,) ! etc. In fact, separators may be any character except those which can appear in an item-name. The other things you can use as your separator are ↵ (a carriage return), and ↵↵ (two carriage returns). The ↵↵ separator has a particular role as we shall see.

Choosing which separator to use after a particular piece of information is simply a matter of selecting a character which:

- will never appear in the information itself
- will give each data record a layout that is clear

We put this second requirement in to make the data file as easy as possible for you to prepare and maintain. LocoMail isn't affected by how readable the data file is – just by whether the pieces of information and the separators are inserted correctly!

Where all the information is short – say, a few words or a number – a good choice of separator is something simple like a comma between each of the items (provided this isn't used in any of the pieces of information themselves). An alternative approach is to choose something appropriate to the information that follows the separator: for instance, you might use £ as the separator before a price or a salary amount.

For example, if the data file is a company's internal telephone list you might opt for commas:

Name, Position, Extension

because this will give you data records like:

Steven Arnold, Export Assistant, 251

and **John Armstrong, Product Manager, 569**

or, if you are setting up a data file with salaries, you might choose £ as the separator before each salary:

Name, position£salary

giving records like:

Steven Arnold, Export Assistant£8518

and **John Armstrong, Product Manager£17500**

You can further improve this by putting spaces or tabs immediately after each separator so that the entries line up. LocoMail automatically ignores leading spaces and tabs (ie. spaces and tabs before the information) when taking information from a data file.

Steven Arnold, → Export Assistant, → 251
John Armstrong, → Product Manager, → 569

Note, however, that spaces or tabs within or following an item of information are not ignored: they will be treated as part of the item.

Another simple choice is to make ↵ the separator for all the items – thus putting each piece of information on a new line. For example, you might be preparing a data file of children's books and decide to lay this out as:

Author ↵
Title

giving you data records like:

Edward Lear ↵

The complete nonsense of Edward Lear

and A A Milne ↵

Now we are six

In general, there are any number of combinations of layouts and separators you can choose. All you have to be careful about is that the separator after a particular piece of information is never used in the information itself.

The problem of multi-line information

The main difficulty in choosing a suitable separator comes when you want your data file to store multi-line pieces of information like addresses. For example, suppose you wanted your data records to look something like this:

Mr, James, Robinson ↵
1, The Mews ↵
Cottage Row ↵
Oldville ↵
0777 25824

ie. you want to lay out your data as follows:

Title, first_name, surname ↵
address ↵
telephone

However, a record pattern like this won't give you the right results, because of the ↵'s in the address itself. (LocoMail would think it had reached the end of the address after 1, The Mews.)

There are two possible solutions to this problem. You can either select as the separator a character like ; that you won't find in any of the addresses in the data file, or you can use the special separator ↵↵.

Replacing the ↵ immediately following the address by a ; will give you the pattern:

title, first_name, surname ↵
address;telephone

and data records like this:

Mr, James, Robinson ↵
1, The Mews ↵
Cottage Row ↵
Oldville;0777 25824

Alternatively, using the special separator ↵↵ as the separator after address will give you the pattern:

title, first_name, surname ↵↵
address ↵↵

telephone

and will make your data records appear like:

Mr, James, Robinson ↵↵
1, The Mews ↵↵
Cottage Row ↵↵
Oldville ↵↵
↵↵ 0777 25824

Either of these schemes will work.

When you have chosen suitable *item-names* for the items of information, put them in order and chosen suitable *separators*, all you need to do to complete your record pattern is to select a suitable *terminator* to mark the end of the record pattern.

2.1.3 Terminators and End-of-record markers

The record pattern *terminator* is the object that marks the end of the record pattern. This must be one of:

↓ (LocoScript's end-of-page character)
(Unit) (LocoScript's Unit Code)
/ ↵ (slash followed by return)

The ↓ and (Unit) are also end-of-record markers – ie. they can also be used between each of the data records, whereas the / ↵ is only a record pattern terminator. In general, when setting up your own data file, you will use ↓ or (Unit) – because then you can use them as end-of-record markers as well. / ↵ is usually only used when creating a record pattern for use with data produced by another program, as we'll see in Section 2.2.

The end-of-page character is usually the recommended choice both as terminator and as end-of-record marker, because:

- i) it puts the record pattern on a separate page – making it easy to read.
- ii) it puts each data record on a separate page – making it easy to read.

In our example, using the end-of-page character as the terminator gives you a record pattern looking like this:

```
title,first_name,surname↵
address;phone↵
```

Each data record should then follow this pattern exactly, finishing with an end-of-page character.

- *You are now ready to create your data file.*

2.1.4 Creating the data file

On the Disc Manager Screen, select the disc and the group that you want to store the data file in and then press **C** to create a new document in the usual way. It is important that you have a blank document to start with, so if the document is not blank when created – ie. it was created using a template which includes text – you should delete this text.

Start by typing in the record pattern you have just worked out, complete with the record pattern terminator.

Note: If you are planning to use the data records with a number of different record patterns (ie. you want to use the same data under a number of different item-names), stop when you have finished typing in the record pattern and save this separately. Similarly, if you want to use this record pattern as the pattern for a number of different data files, save it and create separate documents for each set of data records.

Next type in each set of information you want to put in the data file, carefully separating the different items of information within each set by your chosen separators. If you want, you can use spaces or tabs immediately after any separator to make your data file look neater. When LocoMail takes information from the data file, it ignores any spaces and tabs before the first character of an item.

However, you must only use 'white space' like this immediately after separators – spaces and tabs within or immediately following an item of information won't be ignored and will be inserted into the master as part of the information. For example, suppose your data file included two consecutive records, one with a tab following each separator, and one with a tab before the separator:

```
title,name,telephone↵
Dr,→ Helen White,→ 0888 25834↵
Mr→ ,Chris Brown→ ,755556↵
```

and your master document included the sentence:

You alone, (+Mail) name (-Mail), have been chosen...

When you merge this master with the data file, the two versions of the document will appear:

You alone, Helen White, have been chosen...

You alone, Chris Brown→, have been chosen...

As you can see, the tab before the items of information in the first data record has not been swallowed: the tab after the information in the second data record has not.

We suggest that you design your master documents so that they don't require any spaces after the pieces of information in the data file (ie. it is better to include any spaces you want in the master document, rather than the data file).

Follow the last item of each record with an end-of-record marker. As we mentioned above, the characters that LocoMail can accept as end-of-record markers are the end-of-page character ↵ or the Unit code (Unit), both of which can be used as record pattern terminators.

For example, if you have the following record pattern:

```
title,first_name,surname↵
address;phone↵
```

then you might have records like this:

```
Mr,James,Robinson↵
1,The News↵
Cottage Row↵
Oldville↵
England;0777 25824↵
```

or:

```
Ms,→ Jill,→ Smith↵
24 The Street↵
Blueberry↵
Wild West;→ 010-331-75676↵
```

if you were using tabs to spread out the data.

It isn't obligatory to use the same end-of-record marker throughout the data file – but for the sake of the data file's appearance it is best to choose one marker and use it both as the end-of-record marker and as the record pattern terminator.

- *When you have finished typing in all the information and making any corrections, save this in the usual way, ie. by pressing [EXIT] followed by [ENTER]. Assuming you've typed all the information correctly, your data file will then be ready to merge with LocoMail master documents. The next step is to test the data file – how to do this is described in Section 2.3.*

2.1.5 Special cases

In the preceding description of setting up data files, we have been dealing with cases where each record is quite straightforward – containing an item of information for each *item-name*. In this section, we will describe what you need to do when things aren't quite so simple.

Omitting items in records

When you are typing the data records into your data file, there will be times when you don't have information for every item in the record. At times like these, you can't just skip the entries altogether – you need to have some way of telling LocoMail that you've omitted the item. If you don't do this, LocoMail will match the wrong pieces of information with the item-names, and you will end up with the wrong details inserted into your finished document. We will start by describing the general case of missing out items of information from a record, then show you how you can omit a number of items at the end of a record. We also explain why you need to take extra care when omitting details at the start of a record.

• The general case

All you have to do, when you want to omit an item from a record you're typing, is to put the separator that normally finishes the information you're omitting immediately after the previous separator – or separated from it by 'white space' (spaces and tabs). As LocoMail reads through the record, it will find the separator signalling the end of the item, throw away any leading white space it finds and record that this particular item as 'null' (ie. no characters at all).

For example, you might be storing membership details for your society, using the record pattern:

```
title,initials,surname,telephone;age,group↓
```

Typical complete records will then appear:

```
Mr, S D, Twain, 221 6123; 35, bass↓
Ms, R S, Trott, 75284; 22, soprano↓
```

Whenever you want to omit an item of information – such as the age of a member, or their title, etc., you need to put in at least the separator that ends the piece of information you are omitting:

```
Mr, R M, White, 456 7890; , baritone↓
```

– where you don't know the age

```
, A, Smith, : 50, bass↓
```

– where you don't know the title or the telephone number

If you don't put the separator in the record, then LocoMail won't know that you've omitted the information. For example, in the second record, if you omit the comma that finishes the title item, LocoMail will take A as the title, Smith as the initials, and so on.

• Omitting the last few items in a record

When you have no more information to store in a record, you don't need to put a separator for each omitted item. You can simply tell LocoMail that the record is complete by putting in an end-of-record marker after you have typed the last item to be included.

For example, the membership details for your society might use the record pattern:

```
title,initials,surname,telephone;age,group↓
```

A typical complete record will then appear:

```
Mr, S D, Twain, 221 6123; 35, bass↓
```

If you just know a person's name, but no details of their title, telephone number, age or group, you can finish the record after the information you do have by using the end-of-record marker:

```
, A N, Brown↓
```

When LocoMail comes to interpret the data file, it will simply set the items of information that you didn't give in this record to null (ie. a complete absence of characters).

• When you are using ↓ as the first separator

There is one special case where you cannot indicate a missing item just by including the separator in the document. This is when you want to omit the first item in a record, and you are using ↓ (or ↓↓) as the separator after this item.

LocoMail ignores any carriage returns before the first item of information in a data record. So simply putting the ↓ separator when you have no information for the first item will not work, because LocoMail would ignore the ↓ and not realise that it signalled a blank item. What you need to do to stop LocoMail from ignoring the ↓s (and using the second item of information under the first *item-name*), is simply to put a space before the ↓ character. There is no risk of LocoMail inserting the extra space in a document, because it ignores leading spaces in any item. But LocoMail then knows there is a blank item at this point.

For instance, if your record pattern is:

```
title↓
initials↓
surname↓
```

then you must include either some information as the title or at least a space character if you don't know the details. If you leave it completely blank in any record, eg.

```
↓
```

```
H G↓
```

```
Rutherford↓
```

then LocoMail will ignore the first ↓, take H G as the title, Rutherford as the initials and set the surname to null.

2.2 Converting an existing data file

Note: The only data files you need to convert are ones prepared by a CPIM database or accounts program and lists of data that you have set up for your own use. LocoFile datafiles can be used as they are and do not need to be converted.

The job of converting an existing data file (for example, an ASCII data file set up by your accounts program) is chiefly one of sorting out how the information is divided up at present and then creating a record pattern that fits the existing data file. You can then use the *item-names* assigned to each piece of information by the record pattern in the master documents you create.

If you are creating the record pattern so that the data file may be used with existing master documents, you should look through the masters to see what *item-names* are used in the instructions where the information is to be taken from the data file. You will also need to make sure that any other *item-names* you use in the data file don't clash with other instructions in the masters (eg. ones which remember information typed at the keyboard).

An existing data file will already be split up into records and each record will itself be divided up into separate pieces of information. There are essentially three possible ways in which the items of information can be separated in this existing data file:

- In the simplest case, the items of data in each record are separated by a single character, and this character is one of LocoMail's acceptable separators. How to create a record pattern to use this data file is described in Section 2.2.1.
- In a more complicated case, the items of data are separated by two or more characters, but are still recognisable by LocoMail. Creating a record pattern to use this type of data file is described in Section 2.2.2.
- In the most complex case, the characters between items are not recognised as separators by LocoMail. In this case you will need to edit the existing data file and replace these separators with suitable characters - possibly putting in end-of-record markers as well. If you do have to edit the existing data file in this way, you might as well put an end-of-record marker after each record.

If you are lucky, the records may be divided by an ASCII form feed (FF) character - this is equivalent to LocoScript's end-of-page ↓. But, in general, the character(s) used between records won't be one of LocoMail's end-of-record markers. For example, a WordStar Mailmerge file has a CR [LF] at the end of each record which is equivalent to LocoScript's ↵ - a separator, not an end-of-record marker. So when you are creating a record pattern for data that has a separator after the last item in the record, you will need to include a *separator* after the last *item-name* in the record pattern. This will allow LocoMail to spot the end of the last item of information in each data record.

The record pattern will still need to be finished with a LocoMail terminator character (ie. a (UnitP) code, a ↓ or a / ↵ combination).

Using separators after every item in a record

As you have seen, you can tell LocoMail when a record is complete by putting an end-of-record marker after the last item in the record. However, there is another way LocoMail can tell when it reaches the end of a record - when it has counted the correct number of items.

To do this, LocoMail must know how many items there are in a record, and how each one of these items is finished. So the record pattern must include a separator after the last item, to tell LocoMail where that item finishes in the data records. For example:

```
title, initials, surname, telephone: age, group ↵
```

Here, the ↓ is used as a record pattern terminator. LocoMail knows from this record pattern that there are six items in a record, and it also knows that the sixth item in any record is finished with a ↵.

With a record pattern like this, you now have the choice of finishing each data record either with an end-of-record marker, or with a ↵ after the group information.

If you finish records with the separator, your records will appear:

```
Mr, → S D, → Twain, → 221 6123; → 35, → bass ↵
Ms, → R S, → Trott, → ; → , → soprano ↵
```

When LocoMail reads through these records, it will start a new record as soon as it has found six items. If you are using the final separator to mark the end of each data record, you must include all the separators in each data record regardless of whether some of the items are blank. LocoMail looks for the separators in sequence and must find every one before it can move on to the next record.

If you want to separate records with an end-of-record marker, your records will appear:

```
Mr, → S D, → Twain, → 221 6123; → 35, → bass ↓
Ms, → R S, → Trott, → ; → , → soprano ↓
```

The one thing you must not do is finish a data record with a separator after the last item and an end-of-record marker, because then LocoMail will 'see' an extra record - a completely blank one! So if you had the following in your data file:

```
Mr, → S D, → Twain, → 221 6123; → 35, → bass ↵
↓
```

LocoMail will see a complete record for Mr Twain once it has found the ↵ after bass. It then looks for the next record, finds an end-of-record marker and so sets all the items in this next record to null (ie. no characters at all). If LocoMail fills in a master using this data file you will end up with a version for Mr Twain followed by a version with nothing inserted in the blanks!

2.2.1 A simple data file

This section describes how you produce a record pattern to match the pattern of information in each record of an existing data file, where the data file has a simple structure:

data separator data separator ...

ie. the separators between items of data are single characters, and they are ones that LocoMail recognises. For example, a data file that has been prepared by an accounts program for use with WordStar's Mailmerge program typically has commas between the separate items of information. These characters can be readily picked out as separators by LocoMail.

The structure of the data records that LocoMail requires is, as you have seen:

item1 separator item2 separator item3 ...

These structures are the same. So creating a record pattern that will allow the data in the existing file to be used is simply a case of assigning suitable *item-names* to the data. These *item-names* can then be used in master documents.

The steps to creating a suitable record pattern are as follows:

- 1 Find out how the data file is laid out at present.

A good way of going about this is to print out the first few records of the data file. For example, you might get a print-out like this:

```
Jack Gold, 22 Dunbar Close, Oldville,, 76
Joe Smith, 1 The Cottages, Kingston, KT32 0XX, 2
```

Obviously, in this data file, commas are used to separate the pieces of information and each line is a separate data record. So the last item in each record finishes with another separator character (in this case a carriage return, which is not visible on the printout).

- 2 Give *item-names* to each piece of information in the data file. Remember to choose names relevant to the item of information so that your masters will be easier to follow. Give names to all the pieces of information in the data file, even if you don't intend to use some of them.

In this example, you might match names to the data records as follows:

name	addr1	addr2	postcode	rating
Jack Gold,	22 Dunbar Close,	Oldville,	Kingston,	76
Joe Smith,	1 The Cottages,		KT32 0XX,	2

The *item-names* you choose must obey the rules given in Section 2.1.1.

- 3 Your record pattern will then simply be these *item-names*, each followed by the *separator* character used in the existing data file, and with one of the following record pattern *terminators* at the end:

```
↓ (LocoScript's end-of-page character)
(UniT) (LocoScript's Unit Code)
/ ← (slash character followed by a return)
```

(If you are not going to type in the record pattern using LocoScript, but instead get your accounts program (say) to generate it, the choice becomes / ← or the ASCII FF (form-feed) character.)

In this example, the record pattern you use might be:

```
name, addr1, addr2, postcode, rating ←
/ ←
```

Either edit the data file and insert the record pattern at the beginning of the file or put this pattern in a separate document.

Save the data file (and record pattern if it's in a separate document). For convenience, give the record pattern a name which will place it next to the document holding the data records on the Disc Manager Screen.

- Assuming you've typed all the information correctly, your data file is now ready to use with a LocoMail master document. Before you use the data file in earnest, you should test it. How to do this is described in Section 2.3.

2.2.2 A more complicated data file

This section describes how to create a record pattern to use an existing data file with a more complicated structure, where more than one character is used to separate the items of data. A typical example of this type of structure is:

data separator separator data separator separator separator data...

What you need to do, to allow LocoMail to use the data from the data file, is create a record pattern that effectively imposes a different structure on this data:

item1 separator item2 separator item3 separator...

By comparing these two structures, you can see that the structures can be made to match by making the middle separator in each group of three an item in its own right. (If just two characters or more than three characters separate the items of data, then you make the structures match by keeping the two outer characters as separators and treating all the characters between as one item. In the case of two characters between the real data items, the item between the separators will always be blank.)

So in the record pattern we create, some *item-names* will correspond to pieces of data in the existing data file, and some will correspond to separator characters in the existing data file.

The way to create a suitable record pattern is essentially the same as the procedure given in Section 2.2.1.

The first step, as before, is to print out the first few records of the data file. A typical printout might appear:

```
"Peter", "Wilkins", "Director", "XJG"
"Bill", "Green", "Salesman", "XRG"
```

Here, each line is a separate data record – with the last item in each record a carriage return (which can't be seen on the printout, but is shown above for convenience).

The next step is to assign an *item-name* to each useful item of information. Each of these items must be followed by a single character separator which it is useful to pick out at this stage. From the printed out data records, you can see that the " character immediately follows each useful piece of information. So at this stage, you might have:

```
name1      name2      position      car
" Peter"   " Wilkins" " Director"  " XJS"
" Bill"    " Green"   " Salesman" " XR2"
```

You now need to assign *item-names* to the remaining characters (again remembering that each item must be followed by a separator for the data record to have the correct structure). The *item-names* you choose at this point will apply to pieces of information you don't need to use in your masters – so typically you might use names like dummy1, dummy2, etc.

```
dummy1 name1 dummy2 name2 position dummy3 dummy4 car dummy5
" Peter" " Wilkins" " Director" " XJS"
" Bill" " Green" " Salesman" " XR2"
```

Here, dummy1 represents null data (before the first " separator), dummy2..dummy4 all represent commas, and dummy5 represents null data finished with a return character. This arrangement leaves name1, name2, position and car representing the useful items of data.

Your record pattern will then simply consist of all these *item-names*, together with the separators, and with one of LocoMail's record pattern terminators at the end.

In this example, the record pattern would be:

```
dummy1"name1"dummy2"name2"dummy3"position"dummy4"car"dummy5↵
```

This record pattern will let you use the *item-names* name1, name2, etc. in your masters and get the right results. As in Section 2.2.1, the final steps are to save the record pattern, either at the start of the data records or in a separate document if necessary.

- Assuming you've typed all the information correctly, your data file is now ready to use with a LocoMail master document. Before you use the data file in earnest, you should test it. How to do this is described in Section 2.3.

2.3 Testing the data file

When you have created a new data file, it is a good idea to test it to make sure that it will give you the results you require. The only real way to test a data file properly is to merge it with one of the master documents you intend to use it with (of course, you'll have to be sure that this master is error-free first!). Alternatively, you could quickly create a special master document with a number of very simple instructions to fetch each item in a record. For example, if the record pattern of the data file you have created consists of the *item-names* Name, Address and Telephone, then your very simple master could be:

```
Name is:↵ (+Mail)name (-Mail)↵
Address is:↵ (+Mail)address (-Mail)↵
Phone is:↵ (+Mail)telephone (-Mail)↵
```

(Creating a master document and testing it are covered in Chapter 3.)

To merge your master document with the data file, display the Disc Manager Screen. Move the File cursor to the master document you've created and press M; this selects both the master document and the M=Merge option. Then move the File cursor to your data file and press ENTER.

Note: If the record pattern and the data records are in separate files, move the File cursor first to the document holding the record pattern and press ENTER, then when you are prompted move it to the file holding the data records and press ENTER again.

Check that you have selected the right document on the Selection menu that appears, set the print options to Draft quality and one copy. When testing your data file, it is best to carry out a Manual merge as this allows you to see what is going on more easily, and abandon the merge if things go drastically wrong. Manual is selected for you, so just press ENTER.

Assuming the data file has a valid record pattern, LocoMail will then display the master document, and start to work through it, filling in the 'blanks'. (If the data file doesn't contain a valid record pattern, you'll have to abandon the merge and sort this problem out first.)

At each 'blank' where the information is to come from the keyboard, you will need to type something (or at least press ENTER). What you must check as LocoMail progresses through the document is that the 'blanks' which require information from the data file are being filled with the right information.

If there are no serious errors in the first pass, and you see that all the details are being inserted in the right places, then continue filing versions to test all the records in your data file. Even if things go slightly wrong, keep producing versions for as long as you can. This will allow you to see the full extent of the problems with the data file.

It is important, when checking for errors, to note whether they occur in every version or just particular versions of the document. If LocoMail completed a number of versions correctly before the problem arose, you will know which particular record caused the difficulties and you can immediately check that record. If the problem was more general in nature (such as taking the wrong information for every version) then the problem is more likely to be in the record pattern.

If you are using a simple master such as one described above, you would expect each version to include all the details from the current data record. Any errors which arise at this stage will show up any mismatch of item-names between the master and the record pattern of the data file, or a problems with the separators, terminator and end-of-record markers in the data file – normally just typing problems!

If the merge is going so wrong that you need to stop – or if LocoMail stops and displays an Alert message – select **Abandon LocoMail** (either from the Alert message box, or from the menu that appears when you press **STOP** then **STOP** again). You are then returned to the Disc Manager Screen, from where you can edit the data file or master as necessary.

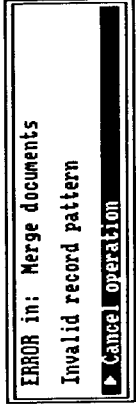
Keep merging the documents, sorting out errors in your record pattern and data records from the descriptions below, until you can successfully produce a version for every record in the data file. The data file is then ready for normal use.

2.3.1 Errors in the data file

The most common errors that you might encounter whilst merging the data file with the master document fall broadly into two categories: those that cause LocoMail to stop and display an Alert message, and errors that cause the wrong information to be inserted without causing LocoMail to stop.

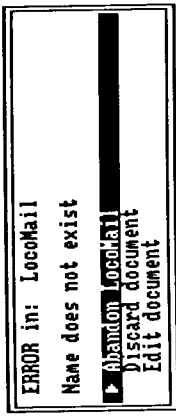
The first group of errors we look at are ones that result in an Alert message:

- LocoMail displays the following Alert message:



This message indicates that your data file doesn't have a valid record pattern as its first record. If the data file has no record pattern at all (ie. it's in a separate document), start the merge off again, selecting the file holding the record pattern before the file holding the data records. Alternatively, you may have an error in the record pattern itself (eg. two terminators, or no terminator). Check that the pattern follows the structure described in Section 2.1.

- LocoMail displays the following Alert message:



This message means that the current instruction asks for information to be taken from the data file, but the *item-name* doesn't exist in the record pattern. Typically, you will have misspelled the *item-name* either in the record pattern or in the master document. Correct the *item-name* so that the name in the master document and the name in the data file are identical. Normally, the name to change will be the one in the record pattern, but if you have used this data file successfully with other existing masters, it is probably easiest to change the name in the instruction in the master.

- A full description of all the Alert messages, their probable cause and how to solve the problem is given in Appendix IV.

The second group of errors insert the wrong information:

- LocoMail produces a filled version correctly, then produces a version with nothing in the blanks

You have probably marked the end of data records with both a separator after the last item and an end-of-record marker. You need one or the other, not both.

- Only the first line of an address is inserted into an address 'blank'

You have probably used a **↵** as the separator after address in your record pattern. LocoMail then thinks it has come to the end of the address after the first line. Choose a different separator to go after **address** in your record pattern and modify your data records to match.

- The wrong items of information are being inserted into 'blanks'

You have included (or omitted) separators in the record pattern or the data records so that the two do not correspond, causing LocoMail to associate the *item-names* with the wrong pieces of information. Alternatively you may have chosen your separators badly, so that LocoMail is not retrieving all of an item of information. Check the use of separators carefully (referring to Section 2.1 if necessary).

If this happens with every version of the document, then the problem probably lies in the record pattern. If it happens only with certain versions, the problem probably lies in the relevant record(s).

Planning and preparing the master document

A master document is simply a LocoScript document that contains all the LocoMail instructions needed to produce the finished documents you require.

Except for these LocoMail instructions, a master document is no different from any other LocoScript document. All the layouts, character styling and word-processing features of LocoScript 2 can be put into your master document, and whatever styling you include in the master will go into each finished version as well. So preparing the master document is essentially the same as preparing any other LocoScript document, except that you put in the LocoMail instructions where you want the finished versions to include specific details.

This chapter is designed to help you in creating your masters, and describes:

- the steps involved in planning your master document
- the LocoMail instructions which can be used to mark the 'blanks'
- how to put the instructions into the master
- how to make the master more readable

The instructions we will describe in this chapter are the ones that are used most often, and are the most straightforward to follow and apply – instructions that fetch information and insert it to produce finished documents. There are a whole range of LocoMail special instructions to do more complicated jobs, such as arithmetic, numbering paragraphs, marking sections of the document as optional, etc. – these instructions are described in Part III. Here, we concentrate on using the simpler LocoMail instructions.

When you have created your master document, the next step is to test it thoroughly by filling it in LocoMail's Fill mode. How to do this, and how to correct the most common errors you might find in the master, is described in Section 3.5.

- *Preparing a master document was covered in Session 4 of the tutorial. You should have worked through the tutorial session before continuing with this chapter.*

3.1 Planning the master

The most important part of preparing a master document is planning:

- planning the overall look of your document
- working out which details you will want to change from version to version
- planning how you will supply LocoMail with the information to put in the 'blanks' – by using a data file or by typing in at the keyboard
- working out exactly which LocoMail instructions to use at each 'blank'

The overall look of the document is nothing to do with LocoMail – that is handled by LocoScript, and so you approach it exactly as for all LocoScript documents. Planning how you will supply the information, and choosing the instructions to use, are where you need to think about how LocoMail is to do the job.

If the information that you want is already stored on disc, think about how the information is stored, under what *item-names*, etc. If you intend to use the same master with a number of different data files, then you must ensure that the *item-names* for similar information in the different data files are identical (ie. that a person's name isn't stored as *name* in one data file and *full_name* in another). If this sort of thing is a problem, you may find it easier to produce a number of masters to use the different data files. Alternatively, you can change the *item-name* in the record patterns (and the corresponding *item-names* in the other masters) if there aren't too many masters using this information.

Note: Careful thought during the early stages of producing a master will make creating and then using the document far easier. As you become more experienced with LocoMail and tackle more ambitious projects, designing the masters properly becomes crucial. Chapter 11 in this User Guide describes the best ways to go about designing more complex masters.

Creating the master

You can create your master document from scratch – typing in text where you want text and including LocoMail instructions where appropriate. But in general, you'll find preparing the master easiest if you start from an existing LocoScript document that matches the sort of document you want LocoMail to produce from the master. Not only does it help you to sort out the details you will want to change from version to version (name, address, etc.), but this document can be readily converted into your master document by replacing those details with the corresponding LocoMail instructions.

For example, starting with a LocoScript document like this:

```

(John)
(John)
(John)
(John)
(John)
Dear Mr Smith,

Thank you for your letter of 5 July. I am extremely sorry to
learn that the Superfac is not working correctly. I can assure
you that our customers have reported very few problems with
this product.

May we suggest in the first instance that you take your
Superfac back to our York branch for servicing. Ask to speak to
the Service Manager. He has been advised of the problems you
have been experiencing.

Yours sincerely,

Joan Smith
Customer Services Manager
  
```

11 July 1984

The Widget Company
New Industrial Estate
Little Meadow
Morestrey

(John)
(John)
(John)
(John)
(John)
Dear Mr Smith,

Thank you for your letter of 5 July. I am extremely sorry to
learn that the Superfac is not working correctly. I can assure
you that our customers have reported very few problems with
this product.

May we suggest in the first instance that you take your
Superfac back to our York branch for servicing. Ask to speak to
the Service Manager. He has been advised of the problems you
have been experiencing.

Yours sincerely,

Joan Smith
Customer Services Manager

Joan Smith
Customer Services Manager

you can easily create a LocoMail master suitable for filling like this:

```

(John)
(John)
(John)
(John)
(John)
Dear (Name),

Thank you for your letter of (Date). I am extremely sorry to
learn that the (Product) is not working correctly. I can assure you that our
customers have reported very few problems with this product.

May we suggest in the first instance that you take your (Product)
back to our (Branch) branch for servicing. Ask to speak to
the Service Manager. He has been advised of the
problems you have been experiencing.

Yours sincerely,

Joan Smith
Customer Services Manager
  
```

The Widget Company
New Industrial Estate
Little Meadow
Morestrey

(John)
(John)
(John)
(John)
(John)
Dear (Name),

Thank you for your letter of (Date). I am extremely sorry to
learn that the (Product) is not working correctly. I can assure you that our
customers have reported very few problems with this product.

May we suggest in the first instance that you take your (Product)
back to our (Branch) branch for servicing. Ask to speak to
the Service Manager. He has been advised of the
problems you have been experiencing.

Yours sincerely,

Joan Smith
Customer Services Manager

The Widget Company
New Industrial Estate
Little Meadow
Morestrey

(John)
(John)
(John)
(John)
(John)
Dear (Name),

Thank you for your letter of (Date). I am extremely sorry to
learn that the (Product) is not working correctly. I can assure you that our
customers have reported very few problems with this product.

May we suggest in the first instance that you take your (Product)
back to our (Branch) branch for servicing. Ask to speak to
the Service Manager. He has been advised of the
problems you have been experiencing.

Yours sincerely,

Joan Smith
Customer Services Manager

The Widget Company
New Industrial Estate
Little Meadow
Morestrey

(John)
(John)
(John)
(John)
(John)
Dear (Name),

Thank you for your letter of (Date). I am extremely sorry to
learn that the (Product) is not working correctly. I can assure you that our
customers have reported very few problems with this product.

May we suggest in the first instance that you take your (Product)
back to our (Branch) branch for servicing. Ask to speak to
the Service Manager. He has been advised of the
problems you have been experiencing.

Yours sincerely,

Joan Smith
Customer Services Manager

The Widget Company
New Industrial Estate
Little Meadow
Morestrey

(John)
(John)
(John)
(John)
(John)
Dear (Name),

Thank you for your letter of (Date). I am extremely sorry to
learn that the (Product) is not working correctly. I can assure you that our
customers have reported very few problems with this product.

May we suggest in the first instance that you take your (Product)
back to our (Branch) branch for servicing. Ask to speak to
the Service Manager. He has been advised of the
problems you have been experiencing.

Yours sincerely,

Joan Smith
Customer Services Manager

If you don't already have a LocoScript document similar to the LocoMail master document you want to produce, create one. (With the Disc Manager Screen displayed, move the File cursor to the drive and group in which you want to store the document and press C to create a new document. Add text, layouts etc. just as with any other LocoScript document.)

When preparing the sample version, use either dummy or 'real' names, addresses etc. where you will want this information to appear in the finished versions. Remember that the final versions produced by LocoMail from the master document will look exactly like this LocoScript document. In particular, ensure that all the details in the Document Setup (printer, paper, etc.) are correct and that all the word-processing codes you require are in this LocoScript version.

For certain types of LocoMail master document, this could mean creating a new paper type specially for your master document. For example, you may need to set up a special paper type if your master document will be used to print address labels. How to set up new paper types is described in Session 19 of the LocoScript 2 User Guide (or Session 20 of the PCW9512 User Guide). You can also see it in practice in Chapter 14 of this LocoMail User Guide, where we give a full worked example of setting up a master document for printing on continuous 'labels' stationery.

Picking out the details to change

The first stage of converting the existing LocoScript document into a LocoMail master document is to work out which details you will want to change in order to 'personalise' each version. These will become the 'blanks' that you will use LocoMail to fill. For example, if you want to prepare a standard letter, you will probably want to change the name of the person the letter is addressed to, the address and the date.

Work through a print-out of the LocoScript document, marking these parts. Mark specially those items of information that appear more than once in the document (such as the name, perhaps), and those items like the date where the information will be the same for all versions you produce in one batch. Your marked up document might then appear:

The Widget Company
New Industrial Estate
Little Meadows
Worcester

11 July 1988

Date used every version

Dear Mr Smith

Thank you for your letter of 5 July. I am extremely sorry to learn that the SuperVac is not working correctly. I can assure you that our customers have reported very few problems with this product.

May we suggest in the first instance that you take your SuperVac back to our York branch for servicing. Ask to speak to the Service Manager. He has been advised of the problems you have been experiencing.

Yours sincerely
Product used twice in the same document

Joan Smith
Customer Services Manager

Now decide on the *item-names* to represent each different piece of information. Of course, if some of these details will be taken from a data file that you've already created (such as a *name* or *address* from an address list data file), you will already know some of the item-names. For others, you will need to choose new names. Be sure not to use any item-name which has been used for a different piece of information in the data file.

Each item-name must follow the rules given in Section 2.1.1.

The item-names you choose will probably be the standard words for the type of information you will want to insert - for example, *date* or *address* or *name*. Simple words like these should always be used if possible. However, if it seems more natural to describe the item by two words, then you can join the two words with an underline character - for instance, using the words *local branch* in the above example (rather than *localbranch* which is difficult to read, or just *branch* which is less descriptive).

Remember that you don't have to work out a separate *item-name* for each line of a multi-line piece of information such as an address. You can just use one *item-name* to represent all the information. However, if you intend to use the different lines of this information at different points in the document, or to present the information in a different order, then it is better to choose item-names for each line.

When you have worked out which items are to change in each LocoMail version, and decided on the item-names to refer to each item, the next step is to decide which LocoMail instructions are best suited to each particular 'blank'.

What you need to decide for each 'blank' is:

- where will the information come from? (ie. fetched from a data file, or typed in at the keyboard)
- does the information appear just once, or is it used several times in the document? (ie. do you want LocoMail to remember it for use later)
- do you want the same typed-in piece of information to appear in other versions of the document?
- is the information a number, and if so would you like LocoMail to check that is what you've typed?

IMPORTANT: The end of lines and the ends of pages in your filled documents are very likely to be different from the ones in your master document. LocoScript can ensure that every filled document you produce is well laid out, but to take advantage of this you have to make use of LocoScript various facilities for controlling line breaks and page breaks - Keep codes, in particular. These are described in Session 15 of the LocoScript 2 User Guide and in Session 16 of the PCW9512 User Guide. You also need to check that you don't have any spurious carriage returns at the ends of lines in the middle of paragraphs. The only carriage return you need in a paragraph is the one at the end!

- Once you have worked these things out for each of the 'blanks' in your LocoScript document the next step is to replace the marked-up text with the LocoMail instruction that will fetch and use the information in the correct way.

3.2 Putting in the LocoMail codes for marking blanks

All LocoMail instructions begin with a (+Mail) code and finish with a (-Mail) code. Between these codes come the details that actually tell LocoMail what to do. So each 'blank' in your marked-up LocoScript document will need to be replaced with an instruction of the form:

(+Mail) *instruction* (-Mail)

The (+Mail) and (-Mail) codes that surround the *instruction* effectively tell LocoMail 'everything between these codes is an instruction to be carried out' - so when LocoMail is processing your master document, it does what the *instruction* says. Everything in the master document that isn't enclosed between a (+Mail) and a (-Mail) code - be it a character, code, text, etc. - is treated as ordinary LocoScript text and will appear in the finished document.

In this description, we have used *instruction* to represent the sequence of characters, text, etc. that tell LocoMail exactly what to do.

This section describes how to insert the (+Mail) and (-Mail) codes into your document in place of the marked up sections of text; Section 3.3 describes the instructions you can use to make LocoMail do what you want, and at the end of Section 3.3 there are two examples of choosing and inserting the correct LocoMail instructions for the 'blanks' in the MASTER.EG1 document

To insert the LocoMail instructions into your LocoScript document, you need to edit the sample document (or a copy of it). So, with the Disc Manager Screen showing, move the File cursor to the document that is to be the master, and press **F** then **ENTER**. In order to see exactly what you are doing as you add the LocoMail instructions to the document, it is better to work with LocoScript codes displayed. If necessary, press **F8** to select the Options menu, then press **F** to set the Codes option and press **ENTER**.

Move in turn to each of the places in the document that you have marked-up on the printout. Use the **DEL** or **DEL** keys to delete the dummy information at this point. Be sure to delete just those characters that you don't want to appear in any of the finished versions. In particular, don't delete the space before or after the dummy information. If you do, when LocoMail inserts the information at that point you may find that the spacing in the finished document is unsightly.

When the dummy information is deleted, insert a (+Mail) code. This can be entered from the Set menu, in the same way as you would enter any other LocoScript word-processing code that begins with a '+' (such as (+Bold) or (+UL), etc.): you can either

- press **F** then **M**, or
- press **F**, select Mail from the Set menu when this appears and press **ENTER**

When the (+Mail) code appears on screen, type the instruction itself as if it were text. What you type depends on what you want LocoMail to do at that point - details are given in Section 3.3 (and in Part III).

When you've entered the details correctly, finish the instruction with a (-Mail) code. Again, this is entered just like any other LocoScript code that starts with a -; either press **F** followed by **M** or press **F**, select Mail from the Clear menu when it appears and press **ENTER**.

Note: You will see that the details you type between the (+Mail) and (-Mail) codes are highlighted on the screen: this is because one of the effects of the Mail codes is that of (+Rev) and (-Rev).

This is the method you should follow at every 'blank' in your LocoScript document - deleting the dummy information, then inserting the (+Mail) code, the necessary details and the (-Mail) code. What you need to know next is what details to insert between these codes at each 'blank' - the LocoMail instructions.

3.3 LocoMail instructions

This section describes the LocoMail instructions that take in information from a data file or the keyboard. There are a range of different LocoMail instructions, as each one handles information in a different way.

The details that make up the instruction between the (+Mail) and (-Mail) codes tell LocoMail:

- whether the information is to come from the keyboard, or from a data file or memory
- whether the information typed at the keyboard is to be stored in LocoMail's memory or discarded
- whether the information is to be carried over from one pass through the document to the next
- what *item-name* the information is to be stored under (if any)

(There are other instructions which tell LocoMail to carry out some more complex tasks. These commands are covered in Part III.)

Each type of instruction has a different format. However, there are many similarities, as you will see. These similarities make it easy to change a master document as your needs change.

As well as the details telling LocoMail what to do, each instruction can include a semi-colon and some additional text at the end of the instruction. This text is useful as a comment, to remind you of the purpose of that instruction as you look at the document. If LocoMail needs to take information from the keyboard, then LocoMail will use this text as a prompt, to tell you what to type.

The semi-colon and prompt or comment text is optional, and can be left out if you wish. We show this in the following sections by surrounding it with slanting

square brackets eg. `[; prompt]`. Later, we will leave it out altogether: simply remember that you can include `;text` as and when you need, and that the text can say whatever you feel would be useful at that point in the document – either as a prompt or as comment. Note that you don't type the slanting square brackets in your instructions.

In addition to this convention of slanting square brackets to indicate optional items, we have used *item-name* in the instructions to show you where you will have to put a real item-name. The item-name you put in the instructions can be any name you choose, so long as it obeys the rules given in Section 2.1.1. If the information you want to use is stored in the data file, the item-name has to be the one given in the data file's record pattern. If the information you want is to be typed in at the keyboard, then you can use any name other than the ones used in the data file's record pattern, even names that you have used earlier in the same document – though you must be certain that you don't need the 'old' information any more.

The full range of actions you can carry out, and the instructions you need, are given below. We also give examples of when and where you might use these instructions in your master document. For convenience, the instructions are described in three groups:

- instructions that take information from a data file
- instructions that take information from the keyboard
- instructions that set up information for later use

3.3.1 Instructions that take information from a data file

LocoMail has just one type of instruction for taking information from a data file stored on disc:

```
(+Mail) item-name [ ; comment ] (-Mail)
```

and you use this instruction (substituting the appropriate *item-name*) wherever LocoMail is to fetch the piece of information from your data file.

Remember that the *item-name* used in the LocoMail instruction must exactly match an *item-name* in the record pattern of the data file that you will merge the master document with. The record pattern, as we explained in Chapter 2, tells LocoMail how to pick out the information from the rest of the data record.

For example, you might have a data file made up of **title, initials, surname, address and postcode** – ie. these are the *item-names* specified in the file's record pattern. The LocoMail instructions you would insert in the master document where you wanted to use these pieces of information might then be:

```
(+Mail)title;Mr or Mrs or Ms (-Mail)
(+Mail)initials(-Mail)
(+Mail)surname(-Mail)
(+Mail)address;full address excluding postcode (-Mail)
etc.
```

3.3.2 Instructions that take information from the keyboard

When you want to insert items of information that aren't stored in your data file, you need a LocoMail instruction that asks for the information to be typed in at the keyboard. There are quite a number of these instructions – which one you use at any point will depend on what you want the master document to do with the details you type. For example, you might just want LocoMail to insert the details, or you might want it to insert the details and remember them later in the document, or even to carry them over from one version of the master to the next.

• When the details that will be typed are wanted only at this point in the document and will change from version to version:

```
(+Mail)?[ ; prompt ] (-Mail) if any type of information is acceptable
(+Mail)?# [ ; prompt ] (-Mail) if only numbers are acceptable
```

For example, if your standard letter has a single slot for the recipient's full address, you might mark this blank with the instruction:

```
(+Mail)?;full address including postcode (-Mail)
```

Notice that you don't have to mark out a separate line for each line of a multi-line piece of information such as an address. If you have stored the whole address under one *item-name*, you just use this name.

• When the details that will be typed are wanted here and at a number of other points in the document:

At the point where the information is to be typed:

```
(+Mail)?item-name [ ; prompt ] (-Mail) if any type of information is acceptable
```

```
(+Mail)?#item-name [ ; prompt ] (-Mail) if only numbers are acceptable
```

Subsequent slots:

```
(+Mail) item-name [ ; comment ] (-Mail)
```

Notice that the text after the semi-colon in the first type of instruction is a *prompt* which LocoMail displays on the screen, and that in the second instruction it is a *comment* that you can include to make it easier to remember what information this instruction will insert. There is no need for these pieces of text to match.

For example, a travel agent might want to put a customer's choice of holiday destination (Ibiza, Benidorm, North Pole, etc.) more than once in the letter confirming the booking. The first slot might then be marked:

```
(+Mail)?dest;Holiday destination? (-Mail)
```

and the subsequent slots where this information is to be inserted:

```
(+Mail)dest;holiday destination (-Mail)
(+Mail)dest;resort (-Mail)
(+Mail)dest (-Mail)
```

Chapter 3: Planning and preparing the master document

Similarly, if you wanted to insert the price of an item into your document, and use it again later in the document:

```
(+Mail)? unit_cost ; Price of item? (-Mail)
and the subsequent slots where this information is to be inserted:
(+Mail)unit_cost (-Mail)
```

• When the details are being collected together at the start of the document, for use later in the same document:

At the point where the information is to be typed:

```
(+Mail)item-name = ? { ; prompt } (-Mail)
```

– if any type of information is acceptable

```
(+Mail)item-name = ?# { ; prompt } (-Mail)
```

– if only numbers are acceptable

Subsequent slots:

```
(+Mail)item-name { ; comment } (-Mail)
```

For example, you might have an initial information-gathering block of instructions that ask you to type in a customer's name, address and the product they bought, and the price. This might be:

```
(+Mail)name=?; Customer's full name (-Mail)
(+Mail)address=?; Customer's address (-Mail)
(+Mail)product=?; Product purchased (-Mail)
(+Mail)unit_cost=?; Price of item? (-Mail)
```

and you would then use instructions like:

```
(+Mail) product (-Mail)
(+Mail) unit_cost (-Mail)
(+Mail) address (-Mail)
```

and where you wanted to insert the details.

• When the details are to be entered once and used in all versions:

At the point where the information is to be typed:

```
(+Mail)?item-name { ; prompt } (-Mail)
```

– if the information is to be inserted at this point in the document

```
(+Mail) item-name = { ; prompt } (-Mail)
```

– if it is simply to be stored for insertion later in the document

Note: again, the # may be included after the ? in either type of instruction, if only numbers are acceptable.

Subsequent slots:

```
(+Mail) item-name (-Mail)
```

– if it needs to be inserted in another part of the document
(note: no exclamation mark)

In other words, to arrange that you can use the information in all versions of the document, you simply add a ! in front of the standard instruction to gather the information and store it.

Chapter 3: Planning and preparing the master document

For example, to save yourself effort by only entering the date once, the instruction to use is:

```
(+Mail)!date;today's date (-Mail)
```

If you have an initial information-gathering block of instructions, you could instead have the instruction:

```
(+Mail)!date=?;today's date (-Mail)
```

with a (+Mail)date (-Mail) instruction where you wanted the date to appear.

Note: You can, of course, enter numbers in response to any of LocoMail's instructions. The ?# type of instruction just allows you to make certain that the information entered is numeric, and you should use it whenever LocoMail will use what you type in some sort of arithmetic. The ability to work with numbers (ie. 'numeric' information) greatly increases the range of jobs you can carry out using LocoMail, but it is outside the scope of this general introduction to planning and creating a master document. In Part III you will see more about how you can use 'numeric' information, and what you can use it for.

3.3.3 Instructions that set up information within the master document

The third possible source of information for details to be inserted into blanks is when you set up the details within the LocoMail master document itself. The instruction to use is:

```
(+Mail) item-name=number (-Mail)
```

– if you want to store a pure number in item-name

```
(+Mail) item-name="text" (-Mail)
```

– if you want to store text (ie. a string of characters) in item-name

```
(+Mail) item-name=item-name2 (-Mail)
```

– which copies what is currently in item-name2 and stores it in item-name1

You can then use (+Mail) item-name (-Mail) in the master wherever you need to insert the number or the text.

The first two of these instructions would normally be used where you want to set up a value that you won't need to change very often. For example, you might have an instruction at the start of a master invoice, saying:

```
(+Mail) vat=15 (-Mail)
```

because the vat rate used in the calculations in the invoice is 15%. If, in the future, the vat rate changed it would be much easier for you just to alter this one line of the master document than to go through each of the places in the master affected by the change.

Setting up values within the master document is most useful in some of the more complex tasks that you can use LocoMail for, such as renumbering paragraphs, or arithmetic, as described in Part III.

The last of these three instructions can be used wherever you would like to change an item of information stored in the data file before inserting it in the document. LocoMail won't allow you to change any information stored under an *item-name* that refers to information in the data file – but you can use this instruction to copy it to another *item-name* and then work with the new name.

Examples of choosing suitable instructions

The SAMPLE.EG document, shown in Section 3.1, can be used as an example of choosing instructions to suit each 'blank', and typing them in. The marked-up version of this letter is:

The Widget Company
New Industrial Estate
Little Meadows
Worcester

11 July 1988

Dear Mr Smith

Thank you for your letter of 5 July. I am extremely sorry to learn that the SuperVac is not working correctly. I can assure you that our customers have reported very few problems with this product.

May we suggest in the first instance that you take your SuperVac back to our York branch for servicing. Ask to speak to the Service Manager. He has been advised of the problems you have been experiencing.

Yours sincerely

Joan Smith
Customer Services Manager

To show the use of a variety of instructions, we will take the name information from the data file (using the type of instruction shown in Section 3.3.1), and supply all the other information from the keyboard (using instructions of the type described in Section 3.3.2). We will show two different approaches to collecting the information. In the first example, we will use instructions that ask for and insert the information at each marked-up 'blank': in the second, the information will be collected using a group of instructions at the start of the master, with separate instructions to insert it.

I: Instructions that fetch and insert details:

As described in Section 3.1, we need to examine the requirements of each 'blank' in turn

11 July 1988 – Today's date only appears once in the document, but it is to be carried over and used in each version in the current batch. The information needs to be stored from one pass to the next, so the instruction that is required is:

(+Mail) !?date; Today's date? (-Mail)

Mr Smith – The name of the intended recipient only appears once in the document, and it will be taken from the data file. So the instruction to use is simply:

(+Mail) name (-Mail)

5 July – The date of the customer's letter is only used once in the letter, and will be different for each version. So the instruction required is simply:

(+Mail) ?; Date of customer's letter? (-Mail)

SuperVac – The product name appears twice in the current document, and it will change from one letter to the next. So the instruction you need to mark the first point it appears needs one that will insert the details and remember them:

(+Mail) ?product; Product purchased? (-Mail)

Once the information is stored, it can be inserted at the second point automatically with the instruction:

(+Mail) product (-Mail)

York – The local branch only appears once in the document, and it will change from version to version. So it too is marked by the simple instruction similar to the one used for the date of customer's letter.

(+Mail) ?; Local branch? (-Mail)

Work through a copy of the SAMPLE.EG document, replacing each of the marked-up pieces of text with the appropriate instruction. For example, the 'blank' for Mr Smith would be replaced as follows:

- delete the words Mr Smith (be careful not to delete the space before Mr)
- press ☐ then M to insert the (+Mail) code
- type name
- press ☐ then M to insert the (-Mail) code

Chapter 3: Planning and preparing the master document

When you have typed in all your instructions, your screen should appear:

(Align) The United Company
(Align) New Industrial Estate
(Align) Little Wymondley
(Align) Hereford
#

(Mail) [redacted] date: [redacted] (Mail) #

Dear (Mail) [redacted] (Mail) #

Thank you for your letter of (Mail) [redacted] date of customer's letter (Mail). I am extremely sorry to learn that the (Mail) [redacted] product is producing an extremely poor result. I can assure you that our customers have reported very few problems with this product. #

May we suggest in the first instance that you take your (Mail) [redacted] (Mail) back to our (Mail) [redacted] (Mail) branch for servicing. Ask to speak to the Service Manager; he has been advised of the problems you have been experiencing. #

Yours sincerely #

#

Joan Smith
Customer Services Manager #

Save the resulting master document if you want.

III: An alternative approach – gathering the information at the start of the master

Rather than scatter the instructions that ask for information throughout the master, you might prefer to have an initial group of instructions at the start of the master, to collect up and store all the typed information required for the current version. Then you could mark each 'blank' in the master with a simple instruction to insert the correct information from memory. (Of course, you will still need the instruction to fetch the name from the data file.)

To collect up each item of information needed in the letter and store it, without inserting it, we need instructions of the type:

```
(+Mail) item-name=?; prompt (-Mail)
```

So you can use this type of instruction, with suitable *item-names*, to collect up most of the information:

(+Mail)product=?;Product purchased?(-Mail)
(+Mail)letter_date=?;Date of customer's letter?(-Mail)
(+Mail)local_branch=?;Local branch?(-Mail)

Each of these instructions will prompt for information, then store what you type. As you want today's date to be carried over from one pass to the next, the instruction to fetch and store this item of information needs to include an exclamation mark:

(+Mail)!today=?;Today's date? (-Mail)

Your document will start with these four instructions.

Chapter 3: Planning and preparing the master document

The only other piece of information that is required is the name – and this will be fetched from the data file. So the simple instruction to fetch this will be:

(+Mail) name(-Mail)

Each of the marked-up "blanks" in the master document (ie. SuperVac, York etc) is then marked with instructions of the form:

(+Mail)item-name(-Mail)

Putting these instructions into a copy of the SAMPLE.EG document gives:

[illegible]

Looking at this version, you will see how the master document becomes clearer when most of the details are fetched using a block of these information-gathering instructions. The only problem, as you will see if you create and fill this version of the SAMPLE.EG master, is that the `←` after each initial instruction is inserted into the document – giving you a number of blank lines at the start of each document. The next section describes how you can avoid this sort of problem and still keep your master document looking neat.

3.4 Improving the master document

Once you have typed all the LocoMail instructions into your document, there are still ways in which you can make the master document easier to understand. In the above examples, you saw how a master can be made clearer by using an initial information-gathering block of instructions. The other actions you can take involve:

- spacing out and lining up the separate parts of instructions
- removing (-Mail) (+Mail) combinations between consecutive instructions

Spacing out instructions

You can space out and align the parts of LocoMail instructions in the same way as you space out and align text – by using spaces and tabs. However, you can't just include these anywhere in your instructions – you can only put them around the separate parts of the instruction (the `?`, `!`, *item-name*, `;`, = etc.). Spaces or tabs within the prompt part of the instruction will, of course, be treated as part of the text you want to display.

For example:

```
(+mail)date==?Today's date?(-mail)←
```

```
(+Mail)date=:; today date: ( : : : : : )
(+Mail) reference=:; Reference number? (-Mail)
```

can be aligned using tabs and spaces:

Can I engineer something like this?

(+Mail) date → = ? : Today's date? (-Mail) ←

```
(+Mail) date → - ; , today's date: ( mail /
(+Mail) reference → ? : Reference number? (-Mail)
```

Replacing (-Mail) (+Mail) combinations

If you have two or more consecutive LocoMail instructions, one after another, you probably have:

for property *name*:

The **(-Mail)** (+Mail) combination that separates the commands acts simply as a *command separator*. LocoMail also recognises two other 'command separators' that can be used between different instructions:

- `↵` (a carriage return), giving you :

```
(+Mail) instruction1 ↵
instruction2 ↵
instruction3 ↵
... (-Mail)
```
 - • : (a colon), which gives you:

```
(+Mail) instruction1 : instruction2 : instruction3 : ... (-Mail)
```
- Neither of these can be used in place of `(-Mail)` combinations – with one exception. You can't use a colon as a command separator after an instruction that includes prompt or comment text. This is because LocoMail takes everything after a semi-colon as the text of the prompt (or comment), until it sees the next `↵` or `(-Mail)` code. In this case, you might use a mixture of command separators as follows:

```
(+Mail) instruction1: instruction2; prompt ↵
instruction3: ... (-Mail)
```

All command separators are 'swallowed' as LocoMail processes the instructions, and do not appear in the finished versions. This makes them very useful for tidying up instructions. In particular, it means that you can use returns to space out your instructions, without worrying about them appearing in the final version.

This is particularly important in masters where you have an initial group of instructions (such as in the second example using SAMPLE.IG, above):

```
(+Mail) product=? : product purchased? (-Mail) ←
```

```
(+mail)product=;EProduct purchased: (+mail) letter date=:Date of customer's letter? (-mail) ←
```

```
(+MAIL)TELNET gate=:/data/or/branches/branch: \
(+MAIL)local_branch=?local_branch address?(-Mail)←
```

```
(+Mail) TODAY_BRANCH=1; TODAY_BRANCH=1;
(+Mail) !today=?:Today's date? (-Mail)
```

Because the ←s at the end of each line are between (-Mail) and (+Mail) codes, they will end up in the finished version. However, if you replace each (-Mail) ←(+Mail) combination with a ← command separator you get:

```
(+M=1) product=? : Product purchased? ↵
```

```
((+Mail/product=?;Product purchased:\nletter date=?;Date of customer's letter?<
```

letter_date=;date of customer's receipt

It advances? Today's date? (-Mail) ↓

This gives you just one carriage return in the finished document – the one following the date instruction. However, as LocoMail will always ignore extra command separators, you can also put carriage returns:

- immediately after (+Mail) codes, or
- immediately before (-Mail) codes.

This enables you to get rid of the final $\leftarrow$, and gives you (with tabs to line up the instructions):

(Mail)Product = ; ; Product purchased
 Letter date = ; ; Date of customer's letter
 Local branch = ; ; Local branch
 Local date = ; ; Local date
 (Mail) (Mail)
 (Mail)
 (Mail)
 (Mail)
 (Mail)
 (Mail)
 Dear (Mail) (Mail),
 Thank you for your letter of (Mail) (Mail). I am extremely sorry
 to learn that the (Mail) (Mail) is not working properly. I can
 assure you that our customers have reported very few problems
 with this product.
 May we suggest in the first instance that you take your (Mail) (Mail)
 back to our (Mail) (Mail) branch for servicing, ask to speak to
 the Service Manager. He has been advised of the problems you
 have been experiencing.
 Yours sincerely,
 Joan Smith
 Customer Services Manager

Note that none of these extra lines or tabs will appear in the finished versions.

- When you have tidied up the LocoMail instructions in your master document, you can save it. The next stage is to test the master.

3.5 Testing

After you have prepared and tidied up your master document, you need to test it to check that it works the way you intended.

The best way to see exactly how the master works and whether it uses the details correctly is to use the master in LocoMail's Fill mode, before you try and merge it with a data file. (Of course, you only need to test it in the Merge mode if some of the details are intended to come from a data file!)

Using Fill mode for testing has two advantages over using the Merge mode:

- you don't need to have your data file ready before you test the master document
- it is generally easier to spot where things go wrong in Fill mode (eg. you will see immediately if LocoMail skips over one of the 'blanks' you've marked)

As LocoMail works through a master document in Fill mode, it will prompt for all the details it needs from the keyboard. Whenever it finds an instruction that would take information from the data file in Merge mode, LocoMail automatically treats this type of instruction as if there was a question mark before the *item-name*. If you've included a comment in the instruction, this will be used as the prompt; if you haven't, then LocoMail will use the *item-name* itself plus a ? as the prompt.

For example, the instruction

(+Mail) product ; product purchased (-Mail)

will, when you are using the document in Fill mode, give the prompt

product purchased

If the instruction in the master were just

(+Mail) product (-Mail)

then LocoMail would prompt you with

PRODUCT?

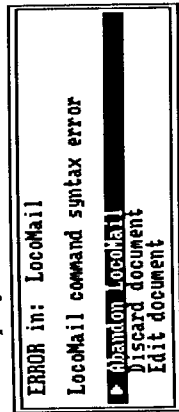
as it filled the document.

To fill the document, display the Disc Manager Screen. Move the File cursor to the master document you've created and press **F**; this selects both the master document and the **F=Fill** option. Check that you have selected the right document on the Selection menu that appears, and set the print options to draft quality and one copy, then press **[ENTER]**.

3.5.1 Errors in the master

The sort of errors you might encounter as you fill the document in Fill mode are:

- LocoMail stops and displays an Alert message:



This means that LocoMail does not recognise the current instruction. The most common reasons are:

- You may have omitted part of the instruction (eg. the semi-colon before the prompt/comment text), or included characters that aren't recognised, or got the parts of the instruction in the wrong order.
- The *item-name* you've used in the instruction doesn't follow the rules given in Section 2.1.1 - letters and digits only, no spaces, starting with a letter, etc.
- The instruction isn't finished with a **(-Mail)** code, or a **←** or a **:** (Note that an instruction can only finish with a **:** or **←** when it is immediately followed by another LocoMail instruction.)
- Instructions are being ignored or inserted into the finished version as text
 - If the instruction is on its own, or the first of two or more instructions, check that there is a **(+Mail)** code preceding the instruction.
 - If the instruction is within a group, check that it is separated from the previous instruction by **(-Mail)** **(+Mail)** or **←** or **:**. If it's separated by a **:**, check that the previous instruction doesn't include a prompt or a comment - because that stops : working as a command separator. You will need to replace the **:** by **←**.
- You need to type details into a 'blank' that you expect to be filled automatically with details typed earlier
 - You may have included a question mark in the second instruction.
 - You may have forgotten to store the information at the first point when it was entered - ie. you forgot to include the *item-name* in this instruction.
 - You have misspelt the *item-name* in one or other of the instructions.
- Details aren't being carried over from one pass to the next
 - You have missed out the exclamation mark from the instruction that asks for the information.

- You can see (-Mail) codes in the finished version
 - Check that the (+Mail) and (-Mail) codes are correctly matched around each instruction.
- Part of the master document disappears from the finished version unexpectedly

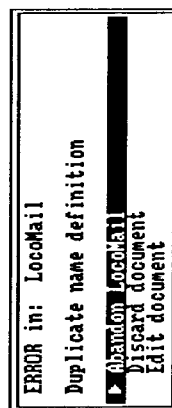
The instruction preceding the missing section of text hasn't been finished properly. Often you can tell this because the highlighting effect of the (+Mail) code extends beyond where you want the instruction to finish.

 - Finish the instruction with a (-Mail) code. Remember, a colon or a ← cannot be used to separate the instructions from text.
- Details are being inserted without spaces before and/or after the details
 - You need to include the spaces that lay out your text correctly in the master document – before the (+Mail) code (or after the (-Mail) code, as appropriate).
- Extra ← characters are appearing in the finished documents

These return characters are outside the LocoMail instructions. If you just need the ←s to improve the layout of the master document:

 - Change (-Mail) ←(+Mail) to ←

When the document works properly in Fill mode, the final stage (assuming you intend to use the master with a data file) is to test the master with the data file you have created, by merging them in the Manual merge mode. This will reveal anywhere where you are not using the details in the data file properly – in particular, where you are trying to store some information under an *item-name* which is used in the record pattern. You cannot change information under *item-names* in the record pattern. The message you will see is:



If you want the information to be taken from the data file, check the instruction in the master – there may be a question mark in the instruction. If you want the information to be taken from the keyboard, change the *item-name* used in the instruction to one that isn't used in the record pattern.

Other errors you encounter when merging a master with a data file are usually due to problems in your data file. These errors are described in Section 2.3.

3.5.2 Dealing with errors

The first step is to identify where the error occurs.

If LocoMail stops and displays an Alert message, you will normally be able to see approximately where the problem is by watching how far LocoMail got when filling the document. Note the point at which LocoMail stopped, then exit from LocoMail – by selecting **Abandon LocoMail** from the Alert message.

If the error you see is the wrong information being inserted, and doesn't make LocoMail stop, simply note where this happened and then select **Abandon LocoMail** from the Exit menu that appears at the end of the pass.

Edit the master and look at the point where LocoMail stopped. Examine the instruction carefully, and check that you have typed the components of the instruction correctly. If necessary, refer to Section 3.3.

When you see an error in an instruction, edit the instructions using the [←DEL] and [DEL→] keys, typing in the correct details just as you would edit any other LocoScript document. Take care to leave the (+Mail) and (-Mail) codes around the instruction (and any : or ← command separators you've used).

When you think you've corrected the mistake (or mistakes), save the new version, fill it again and check.

If the same error occurs at the same point, then either the correction you've made is also wrong or you are correcting the wrong instruction. Re-check the instructions before the point at which LocoMail stops, or any other instructions associated with the one you think is causing the error (eg. ones using the same *item-name*).

Work through your document in this way, clearing up the errors. You will eventually be able to fill the document completely from start to finish. Then you need to test the master with your data file (see Section 2.3).

If you encounter any more complicated problems while filling the document, you may need to refer to Appendix IV (the troubleshooting guide) for more detailed advice.

Full details of how to use the Fill mode are given in Chapter 5.

Using LocoMail in Merge mode

This chapter describes how to produce finished documents by 'merging' information stored in a data file into a LocoMail master document. The data file you use can be either a LocoMail datafile, a LocoFile datafile or some other kind of datafile dressed up a LocoMail datafile (as described in Section 2.2).

A typical use for this is in mailshots – sending 'personalised' versions of a letter to people on an address list, where all the details of name, address, etc. are held in a data file. You might also use LocoMail to slot information prepared by another program into one or more LocoScript documents. For example, you might have a program that stores sales information on disc, and a master document into which you want to insert these details in order to generate invoices.

The LocoMail mode needed for this type of job is the Merge mode. In this mode, the information to be slotted into the 'blanks' in the document is taken from a data file – perhaps with some additional details typed in at the keyboard. You must, of course, have the same *item-names* for this information in the master document as in the data file.

There are two ways of merging a document and a data file – 'Manual' or 'Automatic'. In both cases, LocoMail merges the master with the details from each record in the data file in turn, producing a finished document at the end of each 'pass'. In a Manual merge, you can then choose what you want to do with the document – save it on disc, edit it, discard it or print it. LocoMail then starts with a fresh copy of the master and inserts the details from the next record, again producing a finished document, and so on.

In an Automatic merge, each finished document is automatically printed at the end of the pass, without any intervention on your part. You will end up with a printed version for every record in the data file. As with any other LocoScript document each version is printed on a separate page – all you need to do is keep the printer supplied with paper. None of the finished versions are saved on disc. (Note: Part III of this User Guide describes special LocoMail instructions to let you use details from more than one record in the finished versions, and how to discard versions you don't want even in an Automatic merge.)

Because the end result of using LocoMail is typically to produce printed versions of documents, you need to specify the printing options you see whenever you use LocoScript to print a document – High quality or Draft quality output, and the number of copies to print. These printing options are offered on an initial menu when you select the **M=Merge** option, so that you only need to set them once for all the versions you will produce in the current session.

The merge operation starts and finishes at the Disc Manager Screen. The number of versions that LocoMail prepares while merging is dictated by the number of records in the data file – although you can abandon the merge at any stage if you want to cut short this process.

If you are not sure whether a data file is the one you want to use, or whether it is suitable for merging with your chosen master document, the simplest way of finding out is to try to merge the two. Use a Manual merge so that you can readily abandon LocoMail should the details be incorrect. If the data file and master are incompatible (ie. they don't use the same item-names), you will see an Alert message and the merge will fail. In this event, you may be able to rescue the situation by creating a separate record pattern for your data file which uses the item-names used in the master document.

- If you would like to follow a gentle introduction to using LocoMail in Merge mode, turn back to Sessions 1 and 2 of the tutorial.

4.1 Getting started

- 1 Make sure that you are using a version of the LocoScript word processing software in which you have installed LocoMail.

If your machine is already on, display the Disc Manager Screen. Check that the top lines of the Disc Manager Screen display the LocoMail **F=Fill** and **M=Merge** options, as below:

Disc Manager			
2-Create new document	1-Edit documents	Printer file	Using none
3-Delete documents	4-Print documents	5-Direct printing	F=Fill M=Merge
6-Options	7-File	8-Settings	9-File
10-Options	11-File	12-Settings	13-File
14-Options	15-File	16-Settings	17-File
18-Options	19-File	20-Settings	21-File
22-Options	23-File	24-Settings	25-File
26-Options	27-File	28-Settings	29-File
30-Options	31-File	32-Settings	33-File
34-Options	35-File	36-Settings	37-File
38-Options	39-File	40-Settings	41-File
42-Options	43-File	44-Settings	45-File
46-Options	47-File	48-Settings	49-File
50-Options	51-File	52-Settings	53-File
54-Options	55-File	56-Settings	57-File
58-Options	59-File	60-Settings	61-File
62-Options	63-File	64-Settings	65-File
66-Options	67-File	68-Settings	69-File
70-Options	71-File	72-Settings	73-File
74-Options	75-File	76-Settings	77-File
78-Options	79-File	80-Settings	81-File
82-Options	83-File	84-Settings	85-File
86-Options	87-File	88-Settings	89-File
90-Options	91-File	92-Settings	93-File
94-Options	95-File	96-Settings	97-File
98-Options	99-File	100-Settings	101-File

If these options are not displayed, reset your machine using the the **[SHIFT]**, **[EXTRA]** and **[EXT]** keys and the LocoScript with LocoMail Start-of-day disc (with LocoMail) you produced when you installed LocoMail (Part I). Check that the Disc Manager Screen shows the Fill and Merge options. If the options are still not shown, you must repeat the Installation procedure. Be sure to follow the steps exactly as described.

- 2 Organise your files so that you have access to both the master document and the data file simultaneously from the Disc Manager Screen. If you change the disc(s) in the drive(s), remember to press **[7]** to signal this change to LocoScript.

If both files are on the same disc, simply put this disc in the drive. If they are on separate discs, then you need to arrange things so that you can select both from the Disc Manager Screen:

- if your machine has two drives, put the disc holding the master document in one drive, the disc holding the data file in the other.
- if your machine has one drive, you need to place either the master document or the data file on Drive M (by copying it from Drive A to Drive M), then take this disc out of the drive, put the disc holding the other document in and press **[Z]** – you will then be able to access both documents from the Disc Manager Screen. (Details of how to copy files are given on page 79 of the LocoScript 2 User Guide, or page 100 of the PCW9512 User Guide.)

- 3 Move the File cursor until it is on the master document and press M.

This selects both the master document and the LocoMail **M=Merge** option.

A message will appear telling you to use the File cursor to pick out the file you want to use as the data file.

- 4 Move the File cursor until it is on your data file. Press **[ENTER]**.

You can select either a LocoMail datafile or a LocoFile datafile at this point. If you want to use a LocoMail datafile that has the record pattern stored in a separate file, you must pick out the file containing the record pattern: otherwise, you will get an error message and you will need to start again.

- 5 Check that the details in the menu that appears are all correct. Select the print quality you require, and select the type of merge that you want to use – Manual or Automatic.

Merge documents	
File:	ADDRESS LIST
Group:	TUTORIAL
Drive:	A
Name:	MAILSHOT.IG
Group:	TUTORIAL
Drive:	A
☒ High quality ☐ Draft quality	
Number of copies:	1
☐ Manual ☐ Automatic	

Note: If there is no printer available to LocoScript, you won't be offered the Automatic merge option – only the Manual option.

First, check that the correct files have been selected. If any of the details are wrong – for example, because you accidentally selected the document next to the one you wanted, move the highlighting to the relevant line of the menu and type in the correct details, or press **[CAN]** and try again.

Then, set the print quality and the number of copies you require of each version. The default settings offered will be one copy and whatever print quality is currently selected.

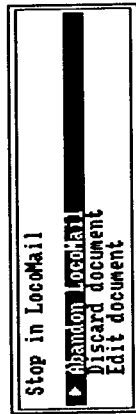
Finally select the type of merge you require. If you just want LocoMail to produce a batch of printed versions, select **Automatic**. Select **Manual** if you want greater choice over what you want to do with each version.

6 When all details on this menu are correct, press **[ENTER]**.

If the data file you picked out (in step 4) just contains a record pattern, you will then see a second message telling you to pick out the data file you want to use. This time, move the File cursor to the file holding the 'real' data records and press **[ENTER]**. Another short menu appears, allowing you to check (and correct) the data file details before pressing **[ENTER]** again.

The master document is then displayed on the screen before being processed by LocoMail. LocoMail works through the document, stopping at each instruction and executing it, then carrying on to the next.

If the document isn't the one you intended using, press **[STOP]** then **[STOP]** again. Then select **Abandon LocoMail** from the menu that appears.



4.2 Filling in the details

Whenever the instruction tells LocoMail to fetch information from the data file or from its memory, the information is automatically slotted in. You don't have to do anything.

When LocoMail comes to an instruction telling it to get the information from the keyboard, the actions are quite different.

The first thing LocoMail does when it needs information from the keyboard is to display a prompt and wait for you to type in the details. The text of the prompt will depend on what was in the instruction LocoMail has stopped at:

- If the LocoMail instruction marking the 'blank' contains a semi-colon followed by text, then this text is used as the prompt.
- If no prompt text is included in the instruction, then LocoMail will offer a prompt consisting of the *item-name* in capital letters. For example, **(+Mail) customer=?** (-Mail) will give the prompt **CUSTOMER?**

- If there is no prompt text and the information isn't to be stored under an *item-name*, then the prompt will just be a question mark. In other words, **(+Mail)?** (-Mail) will give ? and you will need to look at the context of the instruction within the master document to find out what to type

The prompt should make it clear what to type: if not, you should be able to see where the 'blank' fits into the rest of the document displayed on the screen.

Type in the information you want to insert at this point, pressing **[ENTER]** when you have finished typing:

- If the details you want to insert are just a simple line of text, then type the details and press **[ENTER]**.

For example, to enter the date '12th June 1988', type

12th June 1988 **[ENTER]**

- If you want to insert details which spread over a number of lines, for example an address, then type the details, pressing **[RETURN]** to move on to a new line. Only press **[ENTER]** at the end of the address.

For example, to enter the address '1 The Mews, The Street, Oldville', type

1 The Mews **[RETURN]**
The Street **[RETURN]**
Oldville **[ENTER]**

If you make a mistake as you type, use the Cursor keys to position the cursor, the **[←DEL]** and **[DEL→]** keys to rub out the error, and make any corrections. Only when the details are correct should you press **[ENTER]**. However, you don't have to have the cursor at the end of the details before you press **[ENTER]**.

Note: You can only change the text you are actually entering – if you notice something you want to change in the rest of the document, and you are using Manual merge mode, edit the document when LocoMail has filled in all the blanks in this version (see the description of the Exit menu at the end of Part II, page 113).

The prompt remains on screen while you type. However, when you press **[ENTER]** to signal that you have finished typing the details that are to be inserted, the prompt is removed from the screen.

In general LocoMail simply slots whatever you typed into the document, then goes on to the next 'blank'. If, however, you have not typed a 'true' number when LocoMail was expecting one (ie. the current instruction includes the ?#... command described in Chapter 3), you will hear a bleep and both the prompt and what you typed will remain on screen with the cursor placed just after the character it has detected as wrong. Use the Cursor keys and the **[←DEL]** and **[DEL→]** keys to make the piece of text a 'true' number (containing just digits, with a + or - sign and a decimal point if required). When you have the number you require – regardless of the current position of the cursor – press **[ENTER]**.

As LocoMail moves on through the document, LocoScript will automatically 'relay' the paragraphs so that the paragraphs in the finished document are all perfectly laid out.

Note: When you get used to what details are necessary to fill in a particular master document, you could find yourself 'typing ahead' – that is, typing in information before LocoMail has asked you for it. This is fine, provided that you type all the information correctly. But you must be sure that you don't press **[ENTER]** more times than the number of 'blanks' LocoMail is expecting you to fill in the document. In a Manual merge, pressing **[ENTER]** too many times will automatically select the Edit result option at the end of the pass, giving you the chance to sort out the mistake – if you are performing an Automatic merge, there's no opportunity to correct your mistakes and too many **[ENTER]**s could cause real problems!

The sequence of prompts you see won't necessarily be the same on each pass through the document. Don't let this worry you. As you saw in Chapter 3, certain LocoMail instructions can remember details from one pass to the next, and so they don't need to prompt for the information on the second and subsequent passes. There are also more complex LocoMail instructions (introduced in Part III of the User Guide) which can cause sections of the master document to be included only under certain conditions – so any instructions embedded in the omitted section won't appear. The only thing to check is whether the filled document produced at the end of the pass contains all the correct information.

• *If you want to abandon the document you are working on at any stage, press **[STOP]** then **[STOP]** again. Then select the appropriate action from the menu that appears. Alternatively, if you are doing a Manual merge, you can select Abandon LocoMail from the Exit menu (see the description at the end of Part II).*

Adding Style

As a rule, all the word-processing codes needed to control the styling and positioning of details inserted into the 'blanks' will be built into the master document. However, occasionally you may want to put extra styling into particular versions of the document. You might, for example, want to underline part of the information in particular cases.

There are two ways that you can add codes into a version:

- either** include the word processing codes in the details you type (this is the only method in an Automatic merge)
- or** continue through the document until the end, then choose the **Edit result** option from the Exit menu (see below) and insert the necessary word-processing codes just as you would when editing any other LocoScript document (this is only possible in a Manual merge).

LocoMail allows you to include the codes in the details you type because it always waits until you press **[ENTER]**, then it takes everything that you've typed (including codes, characters, carriage returns, etc.) and inserts it into the document just as if you had included the same details in the original master at that point.

If you include any 'styling-on' codes in what you type (eg. **(+Bold)**), you must be sure to include the corresponding 'styling-off' codes before pressing **[ENTER]**, otherwise the styling effect may continue through the rest of the document. For example, if you put a **(+UL)** code in what you type, you must also put in a **(-UL)** code before pressing **[ENTER]** or the whole of the rest of the document may be underlined.

If you are inserting codes, you will probably find it easiest to work with all word-processing codes displayed on the screen. If these are not already being displayed, press **[A]** and select the Codes option from the Options menu.

Insert the codes in the details you type in the normal way: when you reach the point at which the code needs to be inserted either press the **[B]** or **[U]** key and type the short form of the code (eg. **[B]** or **[U]** to turn on underlining), or use the pull-down menus (by pressing the **[B]**, **[U]**, ... keys).

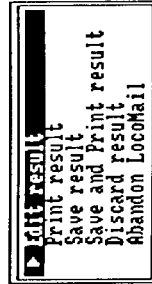
If you use the pull-down menus to enter your styling codes, you will have to press **[ENTER]** to insert the code selected on the menu into your typing. LocoMail doesn't confuse this press of the **[ENTER]** key with the signal that you have finished typing in details to the current 'blank'. A separate press of the **[ENTER]** key is required for this.

4.3 At the end of the pass

Once all the 'blanks' have been filled, LocoMail has finished this particular 'pass' through the document.

In an Automatic merge: the document is printed (following the usual check of the printer and paper types, and provided the printer is on-line and not waiting for paper). LocoMail then starts creating the next version. You don't have to do anything except keep the printer supplied with paper. LocoMail will continue producing and printing versions until the data records are exhausted.

In a Manual merge: LocoMail puts up an Exit menu offering a choice of actions that you can take with the document in its current state.



LocoMail will wait until you select an option from this Exit menu. (The effect of each of the options is explained at the end of Part II, page 113.) Once you have selected an option, and LocoMail has taken the necessary action, it will start to prepare the next version, using the next data record.

When LocoMail has worked through the whole of the data file, creating and filling versions of the master for every record, it returns you to the Disc Manager Screen.

If printing is still being carried out as you return to the Disc Manager Screen, you will see an extra hidden file on Drive M. This is the document currently being printed; it can be ignored.

Note: If the printer is still printing a previous version when the current version is ready to be printed, LocoMail pauses and puts the following Alert message on the screen:

LocoMail will print the document
when the printer becomes idle
▶ printer now idle: retry
Cancel operation

Chapter 5

Using LocoMail in Fill mode

This chapter describes how to take an existing LocoMail master document and produce a finished document by typing in all the information that needs to go in the blanks – rather like you might fill in a form. You might use this either to fill up forms, prepare invoices, etc., or to prepare versions of a standard letter, a few at a time.

The LocoMail mode this uses is Fill mode. In this mode, all the information to be slotted into the 'blanks' in the master is typed in at the keyboard. You don't use a data file at all. The master that you use can either be one that you have prepared specifically for use in Fill mode, or one that you usually use in Merge mode, because in Fill mode LocoMail treats all the instructions as if they were of the type requiring details from the keyboard. In fact, an important use of the Fill mode is in testing master documents that you will ultimately merge with a data file.

Note: If you do want LocoMail to take any of the information in the finished version from a data file stored on disc, you must use LocoMail in Merge mode – as described in Chapter 4. You will still be able to add the other details in each version from the keyboard.

When you use the Fill mode to fill in a master document, LocoMail works through a copy of the master, prompting for all the details. At each blank, it will insert everything that you type before pressing **[ENTER]** (including word-processing codes, text, carriage returns, etc.) into the document it prepares. Once all the blanks are filled in this way, you can choose what you want to do with the completed document from a menu of options – save it to disc, edit it, discard it or print it. LocoMail then asks you whether you want to fill in another version of the document or finish and return to the Disc Manager Screen.

You can produce as many (or as few) versions of the master document as you want, when working in Fill mode. Because LocoMail isn't using records in a data file to produce versions, there is no pre-determined number of versions to be prepared.

The typical end result of using LocoMail is to produce printed versions of the master document, so you need to set the print options you see whenever you use LocoScript to print a document – High or Draft quality, and the number of copies. These options are offered when you initially select the Fill mode, so that you only need to set them once for all the versions you subsequently print. As with any other LocoScript document, each printed version starts on a separate page.

- If you would like to follow a gentle introduction to using LocoMail in the Fill mode, turn back to Session 3 of the tutorial.

5.1 Getting started

- 1 Make sure that you are using a version of the LocoScript word processing software in which you have installed LocoMail.**

If your machine is already on, display the Disc Manager Screen. Check that the top lines of the Disc Manager Screen display the LocoMail **F=Fill** and **M=Merge** options, as below:

[illegible]

If these options are not displayed, reset your machine using the the **[SHIFT]**, **[EXTRA]** and **[EXIT]** keys and the 'LocoScript with LocoMail Start-of-day disc (with LocoMail)' you produced when you installed LocoMail (Part I). Check that the Disc Manager Screen shows the Fill and Merge options. If the options still don't appear, you will have to repeat the Installation procedure. Be sure to carry out each of the steps exactly as described.

- 2 Place the disc holding the LocoMail master document that you want to fill in the drive and press **77**.
- 3 Move the File cursor to the master document you want to fill, and press **F**.

This selects both the master document and the LocoMail **F=Fill** option. Check the Selection menu that appears to make sure you have chosen the right document, and set the print quality and the number of copies of the filled document you require. When all details are correct, press **ENTER**.

The master document will then appear on the screen and LocoMail will begin to process the document. If you see that the document isn't the one you intended, press **STOP** and then **STOP** again – you can then select **Abandon LocoMail** from the menu that appears, and start again from the Disc Manager Screen.

5.2 Filling in the details

Filling in the 'blanks' in the master document when you are using LocoMail in Fill mode is simply a matter of typing in an appropriate piece of text when LocoMail prompts you to do so – very much as you might have done when using LocoMail in Merge mode. The prompt should always make it quite clear what you need to type; even if it doesn't, you should be able to see where the 'blank' fits into the rest of the document displayed on the screen.

The prompt on the screen will always be highlighted, and it will be followed by the cursor. The prompt is used to signal to you that LocoMail is waiting for you to type something. The prompt which appears depends on the type of LocoMail instruction currently being processed.

- If the instruction contains a semi-colon followed by text, then this text is used as the prompt. For example, **(+Mail) ? customer ; Enter customer's name (-Mail)** will give the prompt **Enter**

An instruction like **(+Mail) address ; Insert address (-Mail)** that is meant to take information from a data file but includes a comment, will give the prompt **Insert address**.

- If there is no prompt or comment text in the instruction but an *item-name* is included, then LocoMail will offer a prompt consisting of the *item-name* in capital letters. For example, (+Mail) **CUSTOMER=?** (-Mail) will give the prompt **CUSTOMER?** and (+Mail) **ADDRESS** (-Mail) will give **ADDRESS?**
- If there is no prompt text and the information isn't to be stored, then the prompt will just be a question mark. In other words, (+Mail) **?**

Type in your response to each prompt that appears as LocoMail works through the master, pressing **[ENTER]** only when you have finished typing all the details to be inserted in the blank. If you make a mistake as you type, use the Cursor keys to position the cursor, the **[←DEL]** and **[DEL→]** keys to rub out the error, and make any corrections. When your response is correct, press **[ENTER]**. (You don't need to move the cursor to the end of the details before pressing **[ENTER]**.)

If you want to fill in a 'blank' with information that spreads over a number of lines (eg. an address), use the **RETURN** at the end of each line of the details and only press **ENTER** after the last character of the details.

If you want to add word-processing codes to the details you type, remember that LocoMail will slot everything you type before pressing **[ENTER]** into the master at this point. So if you include any codes which turn a styling effect 'on' (eg. **(+Bold)**, or **(+Italic)**) then you must remember to include a code to turn

the styling 'off' before you press **[ENTER]**. If you don't, all the styling effect may continue through the rest of the document.

Note: If you hear a 'bleep' when you press **[ENTER]**, and the prompt and your response remains on screen, then you have typed a text character when LocoMail was only expecting a number. Numbers may consist only of the digits 0 - 9, a + or - sign and a decimal point. The cursor will be positioned beside the first invalid character in your response. Use the **[←DEL]** and **[DEL→]** keys to delete the character(s), type in the correct value and press **[ENTER]** again.

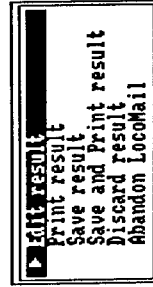
Chapter 4 explained in detail how to insert multi-line information, adding style to what you type, typing 'true' numbers when necessary, etc. If you are unsure how to enter any of these types of information, turn back to Section 4.2.

When what you typed has been inserted, LocoMail goes on to the next 'blank', tidying up the document as it goes so that the paragraphs in the document it creates are all perfectly laid out.

Note: When you get used to what details are necessary to fill in the master document, you could find yourself 'typing ahead' - that is, typing in information before LocoMail has asked for it. This is fine, providing that you type all the details correctly. If you do make a mistake, you may be able to correct this by editing the result at the end of the pass, or you may have to discard this version and start again. In particular, if you press **[ENTER]** more times than the number of 'blanks' LocoMail is expecting you to fill, you will automatically select the **Edit result** option from the Exit menu - so you can make any corrections that are necessary.

5.3 At the end of the pass

Once all the 'blanks' have been filled, LocoMail has finished this particular 'pass' through the document. It then automatically puts up a menu offering a choice of options that you can take with the document in its current state - the Exit menu:



Note: If you don't have a printer connected to your machine, LocoMail won't offer you the option of printing the document.

The effect of each of these options is given in a special section at the end of Part II of this User Guide (page 113).

When an option has been chosen from this menu and the appropriate action has been carried out, LocoMail will typically present another menu - the Fill again/Finish menu.



Fill again - displays a fresh, unfilled copy of the master document and starts the Fill process over again, preparing another version. You don't have to wait for LocoMail to finish printing the current document before you can begin filling in the next version

Finish - leaves LocoMail and returns to the Disc Manager Screen. You don't have to wait for LocoMail to finish printing the last document before you can carry out some other task from the Disc Manager Screen.

This menu does not appear after selecting either Edit result or Abandon LocoMail from the Exit menu.

5.4 Using Fill mode to test master documents

An important use of LocoMail's **F=Fill** mode is to test master documents that you create - in particular, masters that you will eventually use in Merge mode with a data file. Working in Fill mode requires you to type in the information that is used to fill the document - which is particularly useful when:

- you haven't prepared the data file
- the master uses some of LocoMail's more advanced features and you need to check that LocoMail is doing the right things with your data by making some of the items have special values

Note: The second of these points is very important when you use the arithmetic features of LocoMail, and you want to test that the calculations in the master give you the correct results (eg. before working with real customer accounts!).

Whenever you create a new master document you should fill it (as described in Sections 5.1 to 5.3), trying out all possible options (and typing in a range of values, if appropriate) and seeing whether you get the results you expect. You can easily spot if any instructions marking 'blanks' in the master are being skipped, or if there are any problems or mistakes in your instructions.

The mechanics of using Fill mode to test a master document, the sort of errors that are commonly encountered and solutions to these problems are described in Section 3.4 and in Appendix IV, the Troubleshooting section.

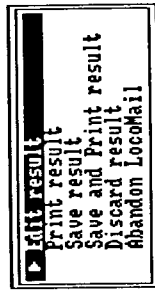
Testing advanced LocoMail instructions

Most LocoMail instructions that are intended to be used in Merge mode can be tested in Fill mode. LocoMail simply takes an appropriate equivalent action.

However, when you test more complex masters than these, you can get errors that are a result of the fact that you are using Fill mode rather than Merge mode. In particular, if your master uses Loop instructions (as described in Part III, Chapter 9) you may see a **Name does not exist** Alert message at the point where the Loop instruction appears in the master.

Try merging the document with a suitable data file (or one set up specially for the purpose) to see whether the problem remains when using the Merge option.

The Exit menu



The Exit menu appears at the end of each pass in LocoMail's Manual merge and Fill modes

Note: If there is no printer available to LocoScript, then this menu won't offer you options which involve printing.

Edit result – allows you to modify the finished document.

When you select this option, you can edit the document on screen just as you would edit any LocoScript document – deleting, adding text or codes, cutting/pasting, etc.

When you have finished making the changes you require, press **[EXIT]**. The Exit menu will appear again, so that you can print, save, etc. the document complete with the changes you've made.

Print result – prints the document. Once printed, this version of the document is thrown away; it is not saved on disc.

Before printing starts, the usual checks are made of the printer and the paper type. Printing then starts (provided your printer is on-line and not waiting for paper) and simultaneously LocoMail starts to prepare the next version of the document.

If the printer does not start printing, check that it has paper in it; and if necessary press the **[FRT]** key and use the **[A]** menu to Resume printing. (If the printer is expecting to use single-sheet paper but you are actually using continuous stationery, you may also have to clear the Waiting for paper state. If you are using LocoScript v2.12 or later, you can do this simply by pressing **[E]**: with earlier versions of LocoScript 2 you instead press **[A]** followed by **[ENTER]** and then press **[EXIT]** to return from adjusting the printer. Set up your printer correctly for the paper you want to use, as described in Session 19 of your LocoScript 2 User Guide, or Session 20 of the PCW9512 User Guide.)

Save result – saves the finished document on your disc. The document isn't printed.

The saved document takes up disc space – a valuable commodity – so we suggest you think whether you really need to have this filled version of the master document on disc as well as the master document itself before selecting the **Save result** option.

When you select this option, the Disc Manager Screen is displayed, together with a message at the top of the screen telling you to use the cursor to pick out the drive and the group in which you want to store the document. Move the cursor to your chosen group (using the **[SHIFT]** key with the Cursor keys, if the group you want to use is currently empty) and press **[ENTER]**.

Introduction

A menu then appears, giving a suggested name for the document together with details of the drive and group you have selected. Edit these details to give the document the name you want (and if necessary to change your selection of drive and group) as you would, for example, when you create a file, then press **ENTER**. The document is then saved.

Save and Print result – combines the actions described above for **Save result** and **Print result**.

The document is first saved as described above for **Save result**. Printing starts after the document has been saved (provided the printer is on-line and not waiting for paper).

Discard result – throws away the document you have just prepared.

If you are using Merge mode, LocoMail starts to prepare the next version, using the next data record from the data file, so if you just want to prepare an amended version of the document on screen with the same details inserted from the data file, don't choose the **Discard result** option. Instead, choose the **Edit result** option and make the changes you require.

Abandon LocoMail – immediately leaves LocoMail and displays the Disc Manager Screen. There are no further prompts and no more versions will be printed.

Note: If the printer is still printing when you return to the Disc Manager Screen you may observe an extra 'Hidden' file on Drive M. This is the document which is being printed; it can be ignored.

The standard LocoMail task is simply to take data from either a data file, the keyboard or LocoMail's memory and slot this information into a master document. This is the action controlled by the fundamental LocoMail instructions described in Part II of this User Guide, and as you have seen, it enables you to get a lot of very useful work done.

However, LocoMail also includes many more advanced features, that allow you to use it to much greater effect. These are:

- special instructions, such as 'move to the next record immediately', 'keep the same record for the next version', 'abandon the current document', etc.
- the ability to change items of information, for example, adding a £ at the front of a number to give you a price
- the ability to compare items of information and then do different things depending on the outcome of the comparison
- the ability to carry out arithmetic on 'numeric' information
- the ability to repeat sequences of instructions until some condition is met

These features give you much greater control over how the information in the data file is used. They also allow you to make your master documents much more versatile. For example, performing arithmetic means that you can use LocoMail to calculate totals as you go, etc. – you don't need to work out figures in advance. This part of the User Guide shows you how to apply these features.

Before you read the chapters in this part, you must read the section titled **More About LocoMail**, overleaf. This short section gives more details about fundamental concepts behind LocoMail – details which will help you to tackle the more advanced tasks that LocoMail is capable of.

- Chapter 6 provides a general introduction to using comparisons to achieve different results, and shows how to control your data files – advancing to the next record when you want, holding on to the current record for the next version, and where appropriate throwing away unwanted versions altogether (eg. only producing versions for people living in Wales).
- Chapter 7 shows how to produce more 'intelligent' masters, which will include or omit different sections of text from one version to the next.
- Chapter 8 shows how to get LocoMail to do arithmetic on the information that you supply. This is particularly useful for business applications, because you can automatically produce totals, discounts, VAT calculations, etc. You will also see how to ensure that paragraphs in the document are numbered correctly even when you miss some out.

• Chapter 9 describes how you can group a number of LocoMail instructions and then use this group to carry out any repetitive tasks in your masters (such as producing lists, etc).

• Chapter 10 describes how you can embed LocoMail instructions within other LocoMail instructions – 'nesting' instructions – and how LocoMail works through these nested instructions.

• Chapter 11 describes the principles of design which we suggest you follow, particularly in your more complex masters. As you make more use of these advanced features, you will need to take more care over the structure and appearance of your master documents so that they will be understandable.

(If the datafile you will be using is a LocoFile datafile, then there are further options for picking out just the information you want and in the order you require. These are described in the LocoFile User Guide.)

Throughout these chapters, LocoMail is used almost as a programming language. If you have no experience of programming, then you might find some parts a little difficult to understand at first. But all should become clear if you study the various examples. You do, however, need to be familiar with the essentials of LocoMail (how it works, how it uses the data files and master documents, etc). If you get stuck, you might find it helpful to ask someone who has experience of programming eg. in BASIC.

More about LocoMail

Before you use the advanced instructions given in this part of the User Guide, there are a couple of features of the LocoMail instructions we've used already that you need to understand in a little more depth.

Firstly, the **(+Mail)** and **(-Mail)** codes themselves. So far, we've treated these as just marking the beginning and end of LocoMail instructions, and also having the convenient effect of highlighting the instruction between the codes, making it very easy to pick out the instructions on the screen.

When handling more advanced instructions, it is better to think of each **(+Mail)** code as making LocoMail 'active', and each **(-Mail)** as making LocoMail 'inactive'. The command separators (ie. **(-Mail)...****(+Mail)**, **↵** or **:**) have no effect on this status.

• **When LocoMail is 'active'**, it interprets the characters it meets as instructions to be executed, and it carries on executing these characters until it meets a **(-Mail)** code. The characters it meets, and the arrangement of these characters, tells it what to do – and as you have seen, this arrangement (the 'syntax' of an instruction) is strictly defined.

• **When LocoMail is 'inactive'**, the characters, codes, etc. LocoMail meets are treated as text and are typically inserted straight into the finished document.

In the early parts of the User Guide, the places in the document where LocoMail is active were clearly visible, thanks to this highlighting effect of the codes. When you use the more advanced features described in this part of the User Guide, you

need to think a bit more carefully about what LocoMail is doing at any point, particularly where you want LocoMail:

- to process whole groups of instructions on some occasions and not on others
- to interpret groups of instructions repetitively
- to carry out instructions within other instructions.

The fundamental rules remain the same:

- LocoMail needs to be active when handling instructions, and inactive when handling text.
- Individual instructions must start with a **(+Mail)**, finish with a **(-Mail)**, and obey the correct syntax.
- Sequences of instructions must be separated by command separators.

But there are occasions where the simple highlighting effect of the **(+Mail)** and **(-Mail)** codes won't make the instructions clear – so what you must follow is the active/inactive status of LocoMail and the order in which the instructions are carried out. We will see more about this in Chapters 9 and 10.

Your task as you set up and use the advanced instructions is essentially the same as it was when working through simple master documents – to make sure that each instruction is typed correctly and that LocoMail always meets intended text when it is inactive, and instructions when it is active.

The other feature we need to look at more closely are double quote marks ("...") in LocoMail instructions. These were briefly mentioned in Section 3.3.3, when describing how to store a piece of text under an *item-name*.

In fact, the double quote marks signal to LocoMail that you want to treat everything enclosed as a single unit, and not to try and act on it. What appears between the double quotes can be any characters or codes you want. Such a sequence is known as a *string* (a term which will be familiar if you have done any programming). For example:

(+Mail) phrase="You owe a (+UL) lot (-UL) of money" (-Mail)

When LocoMail processes this instruction, it knows that it should take everything from the first double quote mark to the next double quote mark, and store it under the *item-name phrase*, without acting on it. (The quote marks themselves aren't stored.) You can even include Mail codes and instructions within the string, without fear that LocoMail will try to execute the instructions at that point. However, the one thing you cannot include in a string like this is other double quote marks – because the next double quote mark that LocoMail meets after the first one tells it this is the end of the string. How you can get round this problem is described in Chapter 10.

Chapter 6

Making better use of your data files

Part II of this User Guide showed the simplest way LocoMail can merge the contents of a data file with a master document to produce different versions of the master. Each of these versions contained items of information from one record in the data file, although these items may have been included many times in the same document. After completing each pass, LocoMail moved on to the next data record. So the end result was a completed version for every data record in the data file.

Often, however, using one record for each version isn't what you require. You might not need to produce a version of the master for each and every person in your address list, for example, or you might want to include details from more than one data record in the finished version, or you might even want to keep the current record for filling in another version (eg. if you make a mistake typing in some details at the first attempt!).

It is possible to do some of these things 'the hard way'. For example, you could print out all the versions of a document and then throw away the ones you don't need. However, this would be wasteful and time consuming. A much better solution is to include special LocoMail instructions in the master document that will automatically do these things for you.

In this chapter we will describe the instructions that make LocoMail:

- produce finished versions only for specific records and abandon the other versions (ie. produce a 'selective' mailshot)
- take information from two or more consecutive data records on the same pass, or keep the same data record for the next pass

6.1 Selecting specific records from the data file

Often, your data file will contain more sets of details than you actually want to use. For example, your address list may contain a number of names and addresses of people that you don't want to include in this particular mailshot. So you need to pick out just the records you want.

The way to do this is to tell LocoMail a condition (or set of conditions) that the ones you want (or, alternatively, the ones you don't want) all meet. If the data in a particular record doesn't meet these conditions, then you can tell LocoMail not to waste time producing this version of the document. Of course, the data in the data file must be such that you can tell LocoMail what the conditions are.

For example, you might have a standard business letter to be sent out to your customers. But suppose your company address list contains names and addresses of people all over the world, and so includes people who speak different languages. What you would then want to do is to send an English mailshot letter only to those people that speak English, a German mailshot letter to the people that speak German, etc. – making your mailshot 'selective'. As long as your address list also includes details of what language each customer speaks, you could do this quite easily with LocoMail.

So that LocoMail can decide whether to produce a version of the master as it looks at each record in the data file, you will need to use a special type of LocoMail instruction in the master document. This special instruction is called a 'conditional' instruction. A conditional instruction tells LocoMail to test whether a condition is true, then carry out different actions depending on the result of the test. Conditional instructions are very important in the more advanced uses of LocoMail, as they effectively allow LocoMail to make choices about what to do next. We shall now describe these instructions in a little more detail, before returning in Section 6.1.3 to the example of producing the 'selective' mailshot.

6.1.1 Conditional instructions

The basic type of conditional instruction is:

```
(+Mail) # condition : < action > (-Mail)
```

What this type of instruction says to LocoMail is 'if this *condition* is true, then carry out the given *action*, otherwise skip to the end of the instruction'. So the *action* is only carried out if the condition is true.

The *condition* is some form of statement that LocoMail can test the truth of (for example, whether one piece of information is the same as another).

The *action* is whatever you need for your purposes – it could be a LocoMail instruction (or series of instructions), or it could be a piece of text to include, or it could be a combination of text and instructions. What you should note is that the *action* is enclosed in a pair of angle brackets < and >. So LocoMail only acts on what is within these angle brackets if the first part of the instruction is true.

You may find this form of instruction easier to remember if you think of # as meaning IF and the < as meaning THEN.

In this chapter, we will concentrate on *actions* that are LocoMail instructions – we look at the various options involving text in the next chapter.

When your *action* is a LocoMail instruction (or series of instructions) then as usual the instructions must be separated with command separators. You must also separate your *action* instruction(s) from the elements of the conditional instruction (the < and >) with suitable command separators.

The command separators you can use are (-Mail) (+Mail), < or : (as given in Section 3.4). For example, if the instructions are short and don't include prompt or comment text, you might use colons to separate the different instructions:

```
(+Mail) #condition:<:instructn1:instructn2:...:last-instructn:>(-Mail)
```

Alternatively, you might use <# (eg. if the instructions include prompt or comment text):

```
(+Mail) #condition :<<
instructn1:instructn2; comment <
...<
last-instructn <
> (-Mail)
```

For instance, you could include an action that sets an *item-name* to a certain value only if a particular *condition* is true, using:

```
(+Mail) #condition:<:number=6:>(-Mail)
```

Here, *number* will be set to 6 only if the *condition* is true; otherwise, it will keep its current value (ie. whatever you've set it to earlier in the document).

Your action could be an instruction prompting for information:

```
(+Mail) #condition:<<
?reply:Type the age<
>(-Mail)
```

Here, you will only be prompted to **Type the age** if the *condition* is true.

Examples of the sort of conditions you can test are given in Section 6.1.2.

6.1.2 Conditions that can be tested

Much of the information that you use conditional instructions to test is text information (usually stored under an *item-name*). What you can test for here is whether two items of information are the same or different.

If the information you are testing is numeric (ie. numbers), you can do rather more – testing whether a value is smaller or larger than another, or whether it is within a given range, etc. Tests involving comparisons like 'greater than', 'smaller than', etc., are meaningless with text.

Note: LocoMail has strict definition of a number. A number can only consist of the digits 0...9, a + or - sign and a decimal point. No other characters are acceptable. (A full description of numeric information is given in Section 8.1.)

Comparisons involving text information

When comparing text information, the only tests you can do are:

- whether two items of information are identical, ie. *item1=item2*
- whether two items of information are different, ie. *item1#item2*
or *item1<>item2*

(Note: The # character is typed by pressing the **EXTRA** and # keys. <> (a 'less than' sign followed by a 'greater than' sign) represents 'not equal'.)

The items of information in these comparisons are either *item-names*, storing the information, or the text itself included directly as a string (ie. between double quote marks). For example:

```
town = "Bath"      - tests whether town is currently Bath
town <> "Bath"     - tests whether town isn't Bath
town = Bath       - tests whether town and Bath hold the same text
```

Note: It is important to remember that the double quote marks here are only used to mark the start and end of the string – they are not included in the comparison of the pieces of text. If the text isn't included as a string, LocoMail assumes it is dealing with an *item-name*. Usually, this will result in an error of some sort: for example, if you have any spaces, etc. in the text you will get an error because *item-names* cannot contain spaces.

IMPORTANT: When LocoMail is comparing pieces of text, it converts everything to upper case before making the comparison – so **Bath** will be considered equivalent to **BATH**, or **bath**, etc.

Comparisons with numeric values

The *conditions* can also test the value of numbers you have stored, and when you are comparing numbers, you can do a greater range of tests:

- whether one value is less than another, ie. *item1* < *item2*
- whether one value is less than or equal to another, ie. *item1* ≤ *item2*
or *item1* <= *item2*
- whether one value is equal to another, ie. *item1* = *item2*
- whether one value is greater than another, ie. *item1* > *item2*
- whether one value is greater than or equal to another, ie. *item1* ≥ *item2*
or *item1* >= *item2*
- whether one value is not equal to another, ie. *item1* ≠ *item2*
or *item1* <> *item2*

(Note: The ≤ character is typed by pressing the **EXTRA** and = keys
The ≥ character is typed by pressing the **EXTRA** and > keys
The ≠ character is typed by pressing the **EXTRA** and # keys)

The items of information in these comparisons can be either *item-names*, storing the values you are comparing, or the numbers themselves. For example:

```
number = 15      - tests whether what is stored in number is 15
year1 < year2    - tests whether the value stored in year1 is less than
                  the one stored in year2
month > 12       - tests whether month is greater than 12
cheque <= 49.99  - tests whether cheque is less than or equal to 49.99
balance >= 0     - tests whether balance is greater than or equal to 0
number <> -3     - tests whether number is not equal to -3
```

When you are comparing numbers, LocoMail will allow you to combine some of these tests, effectively allowing you to test whether a value is within a range:

```
1946 < birth < 1953      - tests whether birth is between 1946 and 1953
```

In this case, LocoMail actually tests whether **1946** is less than the value stored in **birth**, and tests whether the value in **birth** is less than **1953**. Only if both of these tests are true will the expression be true.

Note: When you are comparing numeric values, LocoMail automatically treats them as if they were strings – so you don't need put double quotes round them in the comparisons. In fact, using quoted values can cause certain comparisons to fail, so you should specifically avoid putting numbers in double quotes.

It is also very important to be sure that the values stored under any *item-names* used in the comparisons are true numbers, and contain no text characters. Similarly, you must be careful not to accidentally include any text characters with any numbers that you use directly in comparisons: for example, the test:

```
total <= £100
```

won't be allowed by LocoMail. The <= (is less than or equal to) test is only valid on 'numeric' information and £100 is not numeric. The same applies to all the numeric tests (apart from 'equal' and 'not equal', which are valid with text as well). Of course, **total** must also be storing a numeric value.

Multiple comparisons

You can make multiple comparisons in one instruction by using **OR** and **AND** in the comparisons.

- **OR** between conditions lets you test whether either of the conditions are satisfied
- **AND** between conditions lets you test whether both conditions are satisfied

For example:

```
town = "Bristol" OR town = "Bath"
```

- tests whether town is Bristol or town is Bath (ie. if it is either, the comparison is true)

```
town = "Bristol" AND age >= 18
```

- tests whether town is Bristol and age is greater than or equal to 18 (ie. it must be both for the comparison to be true)

Notice that both sides of the **AND** or **OR** instruction must be proper comparisons in their own right: you can't, for instance, use:

```
town="Bristol" OR "Bath"
```

When you want to carry out more complex comparisons, you can make the operation of the comparison clearer by using brackets around the **AND** or **OR** comparisons. Then you can use further **AND** or **OR** comparisons between these bracketed comparisons. For example:

```
(age>21 AND member="Full") OR (age>25 AND member="Prov")
```

This test will only be true if the age is greater than 21 *and* the membership is Full, or if the age is greater than 25 *and* the membership is Prov(isional).

When using multiple comparisons like these, be sure that you are actually asking the right questions. It is quite easy when using **AND** and **OR** combinations to select exactly the versions you don't want!

Inverting comparisons

There is also a form of conditional instruction which allows you to carry out an action when a certain condition is false:

```
(+Mail) # NOT (condition) : <action> (-Mail)
```

The *condition* you set up within the () brackets is the one under which you don't want to carry out the *action*. The conditions that you can specify here are exactly the same as those you can specify in *#condition* instructions.

When the *condition* in this instruction is a fairly simple one (eg. *item1=item2*), you can often achieve the same results with a different comparison which tests whether a condition is true. For example:

```
#NOT (item1=item2)
```

has precisely the same effect as:

```
#item1 <> item2
```

In both cases, the action will only be carried out when *item1* is not the same as *item2*. Which method you use depends only on which is the most convenient for the circumstances.

However, when the comparisons are more complex than = or ≠, or when they involve multiple comparisons, a **#NOT (condition)** instruction may give you the neatest way of expressing the condition you want to test.

When you use this **NOT** form of condition instruction, always remember to put the entire *condition* within the () brackets.

Flexibility in the comparisons

You can widen the scope of comparisons by using 'wildcard' characters when you want to accept a range of alternatives. For example, if you want the user to give a Yes/No type answer, you really need to be able to accept both Yes and Y. A comparison like **#reply="Yes"** will only accept Yes.

You could just increase the number of comparisons, by using **OR**:

```
reply="Yes" OR reply="Y"
```

However, it isn't really practical to cover any but the simplest possibilities by this method. The best solution is to include 'wildcard' characters in the comparisons. The two wildcard characters you can use are ? and *:

- ? represents a single character or a space, so if you have a comparison like **reply="Y??"**, then **reply** will be considered equivalent to anything beginning with **Y** and with two (and only two) other characters (eg. **Yes**, **Yed** or even **Y s** but not **Y itself**).
- * can represent any number of characters (including none), so if you had **reply="Y**"** then **reply** would be considered equivalent to anything beginning with the letter **Y** (eg. **Y** or **Yes** or even **Yes possibly!**).

Note that for LocoMail to understand these characters as wildcards, they must be part of a quoted string (ie. within double quote marks). Without the double quote marks, LocoMail would take ? as part of an instruction, and try to fetch input from the keyboard! * is also a LocoMail instruction when it appears outside a quoted string. So whenever you want to use these wildcards in your comparisons, be sure to use them only between double quote marks.

In addition, wildcards can only appear on one side of a comparison – if you have them on both sides, then LocoMail will give you an error.

A particularly useful application of the * wildcard is when you want to check whether an *item-name* contains a particular piece of information – for instance, when you want to check whether the *name* record contains **Smith**. The sort of comparison you use is:

```
#name="*Smith**"
```

This will pick out any record where the name includes Smith, such as A R Smith, J Smith-Wright, F Blacksmith, Smith and Jones, etc.

A comparison like **reply="**"** will always be true, so there is no point in using * on its own in a string. * is therefore used only in combination with other characters within the string.

Note: The wildcard characters themselves may appear in the items you want to compare. If they are included in typed input or in information set up within the master document, they will still act as wildcards in comparisons. However, if they are included in items of information in a data file, they will be regarded simply as text characters (ie. just like a, b, x, y, etc.).

Testing your conditional instructions

It is a good idea to test the conditional instructions you want to use before you insert them in your master document, to make sure that they give you the results you expect. This is particularly important if they make complex, or multiple, comparisons.

The best way to do this is to set up a simple test document. This needs to contain instructions to collect the information you require for the test, followed by the conditional instruction(s) that you want to test. For the *action* part of the instruction, use a simple instruction that you can see the effect of (such as ? ; **Type the age**).

Use LocoMail in its Fill mode to work through the master document.

First, type in information that you expect will satisfy the conditions and give a true comparison. You will be able to see if the comparison is working correctly because the prompt will appear (just press **ENTER**) to continue at this point). On subsequent passes through the document, type in details that you expect to give a false comparison, and ensure that the prompt doesn't appear.

If filling the test master shows a syntax error in your conditional instruction, exit from LocoMail and edit the document. Check that:

- All parts of the *action* are correctly separated using the valid command separators, and are similarly separated from the **<** and **>**
- If there are any sections of text in the action, each is preceded by a **(-Mail)** code and followed by a **(+Mail)** code (to make LocoMail inactive/active at the right points)
- The comparisons you are making are valid (that you're not using **>=** with text, for example)
- Each pair of comparisons (in a multiple comparison) is enclosed within brackets

For more complicated instructions (when you are making multiple comparisons, or nested conditional instructions as described in Chapter 10), you will need to make many passes through this simple test master, to test that it works under all possible conditions. Remember that some errors may only appear under particular conditions, so be sure to test all the possibilities.

Only when you are sure that the instruction works as you intended should you include it in the real master document.

6.1.3 Example: Producing a selective mailbox

Now that you have seen how LocoMail can carry out an action depending on the result of a test, the task of producing a selective mailbox should become clearer.

You will remember that we only want to produce versions of the English mailbox for English speakers, versions of the German mailbox for German speakers, etc. So, having tested and discovered that we don't want to produce a letter, the sort of action we need is one that will discard the current version of the master.

The instruction that tells LocoMail to abandon the current version immediately is *****. So the conditional instruction to abandon the current version if a condition is true is:

```
(+Mail) #condition: < *: > (-Mail)
and the one to abandon the document if a condition is false:
```

```
(+Mail) # NOT (condition) : < *: > (-Mail)
```

The first of these instructions says 'if the condition is true, then abandon the current version' and the second says 'if the condition is false, abandon the current version'. Which of these you use depends on the condition you were testing.

IMPORTANT: Be careful not to confuse the two uses that you have seen for the ***** character. When this character appears within a quoted string (ie. between double quotes), it has the effect of a wildcard character as described in 'Flexibility in the comparisons' in the previous section. When ***** appears as an instruction, it has the effect of abandoning the current document.

In our example, you want to send the English version of a mailbox letter only to people on your address list that speak English. Your address list must indicate what language each customer speaks. Suppose that this information was held in each customer's data record under the *item-name* language. The condition you would want to test, therefore, is whether **language** in the current data record is English – if it isn't, you would abandon the version. So your master would need to test whether **language** is NOT English, ie. you would use:

```
(+Mail) # NOT (language="English") : < *: > (-Mail)
```

which says: 'If **language** is not currently English then abandon this version, otherwise carry on'

Similarly, in the French master you would have the instruction:

```
(+Mail) # NOT (language="French") : < *: > (-Mail)
```

and in the German master:

```
(+Mail) # NOT (language="German") : < *: > (-Mail)
```

This instruction is typically put right at the beginning of the master document, so that unwanted versions are abandoned immediately.

To create your various mailshots, you would merge each master document with the address list in turn. The instruction:

```
(+Mail) # NOT (language="English") : < *: > (-Mail)
```

at the top of the English master would pick out from the list just those customers whose language is recorded as English, the French version would pick out just those customers whose language is recorded as French, and so on. Because each record in the data file specifies just one of these languages, at the end of the day there will be only one letter for each person on the address list.

If your address list doesn't include language details, then you would have to think of some other aspect of the data recorded for each person to help you select English speakers, French speakers, etc. For example, you might be able to pick out French speakers by selecting those with the title "M" or "Mme" or "Mlle" with an multiple comparison like:

```
(+Mail) # NOT (title="M" OR title="Mme"
```

```
OR title="Mlle") : < *: > (-Mail)
```

In this case, you need to check for each suitable matching record using a multiple comparison. If you use a wildcard in the comparison (for example, **#NOT (title="M*")**), that will also match title with **Mr**, **Mrs** and **Ms** and produce versions for records containing these titles as well!

If you would like to see the `(+mail)$+(-mail)` instruction in action, merge this master document with the `ADDRESS.LST` data file. Note that the first two `(+mail)address` instructions insert the address from the first data record in the `ADDRESS.LST` file, but after the `(+mail)$+(-mail)` instruction these details come from the second record. When LocoMail completes the document and you print it (or save it, discard it etc.) LocoMail automatically moves on to the third record to start the next version. It then meets the `$+` instruction and takes details from the fourth record, and so on.

A common application of advancing to the next record is printing labels on 'n-across' labels stationery – in this case, you use `$+` instructions repeatedly to collect the details you need for each label across the page, before printing the labels. This type of application is described in detail in a worked example in Chapter 14.

- It can sometimes be a little difficult working out how many **\$+** instructions you need to use in a master document. A simple rule of thumb is that you need one less than the number of records you want to take details from (since LocoMail automatically advances to the next data record at the end of the pass).

6.2.2.2 Keeping the same record for the next pass

The LocoMail instruction that keeps the current data record for use again in the next pass is:

(+Mail) \$-(-Mail)

Essentially, this instruction tells LocoMail to keep the current values of each *item-name* for the next pass, rather than reassigning all the values to those in the next record.

Although the \$- instruction is similar in appearance to the \$+ instruction, there is an important difference in their operation. The \$+ instruction causes LocoMail to advance immediately to the next data record: the \$- instruction tells LocoMail not to advance to the next record at the end of the pass - it does not tell LocoMail to take details from the previous data record.

Suppose, for example, that you have a master document that takes some information from a data file and some quite complex information from the keyboard. If LocoMail always advances to the next data record at the end of each pass, the only chance you have of correcting any mistakes entered at the keyboard is by selecting **Edit result** from the Exit menu at the end of each pass in a Manual merge.

By using the `$-` instruction with a conditional instruction, you could discard any versions that you are unhappy with and use the same record again on the next pass. To do this, you need `$-` as the *action* in the conditional instruction – for example, including a pair of LocoMail instructions like these at the end of your document:

```
(+Mail) success = ?;version successful? Yes or No (-Mail)
(+Mail) #NOT (success = "Y*") :<:$-:*:> (-Mail)
```

The first instruction tells LocoMail to prompt you with the message **Version successful? Yes or No** and to store what you type under the *item-name* **success**. The second instruction (the 'conditional' instruction) then tells LocoMail "if the reply stored in **success** doesn't start with a **Y** then **\$-** (keep the current data record for the next pass) and ***** (abandon the current version of this document)". So if the reply held in **success** does start with a **Y**, LocoMail will ignore the **\$-**: ***** action and as usual will move on to the next data record at the end of the pass.

(Note: You can type your response to the prompt using any combination of upper and lower case – as LocoMail converts all text to upper case before making the comparison, the test will be true whether you type **y, **yes** or **YES**, etc.)**

This pair of instructions is used in **SUCCESS**EG – one of the sample documents on the LocoMail disc.

```
Q:\WORKING\SUGGEST-IG Editing.txt. Printer file Using A4 N.  
Run P12 LSI CR0 IP6 Page 1 Line 17/54  
F1=Actions F2=Layout F3=Style F4=Size F5=Page F6=Options EXIT  
  
Order number: (Alt)P : Order number:(Alt)e
```

रि: ३७३

(Mail) name ; supplier (Mail)

(Email) address : supplier's address (Email)

7-2867

@MailTo: number@Mail - @MailTo: product - @MailTo: post @MailTo:

For delivery to:

Customer: customer's name

+Mail ! ? : customer; customer's address

On: 4/11/2017 11:00:00 AM

(Mail) success=: Version success=: upc pc no

```
# NOT (success = 55ad0ns) { * - $ ! : } { * - $ ! : }
```

Version scrapped and same record used for next pass, only if success doesn't hold Yes or Y

To see this pair of instructions at work, merge this document with the ADDRESS.LST data file and type either Yes or No each time you are asked whether the version was successful.

Note: You cannot hold on to the last record in a data file. If you do need to hold on to the data in the last record for your LocoMail master document to give you the correct results, then give yourself a dummy record after the last data record. A completely blank record will do. You can then hold onto the data in the last 'real' record (since this record will no longer be the last).

Making your master documents more versatile

The simple LocoMail instructions to fill in details in a standard master document allow you to create many similar documents from one master document. But you might still have to produce a number of really very similar master documents – for example, one containing a particular paragraph and one omitting it – or use LocoMail instructions to prompt you to type in these paragraphs. Neither of these methods is ideal.

This chapter illustrates some of the ways in which you make each master document much more versatile, by using instructions to:

- miss out sections of a document
- include different pieces of text in different versions

The key to varying the content of documents is the use of conditional instructions, introduced in Chapter 6. You may find it helpful to refer back to Section 6.1.2 as you follow these examples.

It must be emphasised at this point that the way to get your masters to produce the results you want is to work out carefully the conditions you are testing and the actions you choose as a result of the tests. The proper design of the masters is important too, so we recommend that you study the guidelines given in Chapter 11 as well.

7.1 Omitting sections of a document

A fairly typical requirement, when you are producing versions of a master document to suit different recipients, is to mark a section of text in your master document 'to be included only in some cases'. To do this you will need one of the conditional instructions we introduced in Section 6.1.

`(+Mail) #condition :<action> (-Mail)`

Here, the *action* is to include a piece of text, and the *condition* is the circumstances under which the text is to be included in the finished document. To ensure that LocoMail inserts the text into the finished document rather than trying to interpret it as instructions, you must precede it with a `(-Mail)` code, then follow it with a `(+Mail)` code (ie. make LocoMail 'active' then 'inactive' at the appropriate points). The instruction therefore appears:

`(+Mail) #condition :<(-Mail)text (+Mail)>(-Mail)`

and the whole instruction says 'If the *condition* is true then include *text*'.

Note how the angle brackets `<` and `>` surround the section of text that is included optionally. This becomes even clearer if you hide the **(+Mail)** and **(-Mail)** codes, giving you:

#condition : <text>

The conditions you can use in this instruction are described in Section 6.1.2.

You can include any amount of text in this type of instruction – from a character up to a page or more! So, for example, if you had a special sentence you wanted to add to a letter whenever the **product** is **SuperVac**, you might have something like this in your master document:

..(+Mail) #product="SuperVac":<(-Mail) Alternatively, the suction power of your SuperVac has not been adjusted for your type of carpets. (+Mail)>(-Mail)... which says 'if the information stored under product is SuperVac then insert the text "Alternatively, the suction..'.

Note: Of course, the *action* can also include LocoMail instructions, as you saw in Section 6.1.1. When your *action* consists both of LocoMail instructions and text, you need to make sure that the instructions are correctly separated by command separators, and that you make LocoMail active and inactive at the appropriate points. For example, a mixture of instructions and text might be laid out like this:

```
(+Mail) #condition: <←  
instruction1 (-Mail) ...text... (+Mail) instruction2 ←  
instruction3: > (-Mail)
```

An example

The use of a conditional instruction to omit a section of text is shown in OMIT.EG – one of the sample documents supplied on your LocoMail Master disc. This is a standard letter that a company might send out in reply to letters from customers who have found that their vacuum cleaners are not picking up dust very efficiently (see opposite).

You enter all the details at the start of each pass, and LocoMail inserts them into the appropriate 'blanks'. The first conditional instruction in the document means that the sentence beginning "**Alternatively...**" is only included in the finished letter when the product is a SuperVac.

The next two conditional instructions show the use of a multiple comparison in the condition you are testing. The first of these two paragraphs will only be included if the product is a SuperVac model A (ie. if the product is SuperVac AND the model = A); the second paragraph will only be included if the product is SuperVac AND the model = B. So only one of these paragraphs will end up in the finished letter.

Use LocoMail in the Fill mode with this document to see the effect of these instructions. Fill the document, first making the product SuperVac model A, then making the product SuperVac model B, then some other product, and see how the final content changes in each version.

```

A:ADVANCE/ONLY .FC Editing text. Printer id#, Using #:
PPS LSI CR40 LP6 Page 1 line 1/59
f2=Layout f3=Style f4=Size f5=Page f7=Spell f8=Options EXIT
f1=Actions

```

```

(customer = ?; Customer's name"
address = ?; Customer's address"
product = ?; Customer's machine"
model = ?; Model A or B"
(customer = ?; Customer's name"
address = ?; Customer's address"
date = ?; date of bill)"

```

Dear @MailCustomer(:Mail)^e
We are sorry to learn that your @Mail@~~email~~(:Mail) is not picking up the dust efficiently.^d

**Sentence only
included if product
is SuperVac**

Paragraph only included if product is both SuperVac and model A

Paragraph only included if product is both SuperVac and model B

We are sorry to learn that your (Mail)Product(-Mail) is not picking up the dust efficient!

The problem is likely to be one of fibre retention in the inlet section area. This is a common problem where the carpet being cleaned is either new or long haired. (Mail) ~~PRODUCT = SUPERFLO~~ (Mail) Alternatively, the suction unit Smorlee has not been adjusted for your type of carpets. (Mail) (Mail)

To clear fibres from the inlet suction area, simply press and release the lever to the right of the power button a couple of times. This is illustrated on page 3 of the instruction booklet.*

5 (Mail) product = "SuperVac-RND mode = "A" : 4
 6 Mail 0 adjust the suction power, look up the setting for your carpets in the
 7 table on page 7 of the instruction booklet. Then set the suction control
 8 lever to the correct position. (The suction control lever is situated on the
 9 cylinder body.)

(Mail) ④ product = "SuperVac-400 model = "9" ⑤
 (Mail) To adjust the suction power, look up the setting quoted for your car's weight in the table on page 10 of the instruction booklet. Then set the pointer on the electronic suction control to this number. (The electronic suction control is located directly behind the handle.)

(Mail)@ - If this doesn't clear the problem, please take your **(Mail)** **username**-(**Mail**) **hidden agent**, who will arrange for it to be serviced.
positioned directly behind the handle.

Yours sincerely

Jean Smith
Customer Services Manager

7.2 Including alternative pieces of text

An extension of the idea of having sections of text that can be missed out (as in Section 7.1) is to have a series of alternative sections of text. You might, for example, want to create a letter that gives a customer a different person to contact, depending on the part of the country they live in.

Effectively, what you need to use is a form of multiple conditional instruction that will offer a number of alternative actions, depending on which set of conditions is met.

Where a single conditional instruction can be represented as:

```
(+Mail) # condition: <action> (-Mail)
```

the LocoMail instruction offering multiple conditions is:

```
(+Mail) #condition-1: <action-1> #condition-2: <action-2> ...  
... #condition-n: <action-n> (-Mail)
```

You might find this type of instruction easier to remember if you think of the # as IF, the < as THEN and the > # as ELSE IF (a construction you probably recognise if you have done any BASIC programming).

So what this instruction says is:

'IF condition-1 is satisfied, THEN carry out action-1 and ignore the rest of the instruction. ELSE IF condition-2 is satisfied, THEN carry out action-2 and ignore the rest of the instruction...' and so on.

Again, each action can be LocoMail instructions, text, or a combination of both (with suitable command separators, Mail codes to make LocoMail 'active' and 'inactive', etc.).

IMPORTANT: The actions in this type of instruction are 'mutually exclusive' – only one of the actions can ever be carried out. The moment that LocoMail finds a condition that is true, it carries out the action, ignores the remaining options (regardless of whether they are true or not) and goes on to the next part of the document.

The effect of this multiple conditional instruction is therefore quite different to that of a series of 'IF... THEN...' instructions, used one after another:

```
#condition-1: <action-1> ←  
#condition-2: <action-2> ←  
...
```

The command separator between these instructions means that LocoMail processes each instruction separately and will perform the given action for each condition that is true.

In this section, the alternative actions are pieces of text. So the complete instruction, with all the (-Mail) and (+Mail) codes displayed, is:

```
(+Mail) #condition-1: <(-Mail) text-1 (+Mail)> #condition-2: <(-Mail) text-2 (+Mail)> ... #condition-n: <(-Mail) text-n (+Mail)> (-Mail)
```

For example, if you wanted people living in Bristol to contact Mr Jones at the Bristol branch, and people living in Manchester to contact Mr Robinson at the Manchester branch, your letter to the customers could include this:

```
(+Mail) #branch="Bristol": <(-Mail) Do contact Mr Jones at  
our Bristol branch. (+Mail)> #branch="Manchester": <(-Mail) Do contact Mr Robinson at our Manchester  
branch. (+Mail)> (-Mail)
```

Only one of the alternative sections of text can ever be included. The moment that LocoMail finds a section of text to include (ie. one for which the condition is true) it ignores the remaining options and goes on to the next part of the document.

In a number of instances, you will want LocoMail to include a special piece of text when none of the other conditions have been met. You could do this by using NOT and OR to construct the inverse condition to all the preceding condition:

```
NOT (condition-1 OR condition-2 ... OR condition-n)
```

Such a condition rapidly gets horrendously complicated so LocoMail has a special way of letting you include a 'catch-all' option in this type of instruction – to use when all the other options have been rejected. This option always goes immediately after the end of the list of alternatives as follows:

```
> #condition-n: <action-n> <catch-all-action>
```

If none of the conditions are satisfied, the catch-all action is carried out. Note how there is no #condition preceding the catch-all action.

In this example of including pieces of text, we have:

```
... > #condition-n: <text-n> <catch-all-text>
```

With the (+Mail) and (-Mail) codes displayed, this will look like:

```
... (+Mail) > #condition-n: <(-Mail) text-n (+Mail)> <(-Mail) catch-  
all-text (+Mail)> (-Mail)
```

You might find this type of instruction easier to remember if you think of the > < combination as saying ELSE – making the last part of the instruction say "...", ELSE IF condition-n is true then include text-n, ELSE include catch-all-text"

We would include this in our example if, say, people who didn't live in either Bristol or Manchester should contact Mr Smith at Head Office. Then the example would look like:

```
(+Mail)#branch="Bristol":<(-Mail)Do contact Mr Jones
at our Bristol branch.(+Mail)>#branch="Manchester":<
(-Mail)Do contact Mr Robinson at our Manchester
branch.(+Mail)><(-Mail)Do contact Mr Smith at our
Head Office.(+Mail)><(-Mail)
```

Note: A set of alternatives like these in a master document can make the master very difficult to read and understand. So what you need to do is to lay the alternatives out neatly in the master – say, one underneath another – without spoiling the way the one that is included appears in the finished version. This is best done by putting `<+Ma.i.l>` immediately after the `(+Ma.i.l)` codes and putting tabs and spaces within the instructions as follows:

```
(+Mail) ↵
#branch="Bristol": ↵
>#branch="Manchester": ↵
> ↵
> (-Mail)
<(-Mail)Do contact Mr Jones at
our Bristol branch. (+Mail) ↵
<(-Mail)Do contact Mr Robinson
at our Manchester branch(+Mail) ↵
<(-Mail)Do contact Mr Smith at
our Head Office. (+Mail) ↵
```

or, when the codes aren't displayed:

```

...↵
#branch="Bristol":↵
>#branch="Manchester":↵
>↵
>↵
>↵

```

By doing this, you get the conditions and the optional pieces of text lined up – making your master document more readable – without putting any extra ↔ into your finished documents. The ↔s we have added here have all been inserted immediately after (+Mail) codes, where they act as command separators and so are not inserted into the document. The → will also be ignored, since they appear between the elements of each instruction and not between the (-Mail) and (+Mail) codes.

An example

The LocoMail instructions for using alternative pieces of text are shown in **OPTION.EG** – one of the documents supplied on your LocoMail Master disc.

This is a standard letter that a company might send out in reply to letters from customers who are experiencing problems with a machine they have bought.

```

PRINTING INFORMATION for editing text:
Main File LST CR# LP#
(=actions) P10 L501 CR# 176
              P1=Style P4=Size P5=Page P7=Spell P8=Options P9=
              P10=Mail customer = ?; Customer's name
              P11=Product = ?; Faulty product
              P12=Branch = ?; Local branch(Mail)P
              P13=Mail
              P14=Mail
              P15=Mail
              P16=Mail
              P17=Date; today's date(Mail)P
              P18=Dear (Mail)customer(Mail)P
              P19=Thank you for your letter. I am very sorry to learn that
              P20=your (Mail)problem(Mail) is not working correctly. I can assure you
              P21=that our customers have reported very few problems with
              P22=this product.P
              P23=We suggest in the first instance that you have your (Mail)problem(Mail)
              P24=service. Take it to our (Mail)branch(Mail) branch and ask to speak to
              P25=the Service Manager. He has been advised of the problems
              P26=you have been experiencing.P
              P27=If the (Mail)branch(Mail) still fails,(Mail)P
              P28=Branch(Bristol)P
              P29=We branch=Manchester.P
              P30=We will arrange for your (Mail)problem(Mail) to be replaced at your
              P31=money refunded.P
              P32=Yours sincerelyP
              P33=Joan SmithP
              P34=Customer Services ManagerP

```

You begin by entering all the details at the start of the letter. LocoMail then inserts these details into the 'blanks'. When it comes to the group of conditional instructions, it will test what you entered as the local branch: if you typed Bristol, then the first sentence is included and other options are discarded. If you typed Manchester, then the second sentence is included and the last option discarded, and if you typed neither Bristol nor Manchester then the last sentence is included.

Use LocoMail in the Fill mode with this document to see the effect of these instructions. Fill the document making the branch Bristol, Manchester and also by making it somewhere else like York.

- *In this chapter you have seen how to include or omit sections of text through the use of conditional instructions. Typical applications of this might be in producing contracts or other documents, where the body of the document remains fairly constant, but certain clauses or paragraphs change from version to version. Another common requirement of these types of document is that the paragraphs or clauses should be numbered correctly in the finished document. How to do this, using LocoMail instructions for counting, is described in Section 8.2.*

Manipulating text and numbers with LocoMail

There will be times when you want to do more than just 'fill in the blanks' in a master document – instead, you will want to change the information in some way before inserting it.

For example, you might be creating an invoice where running totals, or other calculations, would be useful. In fact, you can also use LocoMail to carry out this type of arithmetic for you, and insert the results in the finished document!

You might also want to use the information stored in your data file in different ways – for example, your data file might have details stored under *item-names* like **title**, **initials** and **surname** and you might want to insert a person's initials and surname together into a number of 'blanks' in each version of the master documents. Of course, one way of doing this is to use two consecutive instructions at each 'blank' – but LocoMail gives you a much better way of manipulating items of information. It allows you to combine ('concatenate') information, as you will see.

Most of the instructions that fetch, store or insert information can accept information consisting of text, numbers, and mixtures of both (for example John Smith, or 24.99, or 12th June 1988). To perform arithmetic with a piece of information, LocoMail requires the information to be a 'true' number (such as 14 or 3.14159), with no text characters at all. This type of information is known as 'numeric' information – as opposed to the more general 'non-numeric' information, containing both text characters and numbers (such as £14).

This chapter describes the LocoMail instructions that manipulate numeric and non-numeric information. In particular, the sections cover:

- the difference between numeric and non-numeric information
- LocoMail instructions that can count and insert the numbers into a document
- LocoMail instructions to carry out arithmetic on numeric information and format the results
- LocoMail instructions to combine pieces of information

In most cases, you will find it much easier to work with items of information once they have been stored under *item-names*. However, you might be tempted to use some information – and in particular, numeric information – directly in your documents (such as using the value 15 whenever you need the VAT rate, rather than storing it as **VATrate** and using this *item-name*). We strongly recommend that you don't use numbers in this way.

If you get into the habit of storing all numeric information under suitable *item-names* (using instructions of the type `(+Mail) item-name=number (-Mail)`), your master documents will be much easier to understand and update. For example, if the VAT rate changed, you just need to change the `(+Mail) VAT-rate=15 (-Mail)` instruction, rather than look through the whole document for all the places where 15 is used.

Testing your instructions

It is a good idea to test the instructions you want to use in your master documents before you insert them, to make sure that they give you the results you expect. This is particularly important if the arithmetic they carry out is at all complicated.

The best way to do this is to set up a simple document containing the instruction(s) you have devised, then use LocoMail in its Fill mode and type in the information required.

If the instruction performs arithmetic, type in known values that should give a known result – then you can check that you get the right result. If the instruction is designed for counting, or combining information, check whether the information which is inserted into the finished document is correctly ordered, spaced, numbered, etc.

8.1 Numeric and non-numeric information

As far as LocoMail is concerned there are two types of information: non-numeric and numeric. Non-numeric information can be made up of any combination of LocoScript 2's available characters, but numeric information can only consist of the following characters:

- a + or – sign
- the digits 0..9
- a single decimal point

In addition, there are constraints on the size of numbers:

- The largest 'true' number that LocoMail can deal with has 18 digits before the decimal point and 9 after it (ie. 9 decimal places). If you type a bigger number than this (ie. it has more than 18 digits before the decimal point), LocoMail will treat it as non-numeric information.
- If it has more than 9 digits after the decimal point, it will still be treated as numeric but the number will be rounded to 9 decimal places (ie. digits after the 9th decimal place will be lost and the 9th decimal place will be increased by one if the 10th place was 5 or greater).

For example, the following are examples of numeric information:

15 or 19.95 or -51.24 or +33.3333.

While these would be classed as non-numeric:

John Smith or £19.95 or 15%
(text) (because of the £ character) (because of the % character)

In most respects, LocoMail doesn't distinguish between numeric and non-numeric information. Both types can be stored in a data file, or typed in response to a prompt. There is no difference in the *item-names* that the information can be stored under. In fact, the only differences in using numeric information are that you can carry out arithmetic with the numbers (as described in the rest of this chapter), and you can test a wider range of conditions using numeric values in conditional instruction (see Section 6.1.2). If you try to do these things with non-numeric data, LocoMail will display a 'Type mismatch' error message to tell you of your mistake.

Note: You can make certain that a numeric value is entered in response to a prompt, by including a # after the question mark in the LocoMail instruction (see Section 3.3.3). If the reply to the prompt is not a true number, LocoMail will 'bleep' and leave the information on the screen for you to correct.

8.2 Counting

A simple way in which LocoMail can work with and change numeric information is by incrementing or decrementing (ie. adding one to or subtracting one from) a number stored as a variable.

The instructions which increment numbers are of the form:

`(+Mail) +item-name#; comment# (-Mail)`
and
`(+Mail) item-name+1; comment# (-Mail)`

The first of these instructions adds 1 to the value currently stored in *item-name*, and then inserts this new value into the document. The second instruction inserts the current value of *item-name* into the document, and then adds 1 to it. Which instruction you use would depend on how you structure your master document.

A similar pair of instructions make LocoMail count down (decrement):

`(+Mail) -item-name#; comment# (-Mail)`
and
`(+Mail) item-name-1; comment# (-Mail)`

Again, the first of these instructions will subtract 1 from the value currently stored in *item-name* and then insert this new value into the document. The second instruction will insert the current value of *item-name* into the document, and then subtract 1 from it.

The *comment*, as usual, is optional: you might use it to remind you why the number is being incremented or decremented and included at this point in your document.

You should remember, when you come to use either of these commands in your masters, that *item-name* must already have a numeric value stored in it – otherwise LocoMail won't know what value to increment/decrement or what value to insert into the document.

So you first need an instruction that stores a number in the *item-name*:

```
(+Mail) item-name=number (-Mail)
```

This type of information-setting instruction was introduced in Chapter 3.

To show the difference between the two incrementing commands (ie. whether the `++` appears before or after the *item-name*), suppose that you had stored the number 2 in **clause number**. If LocoMail then encountered the instruction:

(+Mail) clause number+(-Mail)

the number 2 would be inserted because with this instruction, LocoMail inserts the current value then adds 1 to it. If, however, LocoMail encountered:

(+Mail)+clause number(-Mail)

the number 3 would be inserted, because LocoMail adds 1 to the current value, then inserts the new value. In both cases, the number now stored in **clause number** would be 3.

Example: Numbering clauses in a contract

Often companies use the same basic form of contract but not all of the clauses will be relevant in every case. Section 7.1 showed how to omit any clauses that don't apply, by using conditional instructions: however, simply omitting clauses you don't want might leave a contract with clauses numbered 1, 2, 4, 7 etc. when what you really need is to renumber the clauses that remain 1, 2, 3, 4, etc. This section explains how you can use the counting instructions to number the clauses consecutively.

To show you how to do this, we will use **CLAUSETG** – one of the documents in the **ADVANCED** group on your LocoMail example documents disc. Start by making a copy of this document, giving it a name such as **RENUMBER.EG1**, so that you will recognise the document at a later date.

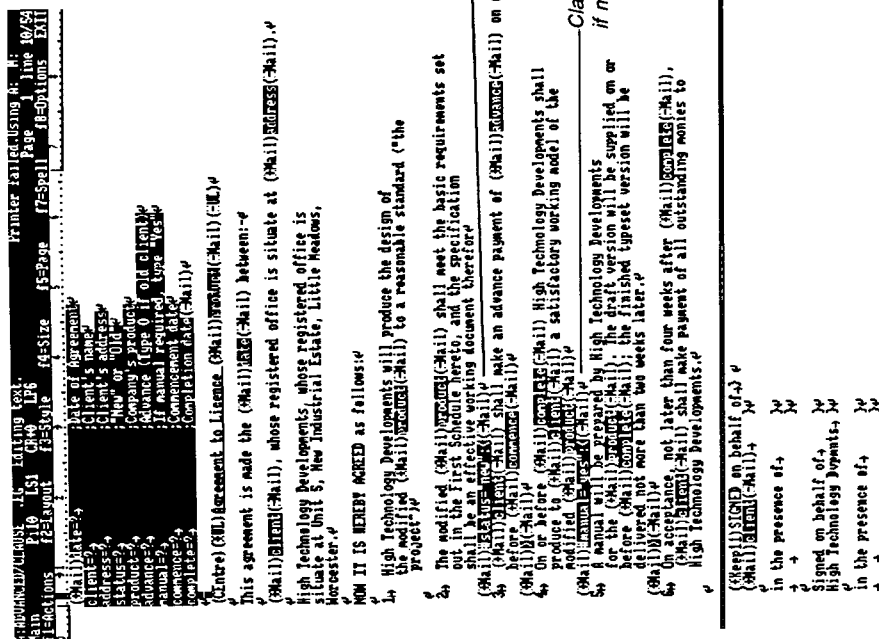
When you have made the copy, edit it. It will appear as shown opposite:

As you can see, the clauses in this master have been numbered. However, Clause 3 of this particular contract is only relevant if this is the first contract between you and this client, and Clause 5 is only relevant if the client wants you to supply a Manual for the product. So both of these clauses have been marked as optional by putting them as the action in a conditional instruction:

```
(+Mail)#status="New":<(-Mail)...clause 3...(+Mail)>(-Mail)
(+Mail)#manual="Yes":<(-Mail)...clause 5...(+Mail)>(-Mail)
```

The use of this type of conditional instruction was described in Chapter 7.

To see what happens when you fill in the master document, return to the Disc Manager Screen (by pressing **EXIT**) then selecting **Abandon edit**, since you haven't made any changes). Press **F** to select the Fill mode, then fill the document – typing **old** in reply to the prompt about the status, and **No** for the manual required prompts. Print out the finished version. You can see that the clauses included in the finished version are numbered 1, 2, 4, 6, 7...



The way to make the clause numbers consecutive throughout the contract (irrespective of whether either of these clauses is omitted) is to replace the actual numbers marking each clause by either:

```
(+Mail) clause_no+ (-Mail)
(+Mail) clause_no (-Mail)
```

or

Note: You must mark **all** the clauses with these instructions – not just the ones that are optional. In addition, whichever type of instruction you choose to use, you must be consistent throughout the document. If you include both types then as LocoMail works through the document the clauses won't be numbered consecutively – in particular, some of the clauses will be given the same numbers.

You now just need to tell LocoMail what initial value to give to **clause_no**.

- If you use the 'clause_no+' type of instruction, the initial value you set for **clause_no** is the *first value you insert in the document*. Typically, this would be 1, so somewhere before the first clause you need the LocoMail instruction:

```
(+Mail) clause_no=1 (-Mail)
```

For example, you might put this instruction immediately before the first clause:

```
NOW IT IS HEREBY AGREED as follows:
+
(Mail) clause_no=1
(Mail) clause_no+
(Mail) High Technology Developments will produce the design of
the modified (Mail) address (Mail) to a reasonable standard ("the
project")
+
```

- If you use the 'clause_no' type of numbering instruction, the initial value you will have to set for **clause_no** is *one less than the first value you want to insert*, because this instruction will add 1 to **clause_no** before inserting the number. So somewhere before the first clause you need the LocoMail instruction:

```
(+Mail) clause_no=0 (-Mail)
```

You might put this at the end of the initial block of information gathering instructions:

```
complete=2; completion date
clause_no=0 (-Mail)
(Enter) (fill) Agreement to Licence (Mail) address (Mail) (fill)
This agreement is made the (Mail) day (Mail) between:
(Mail) address (Mail), whose registered office is situated at (Mail) address (Mail)
High Technology Developments, whose registered office is
situated at Unit 5, New Industrial Estate, Little Meadows,
Worcester.
NOW IT IS HEREBY AGREED as follows:
(Mail) clause_no+
```

To see how these commands work in practice, save RENUMBER.EG1 with these amended instructions in it and then fill it in a few times – setting **status** to New or Old and **Manual** to Yes or No. Whatever combination you choose, LocoMail should always produce finished documents with clauses numbered consecutively.

Note: If you have references within clauses to other clauses (for example, 'See Clause 7'), this will require you to build some more LocoMail instructions into your master document. If the references refer back to earlier clauses, you can put a LocoMail instruction at the earlier point that remembers the clause number at that point – through instructions like:

```
(+Mail) clause=clause_no (-Mail)
```

which stores the current value of **clause_no** as **clause** (see Chapter 3). If you want to put in forward references, you will need to calculate what the clause number will be – by using the LocoMail instructions which we shall introduce in the next section.

8.3 Doing arithmetic

The operations which LocoMail can carry out on numeric information are the standard arithmetic ones, shown below (with the symbols used to perform these operations):

```

+      addition
-      subtraction
*      multiplication
/      division

```

This means, for example, that you can get LocoMail to do many of the calculations involved in invoices, say, multiplying the unit price of an item by the number required, adding up running totals of VAT and net costs, etc.

Any arithmetic you want to do using these operators must be enclosed within square brackets. For example:

- if you want LocoMail to carry out the calculation inside the square brackets and insert the answer in the document, you use an instruction of the form:
(+Mail) [arithmetic] (-Mail)
(arithmetic) represents the chosen arithmetic operation.)
- if you want LocoMail to carry out the calculation inside the square brackets and store the answer without inserting it, you need to use an instruction of the 'item-name=' type.
(+Mail) result=[arithmetic] (-Mail)
- If you want to store the result for later and insert the result, you need to use two instructions with a suitable command separator, ie:
(+Mail) result=[arithmetic] ; result (-Mail)

So, for example, to insert the total cost of a **number** of items at a particular **unit_cost**, you might have the instruction:

```
(+Mail) [number*unit_cost] (-Mail)
```

But, if you wanted to store the total cost, as **total** as well, you would use:

```
(+Mail) total=[number*unit_cost]:total (-Mail)
```

When working out the result of any arithmetic expression between square brackets, LocoMail follows the usual rules of arithmetic – it first works out any multiplications and divisions (working from left to right) and then works out any additions and subtractions (again working from left to right). If LocoMail doesn't seem to be giving you the correct answer to a complicated calculation, you may need to add extra square brackets to clarify what you want it to do. For example, if you wanted to add together two values and then multiply by the vat rate, you would want a LocoMail instruction like this:

```
(+Mail) total_vat=[(cost1 + cost2)*vat_rate] (-Mail)
```

The bracketing makes it clear to LocoMail that you want to multiply the total amount by the vat rate. If you didn't put the brackets in around the **cost1 + cost2** addition, you would get the wrong answer because when LocoMail applies the rules of arithmetic without the inner set of brackets to guide it, it will first multiply **cost2** by **vat_rate** and then add **cost1**.

Note: Remember that LocoMail can only do arithmetic on numeric information, or using items which are currently storing numeric information. For example, if you try to carry out a calculation such as:

```
(+Mail) [number*unit_cost] (-Mail)
```

and **unit_cost** held £1.20, rather than 1.20, LocoMail will display a 'Type mismatch' error message – the £ character means that **unit_cost** isn't storing a true numeric value. Whenever you need numeric information to be typed into your master document which you intend to store and use later in a calculation, use the **..?#..** type of instruction. For example, the instruction requesting the unit cost of the item might be:

```
(+Mail) ? # unit_cost;Type in the price per item (-Mail)
```

When you are fetching values from a data file, you must also be certain that the item fetched is numeric information.

You can, of course, carry out much more complicated arithmetic than just adding or multiplying two numbers together, but in these cases you may need to put square brackets around the separate arithmetic expressions that make up the complicated calculation.

8.4 Formatting the results of arithmetic

All the calculations we have described so far have given us numbers laid out according to a standard set of rules – rounded to 9 decimal places; minus signs before negative numbers, no sign in front of positive numbers; no grouping of the digits in large numbers. This is the default format that LocoMail gives to all numbers it calculates.

These characteristics won't always be what you want. Financial calculations in pounds, for example, are generally meaningless if they are given to more than two decimal places (ie. pence). If you don't want to use LocoMail's default format, you can specify alongside any arithmetic exactly what format you want the result to have – both for inserting in the document and for future arithmetic.

To specify the format, you separate details of the arithmetic from details of the format with a special character – the **|** character. Your arithmetic instruction will now look like this:

```
(+Mail) [arithmetic|format] (-Mail)
```

(The **|** character appears on the PCW9512 keytops. If you have a PCW 8256 or 8512, the **|** is typed by holding down the **[Extra]** key and pressing the **\$** key.)

The *format* consists of a sequence of special characters, specifying:

- how many decimal places you want the answer to
- whether the result is to be rounded (ie. taken to the nearest whole number) or truncated
- whether positive numbers are to have a **+** in front of them
- whether you want to use a full stop or a comma as the decimal point
- whether you want to split large numbers into groups of three digits

The following sections describe the special characters you must use in the *format* part of the instruction for each case. The order that these characters appear in is not important to LocoMail, so long as they appear after the **|**.

IMPORTANT: The *format* you specify within any square brackets will only apply to arithmetic enclosed within that same set of brackets. You can specify different *formats* within different sets of brackets, for example:

```
(+Mail) [(cost1+cost2|format1)*vat|format2] (-Mail)
```

Here, *format1* will apply to the result of **cost1+cost2**, then *format2* will apply to the multiplication of that result with **vat**. Being able to apply different *formats* at different points in a calculation will help you with financial calculations, in which the stage at which you round or truncate the numbers you calculate can be very important.

8.4.1 Getting the right number of decimal places

Simple addition and subtraction of prices will always give answers in pounds and pence, but multiplication and division can give results like £1.333333333 which are plainly wrong on an invoice. What you want is this figure rounded to the nearest number of pence – ie. £1.33.

This is done simply by typing the number of decimal places you require after the !. You can have any number between 0 and 9.

For example, suppose you want the item **share** to hold the result of dividing a cost by 3, to two decimal places. The instruction you would use would be:

```
(+Mail) share=[cost/3|2] (-Mail)
```

This will calculate the result of cost divided by 3, round it to two decimal places, then store it in **share**. So, for example, if you set **cost** to 5.00 (ie. £5), the result of the arithmetic would be 1.666666666 rounded to the nearest penny – which is 1.67 (ie. £1.67). **Share** will then store 1.67.

For VAT calculations, you don't want to round to the nearest penny – instead you must quote to the nearest penny below the calculated amount. In other words, you need to discard the part of the answer after the second decimal place – this is described as truncating the result to two decimal places.

To tell LocoMail that you want to truncate the result of a calculation, you put an exclamation mark after the number of decimal places. So, in our example, if you wanted the result truncated to the nearest penny, you would use an instruction like:

```
(+Mail) share=[cost/3|2!] (-Mail)
```

If we started with **cost** at 5.00 as before (ie. £5), **share** this time would be 1.666666666 truncated to the nearest penny – which is 1.66 (ie. £1.66).

Note: LocoMail will automatically apply rounding to values that it calculates (ie. it discards the digits after the number of decimal places you request, increasing the last remaining digit by 1 if the first of the discarded digits was 5 or more), unless you specify that you want the result truncated.

8.4.2 Putting in plus signs

Another refinement you might like to make to this number is to add a plus sign or an extra space in front of the number if it is positive. (LocoMail automatically puts a minus sign in front of negative results, so the space character can be useful in aligning results in your documents.) This sign or the blank space will be included when you insert the result in the document.

When you want to precede positive numbers with a + sign, you include a + character in the *format*. So if you wanted this in the above example, you would make your instruction:

```
(+Mail) share=[cost/3|+2!] (-Mail)
```

This time, if we were to insert **share** into our document we would see +1.66.

If you want LocoMail to put a space before positive numbers, you precede the number of decimal places you want by a -:

```
(+Mail) share=[cost/3|-2!] (-Mail)
```

Note: If you simply want to position numbers properly in your documents, remember you can always use LocoScript's decimal tabs to line numbers up – decimal tabs are described on page 120 of the LocoScript 2 User Guide and page 154 of the PCW9512 User Guide.

8.4.3 Decimal points

You can specify whether you want LocoMail to mark the decimal point with a comma by including the comma in the *format*. For example, if you want LocoMail to insert **share** using a comma to mark the decimal point, your instruction would say:

```
(+Mail) share=[cost/3|2,!]:share (-Mail)
```

Then, if **cost** is 5.00, the figure 1, 66 would appear in your document.

Note: If you don't specify a comma, LocoMail will use the decimal marker set in the Layout in the master document.

8.4.4 Grouping large numbers

When the numbers you are handling are really large – thousands or millions – you may want to present these numbers with the digits grouped in threes. Again, you use a special character in the *format* part of the instruction to do this.

The character you use in the *format* depends on whether you want to use spaces or commas (full stops, if you are using a comma as the decimal point) to group the digits. Use:

```
\ if you want to use spaces, eg. 2 500 000
```

```
/ if you want to use commas, eg. 2,500,000 (or 2.500.000 if commas are marking decimal points)
```

```
(\ is typed by holding down the [ALT] key and pressing %, whether your machine is a PCW 8256, 8512 or 9512).
```

Note: If neither \ or / is used in the *format*, then the number will be printed without any spaces between the digits.

For example, suppose your master document recorded the total salary bill for your company. You could print this total with the digits grouped in threes by commas with the simple LocoMail instruction:

```
(+Mail) [total|/] (-Mail)
```

Suppose **total** was (£)350000. If you gave **total** this *format* and then inserted it into your document, what you would see would be 350,000.

An example

As an example of using LocoMail's arithmetic instructions, study the ACCOUNT.EG document supplied on the LocoMail example documents disc. When you fill this document, you will see instructions that put the results of calculations straight into the document, ones that save the results for use later and ones that format the answers in various ways.

```

PRINTING ACCOUNT TO dating text:
Main PPS ISI CHKO LP6
F1=Options F2=Layout F3=Style F4=Size F5=Page F7=Spell F8=Options EX11
Using Wk: 1/54
Printer title Page
Invoice Number (#Mail) Invoice number (#Mail)*
TotalE(#Mail)
To: #
(#Mail)% Customer's name(#Mail)*
(#Mail)% Customer's address(#Mail)*
For supplies+ registration number,*
Basic cost,*
Total = (total + cost) * (#Mail)*
Extras = (total + extras) * (#Mail)*
discount = (total + 0.09 / 2) * (#Mail)*
Discount (#Mail)*
Total = (total - discount) * (#Mail)*
Total less VAT,*
VAT = total * 0.15 * (#Mail)*
total = (total + vat) * (#Mail)*
vat = vat * total * (#Mail)*
VAT @ IS4 → (#Mail)VAT(#Mail)*
total = total + vat * (#Mail)VAT(#Mail)*
Total = total + vat * (#Mail)VAT(#Mail)*

```

8.5 Combining items of information

The ability of LocoMail to carry out arithmetic on purely numeric information is very useful. However, when you insert totals into invoices you usually want to precede the total you have calculated with a pound sign (or another currency symbol). All the way through the calculations, of course, the total has been numeric.

One way of including the pound sign is to put it into the master document itself, using LocoScript's tabs etc. to get the position right.

Another solution is to 'add' the £ character to the result of the calculation. LocoMail will allow you to combine items of information using a process called 'String concatenation' – in fact, using this process you can combine any pieces of information, whether pieces of text, numeric information, information stored under an *item-name*, etc. The type of instruction you use to do this:

```
(+Mail) result=sring-1 & sring-2 {& sring-n...! (-Mail)
```

where [*& string n...*] means add as many of these *&string* combinations as you need.

Note that this instruction only stores the information – to insert it in the document you then have to use **(+Mail) result (-Mail)**.

Each of these *strings* can be:

- **either** a literal string of characters, spaces and codes, written between double quote marks – for example **"Mr. "**
- **or** the *item-name* used to store a piece of information – for example **name**
- **or** a formatted arithmetic expression between square brackets – for example **[cost1+cost2]** (using the default format) or **[cost1+cost2|2!]**

The following two examples are typical of how combining items of information can give useful results:

- **Combining text with numeric information**

Suppose you want to put a pound sign before the total of **cost1** and **cost2**. You could do this with the combined instruction:

```
(+Mail)total="£" & [cost1+cost2]:total(-Mail)
```

The first instruction (before the colon) tells LocoMail to calculate the sum of **cost1** and **cost2**, and then adds what's between the quote marks to the front of the result. This result is stored in **total**. The second instruction (after the colon) simply tells LocoMail to insert what is stored in **total** into the document.

Combining information stored under item-names

A common problem when using information from data files is that the information is not divided up ideally for your purposes, or you want to use the same information in small segments in one place and as a single item elsewhere in the document. In particular, data files often contain two or more separate items where you would like to use them as one.

You can, of course, design your master document to use the items of information in exactly the way they are specified in the data file – carefully inserting, for example, a person's title, then a space, then their initials, then a space, and finally their surname everywhere you want the person's full name.

A better solution is to combine all these pieces of information (and the spaces you want to put between them) together into one single piece of information.

For example, you might want to combine the data items **title**, **inits** and **name** (such as might be present in a data file) into the single data item **fullname** to make it easier to insert into the master.

The instruction to use would be:

(+Mail) fullname=title & " & inits & " & name (-Mail)

Note that the " "s have been included to space out the elements of the full name: without these spaces you are likely to get full names like **MrANSmith**, instead of the rather neater **Mr AN Smith**.

Once you have concatenated the pieces of information into one item, you can still use each item separately but in addition you can use the item **fullname** whenever you want to insert the combined information. So, for example, where you might have had the following:

(+Mail) title(-Mail) (+Mail) inits(-Mail) (+Mail) name(-Mail)

you can now use

(+Mail) fullname(-Mail)

Note: You can only concatenate information that is presented as a string (ie. a sequence of characters etc. between double quotes), information that is already stored in memory, or information that is stored in the data file. You cannot take information from the keyboard, and combine this with other information, all in the same LocoMail instruction. You must enter and store the information first and then combine it.

Chapter 8: Manipulating text and numbers

If you have already worked out **total** and stored it, you can still add the £ sign to the front of the value. Then you would use an instruction like:

(+Mail) total="£" & total:total (-Mail)

Note: In these examples, the value now stored in **total** after combining the £ and the calculation is no longer a numeric value: in other words, once the £ character has been added, **total** cannot be used again in any arithmetic. If you intend to use **total** again in arithmetic, then you should either include the £ character directly in the master document, or use an instruction like

(+Mail) current_total="£" & total:current_total (-Mail)

You can then continue using **total** in arithmetic: only **current_total** is non-numeric.

An example

The ACCOUNT.EG document as supplied on the LocoMail samples disc shows text characters being added to numeric information by this method. A copy of the complete document was shown in Section 8.4.4. Below, you can see the concatenation instructions indicated.

```

ACCOUNT.EG document as supplied on the LocoMail samples disc shows
text characters being added to numeric information by this method. A copy of the
complete document was shown in Section 8.4.4. Below, you can see the
concatenation instructions indicated.

To:
(+Mail) Customer's name(+Mail)
(+Mail) Customer's address(+Mail)
For supplying: (+Mail) Date of car(+Mail)
Registration number: (+Mail) registration number(+Mail)
Basic costs: (+Mail) cost: basic costs
Extra: (+Mail) total + costs(+Mail)
Total = total + extra: total of extras £ sign inserted into document
Discount (2%): (+Mail) total + extra: discount(+Mail)
Total = total - discount(+Mail)
Total less VAT: (+Mail) total + extra: VAT(+Mail)
VAT = total + extra: VAT(+Mail)
Total = total + VAT(+Mail)
VAT @ 15%: (+Mail) total + extra: VAT(+Mail)
Total = total + VAT(+Mail)
Concatenates string and arithmetic expression
(+Mail) total="£" & total:total (-Mail)

```

An example

The STRINGS.EG example document, supplied on your LocoMail disc, shows how information can be read in small units, combined into a larger string and then used in both ways.

PRINTING/STYLING OF Editing text. Printer failed using a. H. line 19/54
 f-functions f2=sort f3=style f4=Size f5=Page f7=Spell f8=Options f11=

(Chail) rates;
 = 0; colour = 0; year = 0; Year of manufacture;
 size = size & "cc";
 size = size & "cc";
 doors = 0; Number of doors (number only);
 hatch = 0; Hatchback 0; type V or W;
 roof = 0; Sun roof 0; type V or W;
 lock = 0; Central locking 0; type V or W;
 keys = 0; Keyless type 0; remote;
 number = 0; Number of previous owners;
 hatch = 0; (: doors = doors & hatchback = 0;
 features = 0; (: features = "Sun roof" & "
 lock = 0; (: features = "features" & "and
 features = 0; (: features = "features & Central locking";
 rust = 0; (: condition = "Very good"; condition = "Excellent";
 owners = previous owners;
 number = 1; + (: owners = "careful owner";
 owners = number & owners;
 description = year & " & make & " & model;
 (Chail) For Sale (Bold) P
 (Chail) description (Chail) (Chail) doors (Chail) (Chail) size (Chail)
 (Chail) features (Chail) (Chail) condition (Chail) condition. (Chail) owners (Chail) P
 Price: £ (Chail) price (Chail) P

Entering details
 and storing under
 item-names

Concatenating item-names
 and strings

Using groups of LocoMail instructions

There are many occasions when it is useful to 'group' LocoMail instructions together:

- where there is a sequence of instructions in the middle of a master document, breaking up the structure of the master and making it difficult to understand
- where you carry out different sequences of instructions depending on the value of a number of different items – ie., when the alternative is to nest complex sequences of instructions within a range of conditional instructions
- where you have instructions that carry out an action that you want to repeat a number of times – for example, printing the entries on an invoice

LocoMail lets you store a sequence of LocoMail instructions (and text) under a 'program-name', then use this name wherever you want to use the instructions. Such a sequence of instructions is called a 'Program Unit', and typically each program unit will contain all the instructions you need to carry out a particular task – such as printing one of the entries on an invoice.

Using program units gives you clearer, more understandable masters (particularly if you choose names for the program units that reflect the purpose of the instructions), and reduces the need for the same instructions to appear at different places in the master.

More importantly, once instructions are grouped into a program unit, you can repeat the same sequence of instructions over and over again – either a fixed number of times (if you know in advance how many times you need to repeat these instructions) or you can arrange that LocoMail repeats the instructions until some condition is met. The sort of 'loops' in your LocoMail master that this produces will be very familiar to anyone who has written computer programs, for example, in BASIC.

Repeating program units of instructions is ideally suited to preparing documents which include lists of items – for example, invoices or lists of names and 'phone numbers. All you need to do is set up the instructions necessary to produce one line of the list, group these into a program unit and then repeat this program unit until the complete list has been built up. Doing this type of task any other way is extremely hard work!

In this chapter, we describe:

- how to put instructions into a program unit
- how to tell LocoMail to carry out the program unit
- how to repeat a program unit – including how to tell LocoMail the number of times to repeat it

Note: Repeating groups of instructions is a very common action in computer programming. If you find this chapter difficult to understand, ask the help of an experienced programmer – programmers know all about loops.

9.1 Setting up a program unit

A program unit is simply a sequence of LocoMail instructions, text, codes, etc. grouped together under a program-name.

To set up a program unit in a master we use the instruction

```
(+Mail) program-name="string" (-Mail)
```

This is exactly the same instruction that you may well have used elsewhere to store text under an item-name. Here the *string* will be made up of a number of LocoMail instructions, possibly with some text as well, but it is essentially no different from any other sequence of characters and codes that you might use this instruction to store.

Similarly, there is nothing special about the names that can be used in this instruction. The name of a program unit is simply an item-name and so has to obey the rules given in Section 2.1.1. We do, however, strongly recommend you to pick program-names that describe the job the program unit does.

For example, suppose your master document sets up a list of the names and phone numbers of the members of your club. For each entry on the list, you need LocoMail to:

- ask for and insert the member's name
- insert a tab
- ask for and insert a phone number, and finally
- start a new line

The sequence of commands you use to do this might be:

```
(+Mail)?;Name?(-Mail)→(+Mail)?;phone no.?( -Mail)↵
```

Ask & insert name Insert Tab Ask & insert number Start new line

and you can group this sequence into a program unit called *memb_name* by using the instruction:

```
(+Mail)memb_name="(+Mail)?;member's  
name?(-Mail)→(+Mail)  
?;phone no.?( -Mail)↵"  
(-Mail)
```

When LocoMail meets an instruction like this, it will simply read all the details within the double quotes and store them under the specified *program-name*. The instructions themselves aren't carried out at this point and, in particular, the Mail codes within the quote marks aren't acted upon – just 'remembered'. How you then use this *program-name* at the point in the master you want the instructions to be carried out is described in Section 9.2.

This is a very simple example – two LocoMail instructions and a tab. This group probably wouldn't be obtrusive in your master and you would only put such a simple group of commands into a program unit when you want to repeat them – as in this example. In a more typical example, you could well have many more instructions than this. (There is a limit on the number of instructions you can have in a program unit but it is very large and so you are never likely to meet it!)

You would also tend to have rather more complicated commands than these, involving any of the LocoMail instructions you've seen in Parts II and III – ones that take information from the keyboard or from a data file, conditional instructions, \$+ (the command that advances to the next record), arithmetic, etc.

For example, if you wanted to produce an invoice with entries like:

2 →	claw hammers →	2.50 →	5.00 ↵
	no. of	description	unit price
	items		calculated
			total

Then the sequence of instructions you need to put into a program unit might be:

```
(+Mail)?number;No. of items(-Mail)→(+Mail)?;description  
(-Mail)→(+Mail)?item;item price?(-Mail)→(+Mail)  
price=[number*item[2]:price(-Mail)↵
```

As always when you have a complex sequence of instructions to handle, good layout will help you both to set up your program unit correctly in the first place and, if necessary, to change it. So when you come to put these instructions into a program unit called, say, *item_line*, don't simply type a massive instruction like this:

```
(+Mail)item_line="(+Mail)?number.....:price(-Mail)↵"  
(-Mail)
```

Instead, lay the instruction out in such a way that you can readily see what it does – for example:

```
(+Mail)item_line="(+Mail)↵  
? number→ ; No. of items→ (-Mail)→ (+Mail)↵  
? → ; description→ (-Mail)→ (+Mail)↵  
? item→ ; item price→ (-Mail)→ (+Mail)↵  
price=[number*item[2]↵  
price→ → (-Mail)↵  
"(-Mail)
```

Program units always have to be defined before they are used. As LocoMail works through each master document from the top, this means that the instruction setting up a program unit must always appear earlier in the document than the

first instruction that uses the program unit. They are not placed at the end of the document like the subroutines in a BASIC program. The best place for your program units is at the start of your master document.

IMPORTANT: Not all sequences of commands that you put into a program unit start with a (+Mail) code and finish with a (-Mail) code. How they start depends on whether you need to make LocoMail 'active' or 'inactive' at the point at which the program unit is used. How they finish depends on whether you need to make LocoMail 'active' or 'inactive' after the program unit has been used. If a program unit is used repetitively, then even more care needs to be taken to ensure that the program unit makes LocoMail 'active' and 'inactive' at the right points. We explain this further in the following sections, in which we describe how program units are used.

9.2 Using the program unit once

At the point in the master where you want LocoMail to carry out the instructions stored in the program unit, you use the 'Perform' instruction:

```
*program-name
```

(where *program-name* is the name of the program unit.) The **program-name* instruction essentially tells LocoMail 'Perform the program unit *program-name*'.

LocoMail then works through the commands stored in the program unit as if they had been inserted into the master document after the next (-Mail) code, or a ←, whichever of these it finds first. If the Perform instruction is followed by a : or a ←, then LocoMail is 'active' as it starts to read the program unit and so treats the first characters it reads in the program unit as commands. If the Perform instruction is followed by a (-Mail) code – even one that is immediately followed by a (+Mail) code – LocoMail will be 'inactive' as it starts to read the program unit.

This affects how you construct your program unit. If the first part of the program unit is a LocoMail instruction, you have to ensure that LocoMail is active when it reads it; if the first part of the program unit is some text to be inserted, then you have to ensure that LocoMail is 'inactive' as it reads this. The only certain way to do this is to think through the effect of each command separator and Mail code in the master document and in the program unit, but a useful rule of thumb to start with is:

- If the program unit starts with a LocoMail instruction, always start the program unit with a (+Mail) code: this will ensure that LocoMail is active regardless of whether the Perform instruction is followed by a (-Mail) code, a : or a ←.
- If the program unit starts with text, then follow the Perform instruction with a (-Mail) code to ensure that LocoMail is inactive when it starts reading the program unit.

Similar considerations apply when you work out how to finish the program unit. There, the next part of the master document simply follows straight on from the last part of the program unit. The command separator or (-Mail) code that follows the Perform instruction has been processed already and is not processed again here. Any command separators or Mail codes that are needed to ensure that LocoMail interprets your master document correctly have to be put into the program unit itself.

Note: Before you put your program unit and the Perform instruction into the 'real' master document, it is always best to test these in a separate test document. Testing is a two-stage process: the first stage is to test that the sequence of instructions you want to use in your program unit actually work outside of your program unit; when the instructions all work, then put them into your program unit and test this.

It is quite difficult when LocoMail finds an error in a program unit to identify precisely where the error has occurred; if you know that the instructions in the program unit work correctly, the error must be as LocoMail starts to process the program unit or as it finishes it – it can't be in the middle.

Moreover, because of the way LocoMail handles program units, it is very important that your test document uses the program unit in the same way as it will be used in the final master document. In particular, you must make sure that the Perform command is followed by a (-Mail) code or : or ←, exactly as in the final master and that it is followed by text or another instruction in exactly the same way. Otherwise, your testing may be in vain!

For example

To show how important it is to work out which Mail codes you need in your program unit, let's think how the program unit *item_line* that we described in Section 9.1 might fit into a master document.

```
(+Mail) item_line=" (+Mail) ←
? number→ ; No. of items→ (-Mail) → (+Mail) ←
? → ; description→ (-Mail) → (+Mail) ←
? item→ ; item price→ (-Mail) → (+Mail) ←
price=[number*item!2] ←
price→ → (-Mail) ←
" (-Mail)
```

If the Perform instruction comes in the middle of some text, it will appear in the document as:

```
...text... (+Mail) * item_line (-Mail) ...text2...
```

In this case, LocoMail will process the program unit as if it were inserted in the document after the (-Mail) code – ie. as if what you actually had in the document was:

```

...text1... (+Mail) % item_line (-Mail) (+Mail) ←
? number→ ; No. of items→ (-Mail)→ (+Mail) ←
?→ ; description→ (-Mail)→ (+Mail) ←
? item→ ; item price→ (-Mail)→ (+Mail) ←
price=[number*item|2] ←
price→ → (-Mail) ←
...text2...

```

Working through this sequence of instructions and text, you can see that LocoMail is indeed 'active' when you want it to carry out LocoMail instructions and 'inactive' when you want it simply to insert text.

A Perform instruction followed by a (-Mail) code is always the easiest case to handle and to understand. When you follow the Perform instruction by one of the command separators : or ←, LocoMail is already active when it starts to process the program unit. In this case, you need to think through very carefully the various changes LocoMail makes between being active and being inactive. We look at this in detail in Section 9.5.

When you are new to using program units and Perform instructions in your master documents, we suggest that you always separate your Perform instructions from any following LocoMail instruction by (-Mail) (+Mail). Your program units should then work in the way that you expect. If they don't, turn to Section 9.5.1: this section explains both the types of problem you may run into and what you do to solve these problems.

• The *item_line* program unit can be seen in action in a Perform instruction in the *PERFORM.EG* master document in the *ADVANCED* group on the *LocoMail Examples* disc.

The instructions that make up item_line are also used outside of any program unit in NOTFORM.EG. This master document gives exactly the same results as the PERFORM.EG master but it is much less readable. Improving the readability is a major reason for using a Perform instruction.

9.3 Repeating program units

Using the simple *%program-name* instruction is fine when you only want to carry out the program unit once. Typically, though, the actions you put in the program unit need to be carried out a number of times – for example, inserting all the different items that need to be included on an invoice. If you have programmed in BASIC (for example), you will recognise this as the looping action that is useful in all sorts of programs.

What you need here is a way of repeating the Perform instruction as many times as necessary.

If the instructions always had to be carried out three times (say), this could be done by using three Perform instructions one after the other in the master – separated by (-Mail) (+Mail) pairs so that each Perform instruction works in

the way you expect. (See Section 9.5.) However, it is more usual to need to vary the number of times the program unit is used according to the details of the actual job you are doing. For example, the invoice you want to produce could be for two items or it could equally well be for six items: your master document must be able to handle both cases.

To allow you to vary the number of times a program unit is used, the LocoMail instructions for repeating program units specify an item which is to be used as a 'test variable'. Repetition of the program unit stops when the value of this test variable is zero or null (ie. no characters). You just need to write your program unit in such a way that this test variable becomes zero or null when the program unit has been repeated the right number of times. We explain how to do this below. First we look at the instructions that are used here and how they operate.

The Loop instructions

The instruction that repeats a program unit has two forms:

- one which tests the current value of the test variable before carrying out the program unit: if you have programmed in BASIC, you will recognise this as WHILE...WEND.
- one which tests the current value of the test variable at the end of processing the program unit and decides on the basis of the test whether to repeat the program unit – which you may recognise as REPEAT...UNTIL.

The instructions are:

```

(+Mail) %program-name@test (-Mail)
and (+Mail) @test%program-name (-Mail)

```

The first of these instructions tells LocoMail to repeat the program unit with the given *program-name* until the value held as *test* is null or zero; this is the REPEAT...UNTIL version of the instruction. The second instruction tells LocoMail while the value held as *test* is not null or zero, repeat the program unit with the given *program-name*; this is the WHILE...WEND version of the instruction. The *@test* part of the instruction effectively tells LocoMail how many times to repeat the program unit.

Which of these instructions to use depends on how you want your LocoMail program to behave at the start of the repeated loop. If you want the program unit not to be executed at all unless the conditions are right, then you must use *@test%program-name* (the 'WHILE...WEND' form of the loop) which makes the test first. If, however, you want the program unit to be repeated at least once regardless of the initial value of the test variable, then you need *%program-name@test* (the 'REPEAT...UNTIL' form of the loop) which makes the test after the program unit has been carried out.

Before you can use either of these instructions in a master document, you need to understand how LocoMail interprets them and you need to know how to set up a suitable test. The *@test* part of the instruction needs to be chosen very carefully, to make sure that LocoMail repeats the program unit the correct number of times. The way LocoMail interprets the instructions is explained below, while the way you go about selecting a suitable *test* is explained in Section 9.4.

For example

Once again, we will take as our example the `item_line` program unit:

```
(+Mail) item_line="(+Mail) ⚡
? number→ ; No. of items→ (-Mail)→ (+Mail) ⚡
?→ ; description→ (-Mail)→ (+Mail) ⚡
? item→ ; item price→ (-Mail)→ (+Mail) ⚡
price=[number*item|2] ⚡
price→ (-Mail) ⚡
"(-Mail)
```

If the Loop instruction comes in the middle of some text, it might appear in the document as:

```
...text/...(+Mail) & item_line @ test (-Mail) ...text2...
```

For the moment, we won't worry about what we need to use as the test: we shall just check how this program unit will link up with the other parts of the master document and how one loop through the program unit follows on to the next.

When LocoMail first processes this loop, it works through the program unit as if it were inserted in the document after the `(-Mail)` code - ie. as if what you actually had in the document was:

```
...text/...(+Mail) & item_line (-Mail) (+Mail) ⚡
? number→ ; No. of items→ (-Mail)→ (+Mail) ⚡
?→ ; description→ (-Mail)→ (+Mail) ⚡
? item→ ; item price→ (-Mail)→ (+Mail) ⚡
price=[number*item|2] ⚡
price→ (-Mail) ⚡
```

This is exactly how the simple Perform instruction described in Section 9.2 worked. The difference comes when LocoMail has read all of the program unit and needs to know whether to repeat it. If `test` is null or zero, then it goes straight on to whatever follows the loop instruction - in this case, some text. Again, the join between these two is exactly the same as that between the Perform instruction and the following text. However, if the test is not null or zero, LocoMail effectively processes:

```
...text/...(+Mail) & item_line (-Mail) (+Mail) ⚡
? number→ ; No. of items→ (-Mail)→ (+Mail) ⚡
?→ ; description→ (-Mail)→ (+Mail) ⚡
? item→ ; item price→ (-Mail)→ (+Mail) ⚡
price=[number*item|2] ⚡
price→ (-Mail) ⚡
(+Mail) ⚡
? number→ ; No. of items→ (-Mail)→ (+Mail) ⚡
?→ ; description→ (-Mail)→ (+Mail) ⚡
? item→ ; item price→ (-Mail)→ (+Mail) ⚡
... (and eventually)
price→ (-Mail) ⚡
...text2...
```

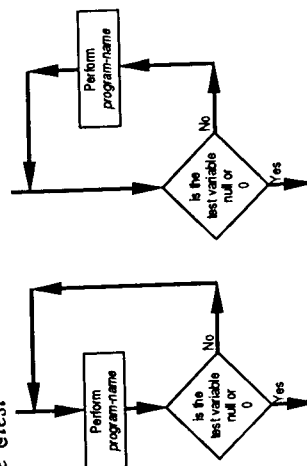
How the instructions work

When LocoMail processes a `&program-name@test` instruction, it starts by working through the commands stored in the program unit as if they had been inserted into the master document after the next `(-Mail)` code, or ⚡, whichever of these it finds first. (You should recognise this as being identical to what happens when you use a simple Perform instruction on its own.) When it has read to the end of the program unit, LocoMail then checks the current value of the test variable. If this is null (ie. no characters at all) or zero, then LocoMail goes straight on to read whatever follows the loop instruction in the master document. If the test variable isn't zero or null, it then works through the commands of the program unit again.

The next command or the repeat of the program unit follows straight on from the end of the previous program unit. The `(-Mail)` code or command separator which follows the Perform instruction has been processed already and is not processed again here. Any command separators or Mail codes that are needed to ensure that LocoMail interprets your master document correctly have to be put into the program unit itself.

When LocoMail processes a `@test&program-name` instruction, it first checks the current value of the test variable. If this is null (ie. no characters at all) or zero, then LocoMail goes straight on to process whatever follows the loop instruction in the master document. If the test variable isn't zero or null, it then works through the commands of the program unit as if they had been inserted into the master document after the next `(-Mail)` code, or ⚡, whichever of these it finds first. When it has read to the end of the program unit, LocoMail then looks at the new value of the test variable, and if appropriate, carries out the instructions in the program unit again as if these followed straight on from the last part of the program unit. Again, any command separators or Mail codes are needed between the end of one 'copy' of the program unit and the start of the next have to be in the program unit itself.

`&program-name @test` `@test &program-name`



Ensuring that LocoMail is active when you want it to interpret instructions and inactive in the text sections of your master document is crucial.

Working through this sequence of instructions and text, you can see that LocoMail is indeed 'active' when you want it to carry out LocoMail instructions and 'inactive' when you want it simply to insert text. In this case, the loop instruction and the program unit have worked perfectly.

The Loop instruction `& item line @ test` that we have worked through above is immediately followed by a `(-Mail)` code. Just as in the case of Perform instructions, this is the easiest case to handle and to understand. When you follow the Loop instruction by one of the command separators : or `←`, LocoMail is already active when it starts to process the program unit. Again as with Perform instructions, you then need to think through very carefully the various changes LocoMail makes between being active and being inactive. We look at this in detail in Section 9.5.

Once again, we suggest that – initially, at least – you always separate your Loop instructions from any following LocoMail instruction by `(-Mail) (+Mail)`. Your loops should then work in the way that you expect. If they don't, turn to Section 9.5.1: this section explains both the types of problem you may run into and what you do to solve these problems.

9.4 Controlling the number of repeats

With either type of loop instruction, LocoMail decides whether to carry out the instructions in the program unit again, depending on the current value of the variable you nominate as the test. If this variable is null (no characters) or zero, LocoMail doesn't repeat the program unit: if the variable is not null or zero, LocoMail does repeat the program unit.

Choosing the right test to control how often the program unit is repeated is very important. You need to arrange that the value of your test variable becomes zero (or null) only when you have repeated the program unit the correct number of times. It is very easy to set up a test that *seems* correct, yet when you come to use your master document turns out to give you either one too few or one too many loops through the program unit!

This section describes how you choose an appropriate test for your program unit and how you use it to control the number of times the program unit is repeated. There are two situations you need to consider:

- when you know how many times the program unit will need to be performed when you set up the master (for example, you always want to insert five items in a list in every version of the master)
- when you don't know how often you will want to repeat the program unit (for example, you want to include different numbers of items in different versions of the master)

In the next two sections, we will see how to set up suitable tests for both situations. Setting up appropriate tests for your own program units takes careful thought as it is very easy to produce a test that is almost, but not quite, right. As

always the best advice is to set up a special master document to test both the program unit and the way it is repeated, before using it in a 'real' master document. Then you can check that the test works and that you get the results you expect.

Note: The test variable that you use in the loop must be set to a value before its value is checked. If you use the instruction `@test#program-name`, you must set the value of `test` (using a `test=value` type of instruction) somewhere earlier in the master. If you use the `#program-name@test` instruction, you can set the value of `test` within the program unit – since the instructions are performed before `test` is checked.

9.4.1 Counting the number of repeats

If you know in advance how many times you want to repeat a program unit, arranging that the test variable becomes zero after the right number of repeats is very simple.

You initially set the value of the test variable (eg. `counter`) to the number of repeats you want to make and include an instruction in the program unit that subtracts 1 from this test variable. Then each time the program unit is performed, the value of `counter` is reduced by 1. By the time the program unit has been carried out the right number of times, the value of `counter` will be zero and LocoMail will stop repeating the program unit.

This technique gives the right number of loops regardless of whether you are using a `@test#program-name` or a `#program-name@test` loop instruction – except when you set `counter` to zero. (If you set `counter` to zero, the `@` loop isn't performed at all but the `#` loop will continue with `counter` getting more and more negative. This is something to watch out for when planning your Loop instructions.)

For example, suppose you want to repeat five times the simple program unit `memb_name` described in Section 9.1, using a `@test#program-name` instruction with `counter` as the test. The fundamental program unit is:

```
(+Mail)memb_name=" (+Mail) ←
?;member's name? → (-Mail) → (+Mail) ←
?;phone no.? (-Mail) ←
" → (-Mail)
```

To repeat this program unit five times, you need to set the initial value of `counter` to 5, and include an instruction within the program unit to subtract 1 from `counter`. The program unit therefore becomes:

```
(+Mail)memb_name=" (+Mail) ←
?;member's name? → (-Mail) → (+Mail) ←
?;phone no.? ←
counter=[counter-1] ←
(-Mail) ←
" → (-Mail)
```

and the instructions that make sure it is repeated five times are:

```
(+Mail) counter=5 ←
@counter@memb_name (-Mail)
```

When LocoMail comes to these instructions, it will set the value of **counter** to 5. Then it will perform the instructions in **memb_name**, making **counter** now 4. The test variable is not yet zero (or null), so LocoMail then repeats the program unit – this time making the value of **counter** 3. The fifth time through **memb_name**, **counter** is reduced from 1 to zero and so when LocoMail next checks its value, it will stop repeating the program unit and continue to the next stage of the master.

The result would have been exactly the same if instead of using an **@test@program-name** instruction, you had used an **@program-name@test** instruction.

• The **memb_name** program unit can be seen in action in a Loop instruction in the **LOOP.EG1**, **LOOP.EG2** and **LOOP.EG3** master documents in the **ADVANCED** group on the LocoMail Examples disc.

LOOP.EG1 shows the program unit being repeated exactly five times by the Loop instruction **@counter@memb_name**.

LOOP.EG2 is like **LOOP.EG1** but instead of the number of repeats being fixed for you, the master starts by asking you how many repeats you want and then makes this the initial value of **counter**. Notice how the program unit isn't performed at all if you say you want 0 repeats.

LOOP.EG3 is like **LOOP.EG2** except that it uses the other kind of Loop instruction, i.e. **@memb_name@counter**. This time if you say you want 0 repeats, **memb_name** is still performed once – in fact, you'll have to press **[STOP]** and then **[STOP]** again to stop it repeating because the loop continues to repeat as **counter** gets more and more negative!

9.4.2 When you don't know how many repeats

When you don't know how many times you will need to repeat a program unit (i.e. because the number of repeats varies from pass to pass), you have two options:

- 1 You can have as your test some item of information that is automatically zero or null on the final repeat. In many cases, this is achieved by typing 0 or just pressing **[ENTER]** (which enters a null) in response to this prompt when the program unit has been repeated enough times.
- 2 If you know that on the final loop some item of information will always have a particular value or that two values will be the same, then you can have a conditional instruction in the program unit that tests for this and sets your test variable to zero or null only in the final loop. (You might use this technique to set a test variable to zero or null, depending on whether you answer, say, yes or no to a question.)

We look at both these options in detail below.

Waiting for a null or a zero

1 – Typing a null or a zero

You can always control how often a program unit is performed by making the test variable something that you type in response to ? instruction within the program unit and arranging that you type 0 or press **[ENTER]** only on the final loop.

If there is some item of information which the program unit asks for which is set to zero or null on the final loop, then this is obviously the item to use as the test variable. If not, you can always put the instruction:

```
(+Mail) test=?; prompt(-Mail)
```

into your program unit. This simply stores whatever you type in response to the prompt as the current value of your test variable (represented here by **test**). All you need to ensure is that you only type 0 or press **[ENTER]** to enter a null in the final loop.

For example, suppose you wanted to repeat the **item_line** program unit described in Section 9.1:

```
(+Mail) item_line="(+Mail) ←
? number→ ; No. of items→ (-Mail) → (+Mail) ←
? → ; description→ (-Mail) → (+Mail) ←
? item→ ; item price→ (-Mail) → (+Mail) ←
price=[number*item[2]] ←
price→ → (-Mail) ←
" (-Mail)
```

When this program unit is performed, you are prompted to type in the details for one line on an invoice – first the number of items, then the description, then the unit cost. You are unlikely to want any of these items to be zero or null in the final loop, so here the best option is to add an extra instruction which will ask you, say, to enter Y if you want to repeat the program unit again but to just press **[ENTER]** if you don't. This would make your program unit:

```
(+Mail) item_line="(+Mail) ←
? number→ ; No. of items→ (-Mail) → (+Mail) ←
? → ; description→ (-Mail) → (+Mail) ←
? item→ ; item price→ (-Mail) → (+Mail) ←
price=[number*item[2]] ←
price→ → (-Mail) ←
(+Mail) test=?; To repeat type Y, to finish press [ENTER] ←
(-Mail) " ←
(-Mail)
```

You could then repeat this program unit as many times as you liked by using the instruction:

```
(+Mail)@item_line @test (-Mail)
```

Then when you fill your master, you would type the details of a line of the invoice and then either type `Y[ENTER]` or just press `[ENTER]`, depending on whether you had another line to add to the invoice or you had finished entering this information. When you just press `[ENTER]`, `test` automatically becomes null and the program unit isn't repeated again.

Note: This isn't a particularly well designed example because typing `N` (as you may well be tempted to do when you don't want the loop to repeat any more) won't stop the loop. It has only been written like this to illustrate how the technique is used. A better way of handling Yes/No type answers is to store your answer under an item-name and then use the value of this item-name to control whether `test` is set to null or zero. We show how to do this below in the section on using conditional instructions to control loops.

- *This method of controlling repetition of a program unit is used in the REPEAT-NUL master document, in ADVANCED group on your LocoMail Example Documents disc. Use LocoMail's Fill mode to see it in operation.*

2 - Using a null or zero in the data file

You may also be able to use zeros or nulls with the datafile that you are using to control your loop. In particular, you can use this to make up a list by taking some information from each record in the data file - thanks to the way LocoMail sets all the items in a record to null if it attempts to read beyond the end of a data file. For example, suppose you wanted a complete list of names and telephone numbers. The LocoMail instructions to insert this information and then move on to the next record might be:

```
(+Mail)name (-Mail) → (+Mail)phone ←
$+ (-Mail) ←
```

You could use this sequence to print out a list of all names and telephone numbers in the data file by grouping them into a program unit (called, say, `list`), then repeating `list` until one of the items in the data file - for example, the `name` item - becomes null, as follows:

```
(+Mail)list=" (+Mail) ←
name (-Mail) → (+Mail)phone ←
$+ ←
(-Mail) " (-Mail)
```

```
...
(+Mail) %list@name (-Mail)
```

LocoMail will keep repeating `list` until `name` becomes null - which (assuming you have no records with a blank `name`) will give you a complete list of all the names and telephone numbers in the data file.

Using a conditional instruction to control the loop

To use a conditional instruction to control a loop, you need to arrange that one of your items of information has a particular value **only** on the final loop. You can then include a conditional instruction in your program unit that tests to see whether this item has the specified value and if it does, set the test variable to zero or null. While you may be able to do this, in terms of one of the items in your data file, you are more likely to set up your program unit so that it asks you to type **Yes** or **No** in response to a question like **More entries?**

Both here and in the instruction we used in the previous sections, the aim has been to ensure that the item we are using as the test variable becomes zero or null when the program unit has been repeated the correct number of times. But whereas in the earlier section we made the test variable null or zero directly, here we are first setting the value of some other variable and then using a conditional instruction to set the test variable. In the examples that follow, the test variable we use is `finish` and in each case you will see that its value is set to zero by a conditional instruction.

Suppose, for example, that you wished to type **Yes** or **No** to control the program unit `item_line`:

```
(+Mail) item_line=" (+Mail) ←
? number → ; No. of items → (-Mail) → (+Mail) ←
? → ; description → (-Mail) → (+Mail) ←
? item → ; item price → (-Mail) → (+Mail) ←
price=[number*item|2] ←
price → (-Mail) ←
" (-Mail)
```

If you were to repeat this program unit using `finish` as your test variable, you would need to add a couple of instructions like these to your program unit:

```
reply=?; More entries? (Type Yes or No) ←
#reply="N*":<:finish=0:>
```

However, simply adding these instructions would give you a string (the "N*" string) within another string (the main program unit string). As we explain in Chapter 10, this introduces complications and should be avoided if at all possible. Here we can avoid the problem completely by storing the "N*" string under an item-name by using the instruction `(+Mail) noword = "N*" (-Mail)`

This makes the program unit we need:

```
(+Mail) item_line=" (+Mail) ←
? number → ; No. of items → (-Mail) → (+Mail) ←
? → ; description → (-Mail) → (+Mail) ←
? item → ; item price → (-Mail) → (+Mail) ←
price=[number*item|2] ←
price → (-Mail) ←
(+Mail) reply=?; More entries? (Type Yes or No) ←
#reply=noword :<:finish=0:> ←
(-Mail) " (-Mail)
```

and we can use it as follows:

```
(+Mail) finish=1 ←
$item_line@finish (-Mail)
```

When you fill your master, you would type the details of a line of the invoice and then either type Yes or No, depending on whether you had another line to add to the invoice or you had finished entering this information. When you type No (or just N if you want), this sets `reply` to No and the conditional instruction then sets `finish` (the test variable) to 0. This stops the program unit being repeated again. (Note: The conditional instruction just tests for something starting with an N, so typing anything else or even just pressing `[ENTER]` counts as typing Yes. Studying the `PROGUNIT` IMP Building block will show you a better way of handling Yes and No answers.)

This method of controlling the repetition of a program unit is used in the `REPEAT.CND` master document in `ADVANCED` group on your LocoMail Example Documents disc. Use LocoMail's Fill mode to see it in operation.

The above method doesn't have to be used with information that is typed in: it can equally well be used to stop a program unit according to information stored in a data file. For example, suppose you want to take details from a sequence of sales records made on a particular day (eg. when the `date` item in the record holds 27/9/88). When you come to the record where the `date` holds a different date (say, 28/9/88), you want to stop. So what you need to do is make sure that the test variable becomes zero only when `date` stops being 27/9/88.

date	name	address; postcode	product; amount
27/9/88	Mr J C Smith	585 New St	
	Antisham	Cheshire; CS4 8CT	
	Fuchsia	£3.45	
27/9/88	Mr F W Tombs	The Gables	
	Elvington Rd	Draftwelle	
	Edinburgh	Roses	£2.95
28/9/88	Dr H Quentine	45 Green St	
	New Bondwelle	Glos; TMS 3EW	
	Blackcurrant	£6.95	

To produce a list of all the records made out for a given date, you need to:

- check if the date is the date in the record: if it is, insert the name and amount
- move on to the next record
- check if the date in the new record is 27/9/88; if it isn't, stop repeating the program unit

Each of these steps can be tackled with LocoMail instructions. The conditional instruction that will do the first step is:

```
(+Mail) #date=given_date:<:name:amount:> (-Mail)
```

The instruction that will do the second step is simply `(+Mail) $(+(-Mail))`. To do the third step, we need to use another conditional instruction which says that if the `date` is not the given date, a variable (`finish`) is set to zero:

```
(+Mail) #date<>given_date:<:finish=0:> (-Mail)
```

These instructions could be gathered into a program unit (called `loop`) and `loop` repeated using the `finish` variable as the test, as follows:

```
(+Mail) loop=" (+Mail) ←
→ #date=given_date → ;test date in current record ←
→ → <:name:amount:> ←
→ $+ → ; advance to next record ←
→ #date<>given_date → ;test date in next record ←
→ → <:finish=0:> ←
→ → (-Mail) " ←
...
(+Mail) finish=1 ←
$loop@finish (-Mail)
```

This will give you the right result: a list of all names and amounts of sales made on a particular day, stopping when the first record for the next day is encountered.

9.5 More complicated cases

The Perform instruction and Loop instructions we have worked through above have all been immediately followed by a `(-Mail)` code because this is always the easiest case to handle and to understand. Indeed, if you find what we describe in this section at all difficult to understand, we recommend that you arrange that all your Perform and Loop instructions are followed by `(-Mail)` codes. If you want another instruction to follow on from these instructions, then separate the two instructions by `(-Mail) (+Mail)`.

What we describe here is what happens when you follow your Perform or Loop instruction by one of the command separators: or `←`. This is somewhat more complicated because LocoMail is already active when it starts to process the program unit. In the cases we have worked through above, LocoMail was inactive at the start.

But before you think you needn't worry about this - a warning: if you improve the layout of your master document, for example by putting a carriage return at the end of your Perform instruction as follows:

```
...text/... (+Mail) & item_line ←
(-Mail) ...text...
```

this seems exactly like the `'...text/...(+Mail) & item-line (-Mail) ...text2...'` we used successfully earlier, but it won't necessarily work the same way because of the `←`. To understand what will happen when this instruction is processed, you need to read on.

For example, suppose you have:

```
...text/...(+Mail) & item_line : LocoMail-instruction (-Mail)
...text2...
```

In this case, LocoMail will process the program unit as if it were inserted in the document after the colon – ie. as if you had written:

```
...text/...(+Mail) & item_line : (+Mail) ←
? number → ; No. of items? → (-Mail) → (+Mail) ←
? → ; description → (-Mail) → (+Mail) ←
? item → ; item price → (-Mail) → (+Mail) ←
price=[number*item|2] ←
price → (-Mail) ←
LocoMail-instruction (-Mail) ...text2...
```

LocoMail is already active as it starts to process the program unit and so the `(+Mail)` code at the start of the program unit is ignored. All is well until you get to the end of the program unit and go on to the *LocoMail-instruction* following the Perform instruction. The `(-Mail)` code at the end of the program unit makes LocoMail inactive at this point and so the following instruction is treated simply as text. This is probably not what you want!

The case with a carriage return in place of the colon works in exactly the same way. Moreover, exactly the same happens if you put:

```
...text/...(+Mail) & item_line ←
(-Mail) ...text2...
```

In this case, too, LocoMail processes the program unit as if it were inserted after the `←`, not as if it were inserted after the `(-Mail)` code. LocoMail will therefore be active when it starts to process your program unit. Hence our warning at the start of this section about program units not necessarily working the same way after you improve the layout of your master document!

The next section describes how to cure these problems but first we ought to look at what happens when you have text at the start of your Program unit – for example, if we were using a program unit (called `tab_line`, for example) which is identical to the `item_line` program unit except for starting with a `tab` you wanted inserted into the document:

```
(+Mail)tab_line=" → (+Mail) ←
? number → ; No. of items? → (-Mail) → (+Mail) ←
? → ; description → (-Mail) → (+Mail) ←
? item → ; item price → (-Mail) → (+Mail) ←
price=[number*item|2] ←
price → (-Mail) ←
"(-Mail)
```

If the Perform instruction `*tab_line` is followed immediately by a `(-Mail)` code, it works in very much the same way as before. However, the case where it is followed by another LocoMail instruction won't give the required result – not just at the end of the program unit this time but at the beginning as well. Because LocoMail is already active as it starts to process the program unit, it will simply treat the `tab` as white space separating out a LocoMail instruction and throw it away! Again, this is probably not what you want.

9.5.1 Curing the problems

When a Perform or a Loop instruction doesn't work as you had planned, the problem is almost always whether LocoMail has the right status ('active' or 'inactive'):

- when the program unit has been inserted after the `(-Mail)` code, the `:` or `←` that follows the loop instruction,
- when one copy of the program unit follows on from the next, and
- when LocoMail moves on from the end of the program unit to the part of the master document that immediately follows.

If LocoMail is inactive (simply inserting text into the document) when it should be active (ie. treating what it reads as commands), your commands will inserted straight into the document. If LocoMail is active when it should be inactive, spaces, tabs and carriage returns will be swallowed but any other characters LocoMail reads it will try to interpret. The result is often a Command Syntax Error.

The first thing to check is that you have equal numbers of `(+Mail)` and `(-Mail)` codes and that the end of the program unit finishes correctly. One very easy mistake to make is to forget to put a command separator or a `(-Mail)` code at the very end of your program unit, with the result that LocoMail runs straight on from the last part of the program unit into the first part of the program unit the second time round, or into the text or command immediately following the Perform instruction. This often shows up as a Command Syntax Error.

If the Perform instruction fails despite the program unit being properly terminated, you need to change either the Mail codes in the program unit or the command separator following the Perform instruction to get LocoMail to be active and inactive at the right points:

• Changing the Mail codes in the program unit itself

There are fixed rules about how this can be done because you must maintain an equal number of `(+Mail)` and `(-Mail)` codes in the string setting up the program unit. In practice this means you can:

- take away or add a pair of Mail codes around the program unit itself. For instance, if your program unit starts with a `(+Mail)` and finishes with a `(-Mail)` you may be able to correct your problems by removing these codes from the program unit.

- add an extra (+Mail) code at the beginning of the program unit and an extra (-Mail) code at the end (or vice versa).
- move a Mail code that is currently at the very beginning of the program unit to the end (again, or vice versa)

• Using a different command separator after the Perform instruction

The other option is to replace the colon (or the carriage return) following the Perform instruction by (-Mail) (+Mail). Then the program unit is processed as if it followed the (-Mail) code.

Changing the Mail codes needs careful thought and it can go wrong in ways that are hard to understand until you are a very experienced user of LocoMail. In the case of the **item_line** program unit, you can in fact get rid of the (+Mail) code at the start of the program unit because it isn't needed. However, because the program unit finishes with some text (a carriage return that you want in the document), you need to add an extra (+Mail) code at the end so that LocoMail is active at the start of the following instruction. (Note: Getting rid of the (+Mail) code from the start of the program unit itself meant that you either had to add an extra (+Mail) or take out a (-Mail) to restore equal number of (+Mail)s and (-Mail)s in your program unit.)

However, this won't give the right results in every case. In particular, you can get into trouble if your program unit contains any strings, for example in conditional instructions. (The problem is how you tell LocoMail that the double-quote starting the string within the program unit isn't a double-quote marking the end of the program unit. See Chapter 10 for details.)

If you can't readily spot a way of adjusting the Mail codes, then change the command separator to (-Mail) (+Mail). How LocoMail then interprets your program unit is very much easier to follow. In the case of some Loop instructions, changing the command separator to (-Mail) (+Mail) may indeed be the only way to get the result you require.

Note: If you need a command separator between the end of one loop through program unit and the start of the next, you don't have to put this at the end of the loop. In certain circumstances, you may prefer to have this separator at the start of the loop.

However, we recommend putting the separator at the end of the program unit because there are a couple of side-effects of putting the command separator at the start of a program unit. For a start, you will probably need two separators between your Loop instruction and the instruction that follows it. The other problem is that your loop will be carried out one more time than you expect if the last part of the program unit is the instruction which sets the new value of the test variable - because this instruction isn't carried out until after the separator has been processed.

9.6 An alternative approach

If all you want to repeat is one or two very straightforward LocoMail instructions, you can opt to put your instructions straight into a LocoMail instruction that repeats them, rather than set up a program unit.

The instructions are:

```
(+Mail) & "instruction-sequence" @test (-Mail)
(+Mail) @test & "instruction-sequence" (-Mail)
```

and

The first instruction executes the instruction sequence and then sees whether *test* is null or zero before repeating the instruction sequence. The second instruction does the test first.

How are these instructions used? Suppose the instruction you wanted to repeat was simply:

```
(+Mail) ? person (-Mail)
```

To repeat this instruction until the given test is zero or null, you would use one of the following LocoMail instructions:

```
(+Mail) & "(+Mail)? person (-Mail)" @test (-Mail)
(+Mail) @test & "(+Mail)? person (-Mail)" (-Mail)
```

Similarly, if you just wanted to repeat the combined instruction:

```
(+Mail) number+ :? person (-Mail)
```

you would use:

```
(+Mail) & "(+Mail) number+ :? person (-Mail)" @test (-Mail)
(+Mail) @test & "(+Mail) number+ :? person (-Mail)" (-Mail)
```

The similarity between these instructions and the ones you use to repeat program units is clear. All you are doing here is saving yourself the trouble of formally setting up a program unit. Once again, the set of instructions are initially processed as if they followed the next (-Mail) code, : or < (whichever LocoMail finds first); if the instruction sequence is repeated, then the second time through the instructions follows immediately on from the first; etc. etc.

As you might expect, you have to be just as careful about ensuring that LocoMail is active at the point you want to it carry out LocoMail instructions and inactive where you want it to insert text as you are when designing your program units. You also need to ensure that you can set up a suitable *test* and that the item being tested or its components are set before any test is made (see Section 9.4.)

All the rules and advice that we have given earlier in this chapter in connection with program units applies here as well.

Two or three instructions is the most you would really want to put directly into a loop instruction. The Loop instruction becomes a little more complex - and untidier - particularly, where the instruction you want to repeat includes a prompt. Indeed, if your instructions are at all complex, then it is very much better to use a program unit!

Postscript

If you have found this chapter hard going or you find that the program units you create don't work, don't panic. The trick, often used by programmers, is to start from something that you know works and adapt it to suit your needs. So if you have problems creating a program unit, we recommend you to pick out a suitable program unit from among the ones in the example documents and building blocks on your LocoMail examples documents disc and simply change it until it does the job you require.

This technique isn't entirely foolproof, however – you do have to be very careful to keep the same structure of Mail codes and command separators and, if you need to remove or add any Mail codes, you must ensure that you keep the numbers of (+Mail) and (-Mail) codes in each program unit equal.

As you probably realised, the bits of the instructions that matter are the beginnings and the ends. Concentrating on these parts of the instructions and following the advice in Section 9.5 on how to sort out what has gone wrong when you see a Command Syntax Error should help you get your program units, Perform instructions and Loop instructions working correctly.

Chapter 10

Nesting instructions

Up until now, we have described instructions as if they always appear 'one after another' in the master document. However, a particular feature of these advanced instructions that makes them much more versatile, is that you can embed one instruction inside another – ie. you can get LocoMail to carry out an instruction in the middle of doing another. Putting one instruction inside another like this is what programmers might describe as 'nesting' instructions, and it can be represented by:

```
(+Mail) start-of-instruction1 instruction2 rest-of-instruction1 (-Mail)
```

└── nested instruction ─┘
└── main instruction ─┘

(Note: In this representation, *instruction2* might itself contain further 'nested' instructions, and so on.)

As LocoMail works through an individual instruction, it checks the syntax of what it reads and makes sure that the instruction is complete and correct. When LocoMail works through nested instructions, the process is no different – every nested instruction it meets must be complete and correct, and once LocoMail has processed any inner nested instruction(s) the outer instruction too must be correct. If any of the nested instructions are incomplete or otherwise incorrect, then LocoMail will give a syntax error.

Making sure that your nested instructions are correct usually just requires a little care over setting them up. For example, in master documents you might have been using a conditional instruction (as described in Chapter 6):

```
(+Mail) #condition:<action>(-Mail)
```

In a more complicated application, you might also want the *action* of this conditional instruction to include another conditional instruction:

```
...#cond1:<:start-of-action1:#cond2:<:action2:>:rest-of-action1:>.
```

└── nested instruction ─┘
└── main instruction ─┘

To make sure that LocoMail processes the sequence correctly, you simply have to ensure that suitable command separators are used and that the *action* that goes with each *condition* is enclosed between a pair of angle brackets (< and >) – in other words, that the embedded conditional instruction is complete, and that the outer conditional instruction is also complete.

Other nested instructions may not be quite so easy to check.

Good layout can often help you to ensure that each instruction is properly constructed. For example, the structure of nested conditional instructions can be made very clear if they are laid out like this:

```
(+Mail) ⚡
#condition-1 ⚡
→ → <: action-1 ⚡
→ → #condition-2 ⚡
→ → → <: action-2 ⚡
→ → → > ⚡
→ → > ⚡
(-Mail)
```

Here, you can see the pairs of angle brackets for each *action* have been vertically aligned by tabs, making it very easy to see if one has been missed out without necessarily working through the various *conditions* or *actions*. For example, if the instruction you actually have in your master is:

```
(+Mail) ⚡
#condition-1 ⚡
→ → <: action-1 ⚡
→ → #condition-2 ⚡
→ → → <: action-2 ⚡
→ → → > ⚡
(-Mail)
```

it is very easy to spot that you have forgotten the > to match the < starting the second *action*.

Strings within strings

As well as wanting to embed instructions within instructions, you may also want to embed one string of characters within another (of course, a string can contain text, codes, instructions or whatever you want). One place where this may happen is when you are telling LocoMail about a group of instructions that you want to use repetitively (as you saw in the previous chapter).

A string of characters and codes begins and ends with a double quote mark ("). So whenever LocoMail comes across a double quote mark, it concentrates on looking for another double quote mark that finishes the string – in much the same way as with conditional instructions, when it looks for a > to match each < it processes.

If you want to embed a string within this current string, you have to tell LocoMail the difference between the double quote marks that start and finish an embedded string, and the double quote mark that finishes the current string.

The way to do this is to put a pair of Mail codes around the embedded string as follows:

```
(+Mail) item-name="... (+Mail) " ... " (-Mail) ... " (-Mail)
```

inner string
outer string

The (+Mail) before the inner string can be thought of as shifting the inner string 'down a level', while the (-Mail) after it moves it back 'up a level'. LocoMail only matches double quote marks that are at the same level, and so it will successfully pick out both the inner string and the outer string.

Without these extra Mail codes, you would have:

```
(+Mail) item-name="... " ... " (-Mail)
```

Here, LocoMail would assume that the string to be stored under *item-name* ends at the second double quote mark, and will then report a syntax error as it tries to interpret the rest of the characters as an instruction.

IMPORTANT: If you use Mail codes to signal embedded strings, you must be very careful that the overall effect of the codes doesn't also change LocoMail's status from active to inactive (or vice versa) when the string is used. Any unintentional change of status might make your LocoMail program fail with syntax errors.

For this reason, we recommend that if at all possible, you write your LocoMail program so that you don't need to nest strings. It is usually possible to store the strings you will need to embed under *item-names* (by using the (+Mail) *item-name*="..." (-Mail) instruction), then replace the embedded string by this *item-name*. Rather than detract from the master document, this approach usually makes the master document much easier to read – especially if you choose an *item-name* that clearly indicates the meaning of the string it represents.

Designing master documents

While you are using LocoMail for tasks like simple mailshot letters created from a LocoScript document, or form filling, inserting LocoMail instructions won't stop your master documents being easy to follow. In particular, when you come back to use or adapt the same master some time later, you will be able to see what information it requires and how LocoMail uses this information to produce the filled document.

However, as you start to use LocoMail for more ambitious tasks (eg. an invoicing system, or handling the records for a club or society), your masters will include many more instructions – and these instructions might be highly complex calculations, program units or any of the other advanced features described earlier in this part of the User Guide. When you first create and use these masters, everything will be fresh in your mind and you'll probably have no difficulty in working out the structure of the document. But when you come back to look at the master some time later, you will probably wonder how you ever made sense of all those instructions!

To make your masters more understandable (and consequently easier to use, upgrade or adapt at a later date), you need to think carefully about their design before you create them. This chapter is a 'discussion' of some of the things you should consider as you create complicated masters: as is so often the case, if you put in a little more effort at the beginning in producing a good design, you will reap the benefits later on.

The main places to put this effort are in:

- analysing the task to make sure that the master will do the job
- structuring and laying out the master for ease of use
- making sure that the master works properly

The remainder of this chapter will describe each of these areas in turn.

11.1 Analysing the task

The first thing to think about when designing a master document is what, precisely, you want it to do. When you're sure you know what you want to achieve, you can see what actions you want LocoMail to carry out. Then you can work out how you are going to carry out each step.

For example, you might divide the task of producing a very simple letter into four simple steps:

- print the letter heading
- get and print the addressee's name and address
- get and print the date
- print the standard letter text

Producing a receipt is a little more complex, and might require a few more steps:

- print the receipt heading
- get and print the date and client name
- get the product name, cost and other details
- print the product name and price, accumulating the total price and VAT
- repeat the previous two steps for each new product, until the user indicates there are no more
- print the calculated totals

When you've split the task up into smaller steps, decide how you are going to carry out each step. You'll probably find that some of the steps only use straightforward text, while others will require LocoMail instructions to get or manipulate information, conditional instructions to select different sections of text, etc.

The steps which essentially only involve LocoScript features, such as fixed text which will always be inserted in every version prepared, should not cause you any problems. You simply need to type it in (complete with word-processing codes, etc.). For example, when creating a standard business letter, you can start by typing the fixed details like your company's name and address. This text tends to provide a sort of framework to the document.

Next you need to work out the LocoMail instructions necessary to perform each of the relevant steps. You might find some useful sequences of instructions in the example 'Building Blocks' included in Appendix III of this User Guide, or in the programs in Part IV, or in the example documents on your LocoMail example documents disc. Simply adapt the instructions given there to suit the master you're creating.

If you find it hard to translate a step into LocoMail commands, keep breaking it down into smaller steps and try to solve those. If you still have problems, is it because you do not understand how to perform the step, or because you don't know the right LocoMail instructions? In the former case, think carefully about how the task should be performed, look it up in a standard text or ask an expert. In the latter case, check that you aren't trying to do something that LocoMail is not capable of - such as stepping back through records in a data file. You might have to think of another way of carrying out that step.

Design each LocoMail step in the document separately, ideally as a separate master document in its own right. This allows you to concentrate on solving a single problem (or group of related problems) at a time. It will also make it easier and quicker for you to write and check the step.

Decide on the instructions to use - each type of instruction is covered elsewhere in this User Guide. What you should think about is the best way of acquiring and using the information, and then write the instructions accordingly. (You may also find, after checking through the completed master a number of times, that you can see ways of improving the instructions. This 'fine tuning' is an important part of the design process.)

When you have got the separate steps right, the next stage in the document design is to put these steps into a well laid-out master.

11.2 Setting out the master document

The aim of the guidelines given in this section is to make your master document both easy to understand and easy to use. This is particularly important if your master document is going to be used by other people, or if you (or anyone else) may want to change this master (for instance, to allow for a change in VAT rate, or to change the conditions for including different text in letters).

General recommendations

- **Structure and lay out the master so that you can understand how it works when you come back to it!** Use meaningful names for your items, and add comments at important points. Use spaces, tabs and new lines to make instructions (and particularly conditional choices) clear, bearing in mind that:
 - spaces and tabs within LocoMail instructions are ignored (unless they are within double quotes or as part of a prompt or comment)
 - carriage return characters `↵` may be used to separate LocoMail instructions. They may also be inserted immediately after `(+Mail)` or immediately before a `(-Mail)` without changing the effect of the instruction
 - colons may also be used to separate instructions, except where the instructions include prompts or comments, when a `↵` or `(-Mail) (+Mail)` must be used
- **Use item-names to store 'fixed' values that you use throughout the master** - such as your company's name, the VAT rate, conversion factors, etc; don't use the value itself. This makes the document more understandable and easier to update. Store the appropriate value under the item-name at the start of the master. This means that if the value isn't as fixed as you thought, you just have to edit that one value: you don't have to go through the whole master changing this value wherever it appears.
- **Use program units to break the master up into separate chunks.** This has a number of advantages, including improving legibility and simplifying the overall master.

To improve the ease of use of your master:

- **Design the document to be as logical as possible.** For example, fetch information, then perform calculations, then generate the finished document. If the document has several distinct parts, sort those out separately. Keep a clear idea of what you're trying to do at any particular stage: this will make your master document easier to design and to use.
- **Help users by displaying prompts that actually explain what information they are meant to type.** You might also display a 'prompt' at the start of a master that explains what is going to happen, ending with something like 'Press [ENTER] to begin'; you can ignore the response, but at least the user can decide to start filling the document when ready.
- **Prompt for information in the same order as users might think of it.** This will help to avoid problems which can arise due to users 'keying ahead'.
- **Try to minimise the typing and waiting required.** For example, accept either Y or Yes; request all the information you'll need at the start, particularly if there is a long wait until the 'end'; also, only ask for information that you need.
- **Check that typed replies make sense (if this is possible) and ask again if they don't.** For example, check the sign of debits and credits and that they are not 0. If you ask for a piece of information, check that it is valid. People do make typing errors, misunderstand prompts and may even type silly answers in frustration!
- **Take the trouble to store information (under an item-name) if this will save typing later.** It costs nothing!
- **Name data files so that they are listed next to the master document on the Disc Manager Screen.** This will make it easier for the user to find the right data file.
- **Turn off codes, etc. in the master document.** This will make the prompts stand out even more.
- **Make sure that prompts don't run to or beyond the edge of the screen.**

If speed of operation is important:

- **Use program units for repeated sequences of commands** – especially in loops.
 - **When reading through a data file, skip records that you do not want to use by using \$+ commands.** If this is not possible, use a * command as early as possible in the master to discard the current version.
- You'll probably have noticed that some of these recommendations can come into conflict. For example, having lots of long prompts and checking of replies will slow the operation down and might irritate users who know what they're doing. It is up to you to decide which 'ideals' are more important to you.*

11.3 Testing your master document

When you've finished laying out your master document, the final stage before you use it in earnest is to check that it actually does what it's meant to. This is, of course, even more important if your complicated master is to be used by anyone else.

Test the master first by running it in Fill mode, to sort out and correct any errors. Only try to use it in Merge mode with your data file when you are certain the master produces exactly the result you require when you type all the information in. Then you can be fairly sure that any problems lie with the structure or content of the data file rather than with the master document.

As well as testing that the master document does what it should:

- **Test the master document by using it as you expect your users to do, or – ideally – by getting someone else to try it out.** This will help you to make sure that it helps rather than hinders. For example, if the master document asks for information in an order quite unlike the way users think, it will cause a lot of mistakes.
- **Check the extremes that you expect the master to handle.** For example, if you expect to have a maximum of ten items per invoice, with a maximum price of £9999.99 each, try these extreme values in the master.

The ideal way to check a new invoice master, for example, is by using it in parallel with a more established system or by using it to handle information that the old system has handled and checking that the results produced are the same. If this is not possible, produce the results some other way (for example, by using a calculator) and compare them.

This sort of thorough checking is only really necessary for masters that do calculations or make decisions, but this will probably be true (to a greater or lesser extent) of most masters you design.

Dealing with problems

Errors which arise in your master document will typically be indicated by an Alert message. The second line of this message describes the nature of the problem. You should be able to tell roughly where the error occurred by watching how far LocoMail gets when working through the document. Occasionally, however, this may not be quite so obvious – for example, when the problem is caused by a group of LocoMail instructions that are read some time before they are executed (eg. a program unit).

Each time you encounter an error, aim to correct just that particular problem. Work logically through the document, sorting out the problems one-by-one and testing after each change you make. In this way you can quickly see whether you are addressing the real cause of the problem or not.

If arithmetic in the document is not giving you the expected results, check that you have specified the calculation correctly – if necessary, use brackets to clarify exactly what you want LocoMail to do (and to clarify it in your own mind!).

Appendix IV describes the Alert messages which are displayed when LocoMail meets an error. Unless you are certain that you know what caused the error, look up this Alert message in the Appendix before attempting to correct any instructions. It also describes the errors which you may encounter where no Alert message is shown – for example, when the wrong information is inserted into the document.

Note: Errors in data files and master documents are also described in Part II.

Part IV: Applications

Introduction

This part of the LocoMail User Guide contains detailed examples of LocoMail in action, performing some of the more complicated tasks it is capable of. All the more complex LocoMail instructions introduced in Part III are used here.

The idea of these examples is to demonstrate how you go about using LocoMail to solve particular problems you have, but in some cases you may be able to use groups of the instructions we give directly in your documents. Often, simple substitution of text or an item-name will be the only change you need.

Each chapter deals with a different group of examples. The areas covered are:

- running a club or society
- maintaining sales records for a business, producing invoices etc.
- producing responses to customer enquiries
- printing address labels on continuous labels stationery (including setting up the document for the new paper type)

The documents described in this part are stored on your LocoMail example documents' disc, in groups 3 – 6.

Note: The example masters are often quite complicated – in many cases, you need a sound knowledge of the principles and instructions described in Parts II and III of this User Guide to understand how the master works. With each master, we give a step-by-step description of its structure and operation: just reading through the master document, noting the approach used and following the description will be good experience.

Adapting these example documents will allow you to develop your own system of LocoMail masters to suit your requirements. But please note that Locomotive Software cannot offer any form of consultancy or other support as to the suitability of these examples to your needs.

Club membership system

The example documents described in this chapter are designed to help you run a busy club or society. The documents are currently in group 3 of your 'LocoMail example documents' disc. The system is based around a data file which contains the details of each member (CLUBFILE); the master documents are:

- MEMBER.LST - prints out the records of all the members, neatly laid out
- SPECIFIC.LST - lists the names and phone numbers of all the members that satisfy certain conditions
- NEWSLETR - produces newsletters, customised to members' interests
- FEESLETR - produces reminder letters for all members whose membership expires in a particular month
- FEESLABL - produces address labels for the reminder letters, and
- UPDATE - updates the data file by copying the existing records and adding new records.

By putting your own details in the data file, and changing the text in the master documents, you should be able to adapt these files to suit your own specific purposes. But first, read through the following sections to see how and why the different techniques described in Parts II and III have been applied. This will help you to put the documents to better use.

12.1 The data file

The data file for the club membership system consists of a number of records, each record containing the details for one member. The information that you store, and the type of information, are determined by the way you want to use the details in the master documents. The item-names we've chosen for the various pieces of information are:

- | | |
|-------------------|--|
| title | - member's title (Mr, Ms, etc.) |
| name | - member's name |
| address | - member's address |
| phone | - telephone number |
| interests | - member's particular interests, stored as a list of single letter codes |
| membership | - type of membership (full, provisional, etc) |
| month | - month of expiry of membership (a number in the range 1 - 12) |
| year | - year of expiry of membership (2-digit number) |
| category | - any other classification you might need |

The separators have been chosen so that the record pattern appears like this:

```
title,name↵
address;phone↵
interests↵
membership,month/year↵
category↵
```

If you are adapting this system to your own needs, you can of course add extra item-names to this record pattern and use the new item-names in the masters. (But remember not to choose item-names which clash with the existing ones.) To see the appearance of the records with 'real' data, look at CLUBFILE.

title	name	address	phone	interest	membership	month/year	category
Mr.	J. C. Seithe	556 New St					
	Wintleshaw						
	Cheshire; 03123-45674						
	A Bo						
	Full, 7/874						
	Group 44						
Mr.	F. W. Tomber	The Cablers					
	Elvinston Rd						
	Bratfield						
	Adelphi						
	Leamouth; 2499-42324						
	Full, 10/884						
	Group 85						

Note: The one condition that your records must obey is that there must be no 'nameless' records. Many of the masters in the membership system are designed to repeat 'program units' of instructions until the **name** item becomes null, which is automatically the case at the end of the data file. If you have any 'nameless' records, the master will stop too soon.

12.2 Printing member's details

Every time you want a complete list of the membership details, you could print out the data file direct from the Disc Manager Screen. But the data file isn't particularly well laid out for easy reading, and anyway it contains abbreviated details (ie. single letter codes representing the member's interests). Instead, you might want the printed records to look like those shown opposite.

The standard headings in this record simply require you to have the correct text, tabs etc. in the master document. The details from the record (the name, the address, etc.) that go beside each heading need to be inserted using LocoMail instructions like **(+Mail) name (-Mail)** and **(+Mail) address (-Mail)**. As these details are printed, we also need to translate the single letter codes stored under **interests** in each record into text (eg. **A** into **Archaeology**, **B** into **Ballooning**, etc.) and insert this text.

NAME: J G Smith
PHONE: 09123-4567
ADDRESS: 589 New St
Wintlesham
Cheshire
INTERESTS: Archaeology Ballooning
MEMBERSHIP: Full
EXPIRY DATE: 7/87
TYPE: Group A

Rather than simply print out the details from a single record, we want to print out all the records in one pass through the master. To do this, the headings and instructions necessary to produce one set of details need to be put into a program unit and repeated.

The master document that produces complete lists of details like this is MEMBER.LST. Merge this document with the CLUBFILE data file to see it in action.

MEMBER.LST can be considered in two parts:

- a group of instructions that convert the single letter interest codes of the current record into the appropriate text
- a program unit that produces the formatted details for the current record, slotting in the interests, and repeating this for every record in the data file

We shall look at each part in turn.

Chair: his master prints out a list of the membership details in better order.

```

; first, two program units are set up;
; the program unit 'pointests' converts interest codes to text;

```

```
#interests="(Mail)"  
#interests="(Mail)Archaeology (Mail)"  
#interests="(Mail)Ballooning (Mail)"  
#interests="(Mail)Canoeing (Mail)"  
#interests="(Mail)Diving (Mail)"  
#interests="(Mail)Electronics(Mail)"
```

```
(-tail)";
; the second program unit 'framework' sets out the details from each data record;
; 'do not break paragraphs' is selected in the Document set-up;
; so that each set of details will be printed on the same page;
```

```

framework = "
NAME: {
  {Mail}name{-Mail}
  {Mail}phone{-Mail}
}

```

```
ADDRESS: +
^
INTERESTS: +
MEMBERSHIP: +
DATE PAID: +
(LAYOU) (+Mail)Address (+Mail) (LAYOU)^
(LAYOU) (+Mail)Interests (+Mail) (LAYOU)^
(+Mail)membership (+Mail)^
(+Mail)month (+Mail)/ (+Mail)year (+Mail)^
```

TYPE: $\rightarrow$ (Mail)category(Mail) $\neq$

不

1. produce the list by repeating framework prog

Frame %Framework(-Hall)4

.....

ProMail User Guide: Part IV – A

Downloaded from <http://ajph.org/> on November 10, 2015

Setting up the
dointerests
program unit to
interpret the
interest codes

Setting up the framework program unit that lays out the details

Performing dointerests

Repeating framework
until the end of the file
is reached

Converting the interest codes

To convert an interest code to a piece of text, MEMBER.LST uses conditional instructions (as described in Chapter 7). As there are five interest codes (A - E), we need five conditional instructions:

```
#interests="A*":<(-Mail)Archaeology (+Mail)><
#interests="B*":<(-Mail)Ballooning (+Mail)><
...
```

The first instruction says "if **interests** in the current record includes the letter **A**, then insert the text **Archaeology** and a space character". Notice that the **A** is between * 'wildcard' characters, so that it will be spotted wherever it appears in the list of codes stored in **interests**. Similarly, the second instruction says "if **interests** in the current record include **B**, then include the text **Ballooning** and a space character", and so on.

Rather than use this cumbersome group of instructions at the point where we want the interest details to be inserted, they are stored as a program unit (called **dointerests**) by the instruction:

```
dointerests="..."
```

At this point in the master, LocoMail just reads through the program unit and remembers the instructions under the name **dointerests**. The instructions in a program unit are only carried out when the program-name appears after a * instruction (see Section 9.2), so the pieces of text aren't actually generated where these instructions first appear. Where we want the instructions to be performed (ie. alongside the **interests** heading), we use the * 'Perform' instruction:

```
*dointerests
```

Laying out the information

The other job of the master document is to set up and use a second program unit (called **framework**) to lay out the details from the current data record and advance to the next record.

The structure of this part of the master is essentially the same as that of the individual records in the finished document, with LocoScript text for headings but simple LocoMail **(+Mail) item-name (-Mail)** instructions where it needs to insert the details from the data file.

The program unit starts by inserting the **NAME** heading on a new line, and a tab. The first LocoMail instruction then appears:

```
(+Mail) name (-Mail)
```

This instruction inserts the name information from the current record. The address line is displayed in the same way. On the 'interests' line, we must perform the instructions we previously grouped under **dointerests**, so we have:

```
INTERESTS→ (+Mail) %dointerests (-Mail)
```

The **%dointerests** command tells LocoMail "carry out the instructions in the program unit **dointerests**". So LocoMail automatically works through the

group of five conditional instructions, generating the appropriate interest text for this record and including it at this point in the finished document.

The rest of our **framework** program unit simply inserts the remaining details from the current record followed by a line of stars, then advances to the next record (using the \$+ instruction).

We repeat this program unit until the end of the data file is found by using the 'Loop' instruction:

```
@name%framework
```

This repeats the program unit **framework** until **name** is null, as it is when the end of the data file has been reached. Of course, the loop would stop before the end of the file if you had any records containing a blank name - hence our warning in Section 12.1 about not having any 'nameless' records. If your data file does contain records with blank names, you should use one of the other items as the test for this loop - something that you know will never be blank.

When this Loop instruction is carried out, LocoMail first tests the value of **name** in the current data record, then starts to execute the **framework** program unit. LocoMail is inactive as it starts to process the program unit, because of the **(-Mail)** code that immediately follows the Loop instruction. So the first part of the program unit (the **NAME** heading) is taken as text - exactly as you require. When **framework** has been performed, LocoMail again tests the value of **name** to decide whether to repeat the program unit but this time, because of the \$+ instruction, it tests the value in the next record. To make LocoMail inactive as it repeats the loop, the program unit finishes with a **(-Mail)** code.

12.3 Finding particular members

You may want to know the names and phone numbers of all the members that meet certain requirements - for example, any members interested in Canoeing, or all the junior members that are interested in Electronics, etc.

As all the membership details are stored in the data file, we can create a master document that will ask for the conditions you're looking for, then sort through all the records in the data file automatically printing out the names and phone numbers of everyone that matches these requirements. The master document that does this is SPECIFIC.LST (shown overleaf).

The best way to understand how SPECIFIC.LST works is to consider how you might tackle the same task using printed membership records:

- first you decide on the requirements you are looking for (which interests, which type of membership and which category of members)
- then you look at the membership records. As soon as you find a member that meets your requirements, you write the headings Name and Phone, and put the details beneath these headings
- then you look at the next record, and, if it meets the requirements, again write down the details (without the headings this time), and you would continue checking records until you reach the end of the data file

Adding these wildcards also means that if you just press **ENTER** in response to the above prompts, you are effectively saying ‘accept any interest’ (or membership, or category).

Producing the list of matches

The required list of all matches is produced by the program unit `loop`.

```
loop=0;(*Mail)*
if findin=interests AND findmem-membership AND findcat-category
(count=count+1); now print headlines only when first match found
count=1;
(*Mail)PHONE=(MIL)*
(*Mail)
(*Mail)
(*Mail)NAME=(Mail)*
(*Mail)*
it's step on to next record
(*Mail)*
(*Mail)*
XloopName=
(*Mail)*
COUNT=COUNT(*Mail)No matching records found in the file. Press ENTER(*Mail)ENTER
(*Mail)
```

The first instruction of the program unit, a conditional instruction, checks whether all the required details match by combining the individual comparisons using AND:

```
#findint=interests AND findmen=membership AND  
findcat=category:<...
```

If all these conditions are met, LocoMail executes the instructions enclosed within the angle brackets of the conditional instruction. These:

- ```

- increment count with the instruction: count=[count+1]
- when the first match is found (ie. when count=1), print the column
 headings. To do this, LocoMail checks the value of count and inserts the
 appropriate text using another conditional instruction:
 #count=1:<(-Mail)NAME→ PHONE←(+Mail)>
- fetch the name and phone number from the data file and insert with a
 LocoScript tab character between them and a carriage return after the phone
 number, to space out the details correctly:

```

name(-Mail) → (+Mail) phone(-Mail) ← (+Mail)

(**Count** enables LocoMail to keep a running total of the number of matches found. This total is used in loop to ensure that column headings are only printed when the first match is found (ie. when **count** is 1), and at the end of the master to display an appropriate message if no matches are found (ie. if **count** is zero after all the records have been checked). **Count** is set to zero at the top of the document by the instruction **count=0**.)

When LocoMail has inserted these details (or skipped the instructions between the angle brackets, because the record didn't match the requirements), it executes the last instruction in the program unit **\$+** to move on to the next record.

| Program unit to search through records                                                                                                                                                                                                      | When first match found, print headings, name and number                                                                                                                                                                                     | Repeat program unit until name is null                                                                                                                                                                                                      |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> 1. Find interest to look for (A-archaeology, B-hallmarking, etc. INTER-ANY) 2. Find membership to look for (I-full, P-provisional, J-junior, ENTER-ANY) 3. Find category to look for (Group A, Group B, Group C, or ENTER-ANY) </pre> | <pre> 1. Find interest to look for (A-archaeology, B-hallmarking, etc. INTER-ANY) 2. Find membership to look for (I-full, P-provisional, J-junior, ENTER-ANY) 3. Find category to look for (Group A, Group B, Group C, or ENTER-ANY) </pre> | <pre> 1. Find interest to look for (A-archaeology, B-hallmarking, etc. INTER-ANY) 2. Find membership to look for (I-full, P-provisional, J-junior, ENTER-ANY) 3. Find category to look for (Group A, Group B, Group C, or ENTER-ANY) </pre> |

The master document approaches the job in just the same way, this time combining the last two tasks into a single program unit. Merge this master with CLUBFILE to see it in action.

## Setting up the requirements

The first step is to specify the particular members you are looking for. These 'search' details must be stored ready to be compared against information the data records; so the kind of instruction we use is:

findint=? : Interest to look for (A-archaeology..

This asks you for the particular interest group you require, as a code letter, then remembers what you type as `findint`. If you wanted to find all the people interested in Ballooning, you would type B; if you wanted all the canoeists, you would type C, etc. (Note this only lets you check a single interest code per pass.) The two other instructions used are similar, prompting for the type of membership and the category.

If LocoMail just checked exactly what you typed against the details stored in the current record, it would reject any records where the two weren't identical in every respect. But clearly you might type **F** to the prompt for the membership type when you want to look for **full** members. So we use 'concatenation' instructions (see Section 8.5) to add 'wildcard' characters to each of your responses to allow some slack in the comparison. (Wildcard characters are described in Section 6.1.2)

```
findint="*" & findint & "*"↵
findmem=findmem & "*"↵
findcat="*" & findcat & "*"↵
```

**The program unit is set up by the instruction:**

```
loop="..."
```

**and repeated by the instructor:**

```
&loop@name
```

This tells LocoMail to repeat `Loop` until `name` is null – which will be the case at the end of the data file (because LocoMail sets all items to null when it reaches the end of the data file). Of course, if any of your records have a blank name, LocoMail will stop carrying out the program unit before the end of the data file!

- You might like to convert a copy of the `SPECIFIC.LST` so that it will search through another data file (one which has different item-names in the record pattern), printing out details when a match is found. This is essentially a case of changing prompts, changing the item-names in the loop program unit and specifying which details are printed.

## 12.4 Customised newsletters

The NEWSLETR master (shown opposite) produces newsletters with the content of the newsletter customised to suit each person's interests. It does this by using the interest codes stored in each record to pick out the sections of text you want to include for each interest. For example, the letter will include details of the next archaeological dig only if the member's interests include Archaeology, details of a diving expedition only if the member is interested in diving, etc.

NEWSLETTER produces one newsletter for each pass through the master. No letter is produced for a member unless you have something specific to tell them – ie. if there is no text for the current member's interests.

The key to how the master document creates these customised newsletters is to think of the task being divided into three stages:

- first decide on what you want to say to each particular interest group  
— then look at a member's interests and work out which paragraphs apply  
— finally, produce the newsletter for that member, including the relevant paragraphs

The NEWSLETR master can conveniently be considered in three parts - each part corresponding to one of these stages.

## Setting up the details

The first part of the master sets out the different paragraphs for each interest. To make these paragraphs easy to manipulate later in the master, each paragraph is stored under an item-name. For instance the paragraph that applies to members interested in Archaeology is stored under **Archaeology** by the instruction:

**!Aclause="There are a number. "↵**

**Part 1 -**  
store the different pieces of text under item-names.  
**DO \$USE=9** there are a number of digs in Curnall this month; contact J.B.S.  
for details if you're interested."  
**DO \$USE=9** Mother Inc. is offering complimentary rides for club members at  
the Dutchess show. Be sure to take your membership card with you  
if you want to take advantage of this offer."  
**DO \$USE=9** The club flyers still have not been returned. A surcharge will  
be levied on all diving members if they are not returned to the  
clubhouse within seven days to cover the cost."  
**DO \$USE=9**

**Part 2 -**  
build up the content of the letter depending on the interests:  
**content=tell your hobby;**  
**interests=48# AND \$CLause)";((content-contentoftheclause);**  
**interests=48# AND \$CLause)";((content-contentoftheclause);**  
**interests=48# AND \$CLause)";((content-contentoftheclause);**  
**interests=48# AND \$CLause)";((content-contentoftheclause);**  
**interests=48# AND \$CLause)";((content-contentoftheclause);**  
**\$content="";(48#)**

**Part 3 - produce the letter.**  
**Mail(\$Align)**  
**Align)**  
**Align)**

**Dear Mail(\$Name(\$Mail),"**  
The following information should be of interest to you:"  
**Mail(\$Content)**  
**Mail)"**  
Thank you for your continued support!  
Yours faithfully,"

Lay out what you want to say to each member

Work out content of letter to match the interests of this member

Framework of letter is into which content is inserted

The exclamation mark at the start of each instruction improves the speed at which LocoMail works through the master because it means LocoMail only has to read and remember the full clauses on the first pass. On subsequent passes, it skips the instruction because it already has something stored under the name **AcLause**.

Any general information that is to be included for all members is stored under **tellevarybody**.

Where there's no news for a particular group of people, the corresponding clause has to be explicitly set to null (i.e. no characters at all): you can't just miss it out. So, for example, NEWSLETR has no news to tell canoeists, and so **Cc**lause is set to null by the instruction:

```
!Cclause="" ↵
```

The last instruction in the first part of the master stores the LocoScript carriage return character as **cr**:

➡ "Yes!"

Storing the carriage return under an item-name like this makes it very much easier to ensure that the carriage return at the end of any optional paragraph is handled correctly. It also makes the master much easier to read – because now, whenever we want to include a ↵ in the finished document, we can simply use the item-name `cr` rather than the cumbersome:

 $(-Mail) \leftarrow (+Mail)$ 

## Building up the text of the letter

The second part of the master builds up the text of the letter for the current member. As LocoMail works out whether to include each paragraph or not, the text is gradually built up under the item-name **content**, complete with LocoScript carriage returns to separate the paragraphs.

**The first step in building up the text of the letter is to store the general information in content:**

**content=tellevarybody**

The next step is to work out whether the member's interests include Archaeology, and if they do, whether you have anything to say to archaeologists. Obviously, you only want to add something to **content** if **both** these conditions are met, so the instruction to check this is a multiple comparison using the AND (Section 6.1.2):

```
#interests="*A*" AND Aclause<"":<:content=content & cr &
 Aclause & cr:>>
```

So if the interests include **A** and **Aclause** currently stores some text, then that text (with a carriage return before and after) is added to **content**.

The next instruction repeats this procedure for the next interest group, and so on. As LocoMail works through these instructions, the correct text and spacing will be added to content.

When LocoMail has checked all five interest groups, all the general and specific text applicable to this member will be stored in **content**. The final instruction in this part checks whether a letter is necessary or not – because if **content** is empty then we can abandon this version immediately:

```
#content="":<:*>
```

Otherwise, LocoMail moves on past the angle brackets and produces a letter.

## Producing the letter

The third part of the master simply the framework of the newsletter – simple LocoScript text and codes – with slots for the member's name and address and for the text stored in **content**. At the end of each pass, this gives you a fully customised newsletter ready for printing.

## 12.5 Membership fees reminders

Unless all your club's annual subscriptions fall due in the same month each year, it is quite likely that each month a number of fees reminder letters will need to be sent out. Your membership records contain the month and year when the annual membership expires – so to produce the reminder letters for any particular month and year, all you need to do is compare the expiry date in the record with the date you are checking. Each member whose membership will have expired (or expires that month) will need a reminder letter and label.

### 12.5.1 Reminder letters

The master document to produce reminder letters where necessary is the FEESLETR master. As well as producing letters only for the appropriate members, the FEESLETR master document will insert a different 'amount due' depending on whether the person is a Full, Junior or Provisional member.

[illegible]

FEESLETR is a form of selective mailshot letter (as described in Chapter 6) because you only want to produce a version of the letter under certain circumstances. Indeed, the conditional instruction to work out whether to produce a version, and to abandon the version if not, is similar to the examples in Chapter 6.

The job of producing the letters can be broken down into three main steps:

- first, ask for the month and year you are producing the letters for
- then check whether the membership expiry date (stored in the current record) is after the date you are checking: if it is, the document is abandoned; if not, the master continues
- calculate the annual fees depending on the type of membership, and produce the letter

The FEESLETR master document tackles each of these steps in turn. Merge this document with CLUBFILE to see it in action.

### Asking for the month and the year

The month and year you want to produce the letters for will be used in later numeric comparisons, so the instructions ask for numeric values and store the responses under the item-names **checkmonth** and **checkyear** respectively. They also need to be carried over between passes, since FEESLETR only checks one record per pass, so the full instructions are:

```
!checkmonth=?#; Issue reminders for which month (1-12)? <
!checkyear=?#; and which year? (88-99)?
```

### Working out whether to produce a letter

The next step is the most important – to work out whether to produce the document or not. This uses a conditional instruction, with the \* instruction as the action – to abandon the letter if it is not required (see Section 6.1.3).

There are two possible circumstances when a reminder letter is not needed:

- when the year of expiry in the data file is after checkyear (ie. if **year>checkyear**), or
- when the years are the same, but the month of expiry in the data file is after checkmonth (ie. **year=checkyear AND month>checkmonth**).

In either of these cases, a letter is not needed and so the version can be abandoned. Putting these conditions and action together, gives us the instruction:

```
(year>checkyear) OR (year=checkyear AND month>checkmonth) :<:*><
```

Notice that we have surrounded each condition with brackets to clarify how we want LocoMail to make the comparisons.

### Producing the letter

The remainder of the master document is the text of the letter, with a few simple LocoMail instructions to insert the right details – in particular, the fee that is due.

As the fees are different for each type of membership, first you need to tell LocoMail what the fees are for each type of membership:

```
fullfee=25: juniorfee=15: provisionalfee=5
```

Now you need a short group of instructions to check what is stored in **membership** for the record, and store the appropriate value in a variable called **fee**. There are three possible membership types, so we have three instructions:

```
#membership="F*":<:fee=fullfee:><
```

which says "if the membership is full then set **fee** = **fullfee** (which we have set to 25)". Similarly,

```
#membership="J*":<:fee=juniorfee:><
```

```
#membership="P*":<:fee=provisionalfee:><
```

LocoMail now has all the information to complete the reminder letter.

### 12.5.2 Labels for the reminder letters

Producing the labels to go with the fees reminder letters is a very simple task, and is accomplished by the FEESLABL master. This master prompts for the month and year that you have produced reminders for, then use exactly the same conditional instruction as FEESLETR to work out whether a label is necessary. This conditional instruction is described above in Section 12.5.1.

Once again, if the dates typed at the keyboard are before the expiry dates, the document is abandoned. Otherwise a label is produced by inserting the **name** details, then a return, and then the **address** details. One label is produced for each pass through the master. The master document is shown below. Merge this master document with the CLUBFILE data file to see it in action.

```
!mail!title generates labels for the fees due letters
!checkmonth=?#; Issue reminders for which month (1-12)?
!checkyear=?#; and which year? (88-99)?
! (year>checkyear) OR (year=checkyear AND month>checkmonth) :<:*><
name({mail})
({mail})address
Produce label
```

**Note:** The Paper Type details in the document setup of FEESLABL document are set to a continuous page length of 9 lines, with a bottom gap of 1 line. You should change these Paper Type details to suit the particular labels stationery you intend to use. How to do this is described in Session 19 of the LocoScript 2 User Guide, or Session 20 of the PCW9512 User Guide.

## 12.6 Updating the membership details

As a club membership grows, the membership data file must be constantly updated to include records with the details of new members. You could do this by editing CLUBFILE, moving to the last record and inserting the new items of information complete with separators and end-of-record markers. However, the job of repeatedly typing in essentially similar details (eg. a name, then a separator, then an address, then a separator, etc.) is laborious and open to error – and, of course, a repetitive task like this is exactly what you want LocoMail to do for you.

UPDATE is a LocoMail master document designed to help you with this job. When you merge UPDATE with the CLUBFILE, the master creates a document that is effectively a copy of the data file, complete with record pattern, separators and end-of-record markers. When the last record in the data file has been read and copied, you are prompted to start typing in the new details in this document – the title, the name, then the address, then the phone number, and so on. LocoMail will automatically put in the necessary separators between these items, and put an end-of-record marker after the category.

After each new set of details, you are asked whether you have any more members to add. If you type Yes, then you are asked for a fresh set of details; if you type No, the master document takes this as a sign that you have no more names to add.

The finished document at the end of the pass is an updated data file – an exact copy of the original with the new details added. All you need to do is save it to disc (by selecting **Save result** from the Exit menu) and give it a suitable name (such as CLUBFILE.001). You can then use the new data file with all the masters described in this chapter.

UPDATE is a fairly long master. The way it goes about creating a new data file can be divided roughly into four stages:

- setting up the separators and end-of-record marker characters
- inserting the new record pattern
- copying all the current items, recreating the records by replacing the separators between items and inserting the end-of-record marker after each record
- taking the new information from the keyboard

Each of these stages is carried out using a combination of LocoMail instructions and sections of text. The following description covers each stage in turn.

**Note:** There are two points to note about using UPDATE. Firstly, you need to have at least one data record in the data file you are updating – it is not sufficient to just have the record pattern. Secondly, the file you produce using UPDATE can be quite large, so before you start you should check that there will be room for the file to be held temporarily on Drive M:

```

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038 1039 1040
```

**Program unit to copy all existing records from data file, re-inserting separators, and advance to next record**

**Program unit to collect all new details, again inserting separators, etc., until user types No**

## Setting up the separators, etc.

The first group of instructions in UPDATE - comma="," : etc. - sets up the separators and the end-of-record marker used in the CLUBFILE, and stores them under item-names. This is necessary, because although we can use simple LocoMail instructions to copy each piece of 'real' information from the data file into the new document, there is no LocoMail instruction to copy a separator or an end-of-record marker.

The characters defined are the ones used in the record pattern of CLUBFILE.

## Inserting the record pattern

The next part inserts a record pattern at the beginning of the document. This uses straightforward LocoScript text, because there is no instruction that allows you to copy details from the record pattern in the data file.

```
(-Mail)title, name ←
address;phone ←
interests ←
membership, month/year ←
category ↓
(+Mail)
```

## Copying the current information

Now that you've defined the separators and included a record pattern, the third stage is to copy across the details from all the records, re-inserting the separators between items and finishing each record with the end-of-record marker.

Fetching all the information from one record is easily done with instructions like (+Mail) name (-Mail) and (+Mail) address (-Mail). The separators can also be inserted easily because you have stored them under item-names. So to copy the details from every record, we just need to group these instructions into a program unit together with a \$+ instruction (to move on to the next record after copying all the details from the current record). This program unit is set up in UPDATE as loop:

```
loop=" " ←
title : comma : name : cr ←
address : semicolon : phone : cr ←
...
$+ ←
"
```

**Note:** Fetching details and re-inserting the separators could also have been performed using a concatenation instruction, such as record = title & " , " & name & ... (see Chapter 8). However, this would have meant nesting quoted strings within the program unit, which in turn would need extra Mail codes to ensure that so that the double quote marks wouldn't be taken as the end of the program unit loop (as described in Chapter 10). Nesting strings is best avoided - by storing the characters and using item-names as we have done.

This group of instructions is repeated, to read all the details in the current data file. We can conveniently control how often the loop is performed by using the instruction:

```
$loop@name
```

This instruction repeats loop until name is null - as will happen at the end of the data file (provided you have no 'nameless' records).

## Inserting new information

The final stage of preparing the updated data file is to add the new member's details from the keyboard. Once again, we use instructions grouped in a program unit to collect the details for one record (inserting separators and an end-of-record marker), and repeat this program unit until all the new records are added. The program unit we use is called newmember.

The instructions in newmember collect all the new details for each new member and insert the separators:

```
comma : ?; Title of new member (Mr, Mrs, etc.) ←
cr... ←
page ←
```

We also need to tell LocoMail when the new records are finished - otherwise it will continue repeating this sequence of instructions forever! Our best approach is to include an instruction to prompt whether there are any more records to be added, and use the response to this prompt to control whether the program unit is repeated. This technique is introduced in Section 9.4.2.

The process starts with the question **More members? Type Yes or No**. If the response is Yes (or anything beginning with Y), then we want to repeat the program unit; if the answer is No then we want to stop; if the response is anything else, we want the prompt to be repeated - so that we only stop adding records when absolutely certain there are no more. The short group of instructions to achieve this make up the PROGUNIT.IMP building block, described in Appendix III, and are shown below:

```
reply=? → ; More members? Yes or No ←
#reply=noword → :<:endinner=0:finish=0 ←
>#reply=yesword → :<:endinner=0:finish=1 ←
→ ><:endinner=1:> ←
```

The first instruction supplies the prompt and stores the response in reply; reply is then used to set the value of the variables endinner and finish. The value of endinner is used to control whether the prompt is repeated. The value of finish is used to control whether to repeat the program unit. The conditional instructions in this short group set finish and endinner depending on the response:

- if the response matches what is stored in **noword** then **finish** and **endinner** are set to 0
- if the response matches what is stored in **yesword** then **endinner** is set to 0 and **finish** to 1
- if the response is anything else, then **endinner** is set to 1 so that the prompt is repeated

Notice how including the PROGUNIT.IMP block in **newmember** gives us a string (picked out by double quotes) within a string. This means we have to ensure that LocoMail doesn't take either of the inner double quotes as the end of the program unit. In this case, all is well - because the Mail codes at the beginning and end of the program unit put the inner pair of double quotes at a different level to the ones surrounding **newmember**. LocoMail is therefore able to match the double quotes correctly.

The final step before **newmember** is carried out is to set up the acceptable responses that control whether the program unit and inner group of instructions are repeated or not (under the item-names **yesword** and **noword**):

**yesword**="Y\*" - so any word beginning with Y will be acceptable  
**noword**="No" - so only the word No will be acceptable

• If you want to store more information in the records of your data file at any stage (eg. you need more item-names in your data file), you can easily do this by making simple changes to **UPDATE**. Put the new item-name(s) and separator(s) in the record pattern details and in the program unit **loop**, then include suitable prompts and separators in the program unit **newmember**. When you copy the existing records by merging the amended **UPDATE** with the existing data file, the new item-name will be included (and will be set to null in all the existing records). Similarly, if you want to change the order of the items, you can do this simply by rearranging the details in **UPDATE**. When you are confident in the use of this master, you might try making these changes (working on a copy of **UPDATE**).

## Chapter 13

# Sales records and invoicing system

The example documents described in this chapter provide a simple system for maintaining sales records and generating invoices, reminders and debtor lists from these records. The documents are currently in group 4 of your 'LocoMail example documents' disc.

The system is built around a data file (ALLSALES) which contains the essential details of each sale - the date, amount, customer's name and address and the status (ie. paid or unpaid). The master documents are:

- NEWSALES** - adds details of new sales to the existing records
- INVOICE** - produces invoices for all sales made on a particular day
- RECEIPT** - updates the status of records when payment is received
- REMINDER** - produces reminder letters for unpaid invoices
- UNPAID.INV** - produces a list of all unpaid invoices, with totals
- TOTAL.INV** - produces a list of all sales in a given quarter

To see these masters in action, simply merge them with the data file.

In the rest of this chapter we describe how the supplied masters work, and how (and why) the instructions and techniques covered in Parts II and III have been applied to the different tasks. This will help when you want to create your own masters to use the information in the data file, or include extra information in the data file, or adapt any of the supplied masters.

## 13.1 The data file

The data file for the sales and invoicing system consists of a number of 'sales' records; each record contains the details of one particular sale. The information which is stored, and the nature of that information, is determined by the way it is used in the master documents. The item-names for each piece of information are as follows:

|                     |                                                        |
|---------------------|--------------------------------------------------------|
| <b>title</b>        | - Mr, Mrs etc.                                         |
| <b>name</b>         | - customer's name                                      |
| <b>address</b>      | - customer's address                                   |
| <b>amount</b>       | - the value of the sale (numeric and VAT-inclusive)    |
| <b>day</b>          | - day of sale (numeric, in the range 1 - 31)           |
| <b>month</b>        | - month of sale (numeric, in the range 1 - 12)         |
| <b>year</b>         | - year (numeric, 2-digit value, in the range 00 - 99)  |
| <b>order_number</b> | - a (numeric) order number, used as the invoice number |
| <b>status</b>       | - either Paid or Unpaid                                |

**Note:** We've said that the amount must be a VAT-inclusive value, to match the way the master documents have been designed. We could just as easily have set up the masters to use VAT-exclusive values and stored these in the data file – the most important thing is that to get the right results, we must be consistent. In addition, if you are using these example documents outside the UK, you will need to change the arithmetic instructions to allow for your local sales tax, etc.

The separators have been chosen to match the different items of information, and to give a suitable layout to the records:

```
title, name ←
address; amount ←
day/month/year ←
order_number, status ←
```

The real data records follow this pattern. The one condition that the data records must obey for these master documents to work correctly, is that there are no 'nameless' records. This is because many of the masters repeat 'program units' of instructions (see Chapter 9) until the name is null, which is automatically the case when the end of the data file is reached. If you have any 'nameless' records, they'll stop too soon!

The data file is stored as ALLSALES. If you want to look at the example data records, edit this document.

## 13.2 Adding records to the data file

As sales are made, the details will need to be added to the existing records. You could do this directly, by editing the data file using LocoScript and adding the new details to the end of the file – typing the title, then the separator dictated by the record pattern, then the name and the next separator, and so on. However, this is a repetitive task, and also very prone to errors (eg. mistyping or forgetting separators). LocoMail, of course, is very good at performing repetitive tasks – so a LocoMail master document can help you add the new records. You still have to type in the customer's name, address, sales amount, and so on, but LocoMail instructions in the master handle separators and end-of-record markers for you.

The master document that will do this for you is NEWSALES (shown opposite). To see the NEWSALES master in action, merge it with the ALLSALES data file. You don't have to save the finished documents when practising.

The master document has to create a data file with exactly the same structure as the ALLSALES data file. LocoMail cannot insert new details straight into the existing data file, so NEWSALES approaches the task in a different way. It:

- starts by creating a record pattern identical to the one in the existing data file
- then copies all the existing sales records (complete with correct separators and suitable end-of-record markers)
- finally inserts new records from the keyboard, again complete with separators and end-of-record markers

Setting up the separators and other characters used in the document

Putting in the record pattern at the start of the new data file

First program unit to copy one record then advance to next

Program unit repeated until end of datafile reached

Program unit to add new details from keyboard

```
(Mail)
; define the separators and end-of-record markers
;
; set up the record pattern
;
; insert the record pattern
;
; set up instructions to copy all the current records
;
; repeat this loop until a null name is found
;
; all existing records have been duplicated - prompt for new datafile
;
; title (Mr, Mrs, etc)
;
; Customer's name
;
; Customer's address (the town for new lines, enter when finished) (Unit)
;
; Invoice amount
;
; Day (1-31)
;
; Month (1-12)
;
; Year (2-digit number)
;
; Amount (2-digit number)
;
; Paid or Unpaid (Y or N)
;
;
; Now use a group of instructions to work out whether mainloop is to be repeated
;
; Add more sales records? Yes or No?
;
; If user types No, mainloop finishes
;
; Send master to finish - if user types Y, mainloop is repeated
;
; Send master to finish - if anything else, response is ignored
;
;
; Store the acceptable responses under master
;
; Note: Willcard file response can be anything beginning with Y
;
; No response can only be the word No
;
; Newsalesfinish (Mail) (Mail)
;
;
; Press enter then select 'Save' option to save sales file (Unit)
;
; Choose a suitable name for the new data file (Unit)
;
; Then use the INVOICES master to produce invoices for the new sales (Unit)
```

These are essentially three separate tasks, and so the master document is in three main parts, each part corresponding to one of these tasks. NEWSALES also begins with a short sequence of instructions to set up certain characters used throughout the document, and finishes with a reminder to save the finished document.

**Note:** The data file that you use with NEWSALES must contain at least one data record – it is not sufficient to just have the record pattern. Also, the file you produce using NEWSALES can be quite large, so before you start you should check that there is enough room on Drive M for the file to be held as it is created.

### Inserting the record pattern

Our updated data file has to start with the same record pattern as the ALLSALES data file. LocoMail instructions pick out information from the data records in the data file, not directly from the record pattern, so you can't use LocoMail instructions to copy the record pattern itself. However we know what the pattern is, so we can simply insert it as text.

### Copying the existing records

The task of copying all the records from the data file into the finished version is simply one of inserting the title from the current record, then the correct separator (a comma), then the name and another separator (a carriage return), and so on. When all the details from one record have been inserted and followed with an end-of-record marker, we need to repeat the process for the next record.

This type of 'looping' action is ideally suited to using a program unit. What we need in the program unit is a sequence of instructions that will make an exact copy of one data record, then move on to the next. We can then repeat it until we reach the end of the current data file. (Setting up and using program units is described in Chapter 9.)

Each item in the record is identified by an item-name, so to insert the information into the new data file, we just use the appropriate item-name (eg. `(+Mail)title(-Mail)`). But you can't use the same technique to copy the separators from between items in the record, or the end-of-record marker after the last item. However, we know what the correct separators are from the record pattern, and we can always use the `↓` as the end-of-record marker, so we can put these separators in ourselves.

There are essentially two ways of putting the right separators back between the items of information we copy:

- either – re-insert the characters by using the concatenation-type instructions introduced in Section 8.5 to build up an item (eg. `record`). Then use `(+Mail)record(-Mail)` to insert these details.
- or – store the separator characters under item-names and insert them from memory.

The second method is typically a better approach, since it makes the master document much clearer and easier to follow. Moreover, unlike the first method, the program unit doesn't need to contain any quoted strings; you have to be very careful when embedding strings within program units, as explained in Chapter 10. All the required separator characters must be stored under item-names **before** they are used in the program unit; you can see that this has been done right at the start of the NEWSALES master.

The final job of the program unit is to move on to the next record. As described in Section 6.2, this can be done using the instruction `$+`.

All these commands are gathered together in the program unit we've called `loop`. To copy every record in the data file, we need to ensure that the `loop` program

unit is repeated for all the records. We probably don't know how many records there are in the data file, but this is not a problem. As described in Section 9.4, we can conveniently control how often the program unit is performed by using the fact that when LocoMail reaches the end of a data file, it automatically sets all values to null. So to repeat the program unit we use the instruction:

```
$loop$name
```

This says 'perform the program unit `loop` until `name` becomes zero or null'. That's why we said earlier that you should have no nameless records in the data file. `loop` will now be repeated until the last record has been copied.

### Adding new records

Once all the existing records have been copied from the data file into the finished version, the final job of our master document is to add the new sales records. Again, this task is essentially a repetitive one; creating a new record, then another, then another, etc. So once again, the best way of approaching this task is to use a program unit to collect up the details for one new record, and repeat this program unit until there are no more details to be added.

#### • Fetching new details

The instructions in the program unit need to fetch new details from the keyboard and insert them into the finished version, putting the correct separators back between the typed details and an end-of-record marker at the end:

- Fetching and inserting details uses simple `(+Mail) ? ; prompt (-Mail) , or (+Mail) ?# ; prompt (-Mail)` instructions
- Inserting the correct separators between details is also very easy because we have already stored them under appropriate item-names.

So to create one new record, we can simply alternate the `?`; `prompt` type instructions for each data item in a record with the item-names representing the separators. Automatically inserting the separators in this way means that you cannot forget or mistype them! Putting all these instructions one after another would make the master very cluttered, even with `←`s at the end of each prompt. However, we can improve the layout by using tabs, as follows:

```
(+Mail) → ? ; Title (Mr, Mrs, etc.) ? ←
comma : → ? ; Customer's name ? ←
... (-Mail)
```

#### • Repeating the program unit

To create the new sales records, we also need to tell LocoMail when the new records are finished – otherwise it will continue repeating this sequence of instructions forever!

The number of new records will not necessarily be the same each time you use the master. In these circumstances, our best approach is to include an instruction to prompt whether there are any more records to be added, and use the response to this prompt to control whether the program unit is repeated. This technique is introduced in Section 9.4.2.

## Extra features in the master

The instructions we've described above form the 'core' of the master document. There are, however, a few extra 'niceties' that you might think about – some of which we've included in NEWSALES.

- **A useful reminder prompt**

At the end of the pass you're left with a finished document which you must save to disc if it is to become the new data file. So the last instruction in NEWSALES displays a simple reminder of this fact.

- **Using Unit codes in long prompts**

Unit codes give us a neat way of getting back to the start of a new line before waiting for a typed response. In particular, they allow us to avoid scrolling the screen to the right when replying to a long prompt, and to keep all the lines of multi-line information (such as addresses) together on the screen as they are typed. A Unit code has the same carriage return effect as the `↵` character, but it doesn't act as a command separator, so LocoMail doesn't think it has reached the end of the instruction at that point.

- **Numbering new records automatically**

It isn't actually necessary to type in a new order number for each new record. Because the invoice numbers of the new records run sequentially, we can use LocoMail's counting instructions to automatically number them.

So that LocoMail knows from what value to start numbering the new records, we record the `order_number` of the last of the existing records. This is stored by including one extra instruction in the loop program unit:

```
invnum=order_number
```

When all the existing records have been copied, `invnum` will hold the same numeric value as the `order_number` of the last record. Now, instead of an instruction prompting for an `invoice_number` in `newsales`, we simply use `+invnum`. This counting instruction adds 1 to the number stored in `invnum`, inserts the new value in the new data file and stores the new value. As `newsales` is repeated, each new record will have an `order_number` one greater than the previous record. (See Section 8.2.)

## 13.3 Producing Invoices

Possibly the most important documents you want to produce on a regular basis are invoices for a particular day's sales. Again much of the task is just the sort of thing LocoMail is best at – repetitive checking of sales dates, the status of records (paid or unpaid), slotting in pieces of information, calculating totals, etc. With a LocoMail master to help you produce these invoices, you can decide on the sales date for which invoices are to be produced and LocoMail will produce an invoice for each record made out on that date where the status is unpaid. As you type in the product details and prices into the invoice, LocoMail will lay the details out correctly, calculating VAT and the running totals.

## Chapter 13: Sales Records and Invoicing System

The process starts with the question **Add more sales records? Type Yes or No**. If the response is Yes (or anything beginning with Y), then we want to repeat the program unit; if the answer is No then we want to stop; if the response is anything else, we want the prompt to be repeated – so that we only stop adding records when absolutely certain there are no more.

The short group of instructions to achieve this are shown below. These make up the PROGUNIT.IMP building block, described in Appendix III:

```
reply=?→ ; Add more sales records? Yes or No↵
#reply=noword→ :<:endinner=0:finish=0↵
>#reply=yesword→ :<:endinner=0:finish=1↵
→ :<:endinner=1:>↵
```

The first instruction supplies the prompt and stores the response in `reply`. `reply` is then used to set the value of the variables `endinner` and `finish`. The value of `endinner` is used to control whether the prompt is repeated. The value of `finish` is used to control whether to repeat the program unit. The conditional instructions in this short group set `finish` and `endinner` depending on the response.

Notice how including the PROGUNIT.IMP block in `newsales` gives us a string (picked out by double quotes) within a string. This means we have to ensure that LocoMail doesn't take either of the inner double quotes as the end of the program unit. In this case, all is well – because the Mail codes at the beginning and end of the program unit put the inner pair of double quotes at a different level to the ones surrounding `newsales`. LocoMail is therefore able to match the double quotes correctly. (Embedding strings within strings is explained in Chapter 10.)

The remaining aspect of NEWSALES to look at is the joins between these sections – an important point to consider in any LocoMail document.

The first join to think about is the one between the initial block of definitions and the part that sets up the record pattern. The latter comprises fixed text, so the initial block of definitions and comments finishes with a `(-Mail)` code.

The rest of the master document is made up entirely of LocoMail instructions, so you might assume that all you have to have is a `(+Mail)` code after the fixed text. However, the presence of program units makes matters more complicated. Each program unit is made up entirely of LocoMail instructions, starts with a `(+Mail)` and finishes with a `(-Mail)`. This ensures that LocoMail is active as it enters the program unit but it also means that it is inactive at the end of the looping, so we need to follow each 'loop' instruction with a `(+Mail)` to make LocoMail active for the next instruction:

- For the first 'loop' instruction (`%loop@name`), we've put the `(+Mail)` after the `↵` which acts as the command separator.
- For the second 'loop' instruction (`%newsales@finish`), we've used `(-Mail) (+Mail)` as the separator. This means that we automatically have a `(+Mail)` following the loop, since the loop is carried out as if it were positioned after the `(-Mail)`.

[illegible]

The master document that will help you to do this is the INVOICE master. Merge the INVOICE master document with the ALLSALES data file to see it in action.

The way to follow the master document is to look at the finished invoice it must produce:

216

## LocoMail User Guide: Part IV – Applications

88904

|                         |                               |
|-------------------------|-------------------------------|
| VAT number: nn nn nn nn | Your company name and address |
|-------------------------|-------------------------------|

Invoice Number : 1

To:  
A Brown  
Brown House  
Brown Lane  
Browtown

5/1/88

For foods supplied on 1/1/88

| No | Item description     | Price (each) | Price (each) rate | VAT amount | Price      |
|----|----------------------|--------------|-------------------|------------|------------|
| 2  | Ash hammer handles   | 2.25         | 4.50              | 15         | 0.87 5.17  |
| 3  | Ratchet screwdrivers | 4.48         | 13.47             | 15         | 2.02 15.49 |
| 1  | Tempered Brodau      | 2.99         | 2.99              | 15         | 0.44 3.43  |

**TOTAL \$:**

913 2188

**Creating this finished document involves a number of different stages:**

- working out whether the current record meets the conditions, and if not, abandoning the version
- inserting the 'Header' details (fixed pieces of information and simple details from the data file or keyboard)
- printing multiple lines on the invoice
- calculating totals with a suitable format

Each of these different steps is carried out by the master document, and in much the same order as it appears on the finished invoice. However, the last two stages are repeated together in the same program unit.

**IMPORTANT:** If you intend to use the master document in preparing invoices for your business, you should satisfy yourself that the VAT-related calculations in the master are correct for your own circumstances.

## Working out whether an invoice is necessary

The first group of instructions in INVOICE prompts for the sales date you want to produce the invoices for. Each of these typed values can then be compared with the corresponding day, month and year in the current record to see if they match.

Essentially, we just need *item-name*=? type instructions, but we also want to compare the typed day, month and year values using numeric-type comparisons and we don't want to have to type the same date again in each invoice, so the instructions we use are:

```
!checkday=?#; Which day's sales...
!checkmonth=?#; and which month (1 - 12)?
!checkyear=?#; and which year (88 - 99)?
```

We then need to abandon the document straight away if any of the typed values don't match the stored ones (ie. when `checkday<>day`, etc.), or if the

## LocoMail User Guide: Part IV – Applications

217

status part of the record is paid (ie. `status="p*"`). The best way of making these multiple comparisons is using ORs in the conditional instruction:

```
#checkday<>day OR checkmonth<>month OR checkyear<>year
OR status="p*":<:*>
```

With this instruction, if any part of the date doesn't match or if the status is paid, this version is abandoned. Otherwise, LocoMail carries on and produces an invoice. (Using a conditional instruction for a 'selective' master document in this way is described in Section 6.2.1.)

### Inserting the 'Header' details

The top part of the finished invoice consists of a mixture of fixed text and simple details like the customer's name. Wherever fixed pieces of text (headings, ruled lines, etc.) are required, these are put straight into the master, using the LocoScript codes and tabs to space out the details correctly. Inserting simple details from the data file, such as the customer's name and address and order number, uses instructions of the type `(+Mail) item-name (-Mail)`.

The 'For goods supplied on' line requires slash characters between the day, month and year values taken from the data file. One way of inserting these characters is to alternate LocoMail instructions with LocoScript single characters, but a better method is to store the slash character under a suitable item-name (eg. `slash`) and insert the character from memory – as we did with the separator characters in Section 13.2. So for the 'goods supplied date' we use:

```
(+Mail) day:slash:month:slash:year (-Mail)
```

### Building up lines of the invoice

Creating the lines of the invoice is a repetitive task, ideally suited to using a program unit of instructions (as described in Chapter 9). What we need in the program unit is a sequence of instructions that will:

- collect all the details from the keyboard for one line,
- calculate the line-totals and maintain the running totals, and
- control how often the group is performed.

The details you'll need to provide for each line of the invoice are the number of items, the item description, and the price per item and whether or not the price is subject to VAT. The quantity and price are used in the calculations, so we must accept only numeric input.

The columns for the gross price, the VAT amount and the line total are all calculated from the typed values. We want each value to be given in pounds and pence. So the arithmetic instructions we use all need to work in pounds, and include a format specifying 2 decimal places (ie. pence) – rounding or truncating as necessary – starting with `itemot=[price|2]`. We've used an instruction like this so that the unit cost (`price`) of, say, £2.00, can be entered as 2 and the figure in the invoice will have the correct format.

The value for the gross price column is calculated from the unit cost and the quantity, and is similarly rounded to the nearest penny:

```
gross=[price*quantity|2]
```

The values to be inserted in the VAT rate and VAT amount columns obviously depend on the whether VAT is payable on this item or not. To check this, we have a prompt for information and use a conditional instruction:

```
#VATcode="Y" :<:vatrate=15:VAT=[gross*vatrate/100|2]|<
>:vatrate=0:VAT=0:><
```

So if VAT is payable, then `vatrate` is set to 15(%) and the VAT amount is calculated at 15% of the value stored in `gross`. The format here includes a ! because VAT amounts are truncated. If response to the prompt wasn't Y, then `vatrate` and the VAT amount are both set to 0.

The final value to be calculated is the net total. The instruction is simply:

```
net=[gross+vAT|2]<
```

At this stage, `gross` holds the gross price, `VAT` holds the vat amount and `net` holds the net amount for this line. To keep running totals of these amounts, we add the calculated values for the current line of the invoice to the totals, using:

```
subtot1=[subtot1+gross]<
subtot2=[subtot2+vAT]<
subtot3=[subtot3+net]<
```

Subtot1, 2 and 3 have to be 0 at the start of each new invoice. So in our group of 'definitions' at the top of the master we have:

```
subtot1=0 : subtot2=0 : subtot3=0
```

The next step is to print out the typed details and calculated values that make up one line of the invoice. To give the correct spacing between details and lines in the finished invoice, we need tabs along the line and two carriage return characters at the end (so that we have a completely blank line between each entry). To stop the master being cluttered, we store the tab and carriage return characters under item-names, then insert the characters from memory (as in the NEWSALES master). The line on the invoice is then printed using:

```
quantity:tab:item:tab:subtot:tab:gross:tab:vAT:tab:net:cr:cr
This gives us a tab between each piece of information, and two carriage returns after the details, as required.
```

All the instructions to build up one line are in the `itemline` program unit, and the lines of the invoice are built up simply by repeating this program unit using the 'loop' instruction `&itemline@finish`. However, to do this we need some way of telling LocoMail when there are no more lines to be added. The technique used here is the method described in the NEWSALES master document, Section 13.2. We simply prompt whether there are any more lines. Then we control whether the program unit is repeated or not depending on whether the user types Yes or No, and repeat the prompt if anything else is typed. As in NEWSALES, we use the `PROGUNIT.IMP` building block to do this.

The last job of the master is to print out the running totals (`subtot1`, `subtot2` and `subtot3`) under the appropriate columns. To do this we must insert the right number of tabs before each total, as follows:

```
(+Mail) tab:tab:subtot1:tab:tab:subtot2:tab:subtot3 (-Mail)
```

### 13.4 Updating the data file

To ensure that the other masters in the sales records and invoicing system give you the right results from the data file, you need to update the status of unpaid records whenever payment is received. You could do this by using LocoScript to edit the data file, then scrolling through the records updating the status of records from Unpaid to Paid as appropriate. However, this method cannot be recommended; it is time consuming and all too easy to corrupt your data file, for instance by deleting a separator or end-of-record marker.

A more efficient method is to use a LocoMail master document to help you update the records and, in particular, to make sure that you update them reliably. The master document that will do this for you is the RECEIPTS master. To see RECEIPTS in action, merge it with ALLSALES.

[illegible]

The task of updating records is similar to the job performed by the NEWSALES master described in Section 13.2, which started with the record pattern, then used a repeated program unit of instructions to copy all the existing records in a data file and replace the separators and end-of-record markers.

Whenever the status is 'paid', our master simply needs to copy the record. Each time an 'unpaid' record is met, however, the master must ask whether payment has been received, and if it has, change the status in the new document to 'paid'. This process continues until all the records have been copied into the new document. The new document can then be saved on disc and used as your updated data file.

The RECEIPTS master approaches the task into two stages. It:

- inserts a record pattern identical to the one in the existing data file
- recreates all the existing sales records (complete with suitable separators and end-of-record markers), updated as necessary

These two stages are carried out by two different parts of the RECEIPTS master. In addition, the document starts with a 'definition' group of instructions, to set up the characters used throughout the master. We will concentrate on the LocoMail instructions that make up the two main parts.

## Inserting the record pattern

As in the NEWSALES master document, the record pattern is inserted in the finished document by including it directly in the master:

```
title, name ←
address; amount ←
day/month/year ←
order number, status ↓
```

## Copying the existing records

Copying the records, updating them as necessary, is carried out by the program unit **loop**. The instructions that make up **loop** are as follows.

- Copying the straightforward details

Most of the details in each record will be copied straight into the finished document; the only detail which might change is the status. Copying the details just requires the appropriate item-name (eg. `(+Mail)title(-mail)`), but, as we saw in NEWSALES, we have the problem of the separators, which we can't transfer by this technique. RECEIPTS uses the same solution as NEWSALES – we store the appropriate characters under item-names then insert them direct from memory. (Remember that the separator characters need to be stored under item-names **before** they are used in the program unit – so we've put them right at the start of the master document in our 'definitions' group.)

Recreating most of the record is then a case of alternating the item-names of the details with the item-names of the stored separators:

```
title : comma : name : cr ↵
address : semicolon : amount : cr ↵
day : slash : month : slash : year : cr ↵
order number : comma ↵
```

### • Dealing with the status

The next step is to check the **status** of the record, and if necessary, update it. LocoMail needs to take different actions depending on this status, so we use a pair of conditional instructions to ensure:

- if the current status is paid (or anything beginning with P), then we simply copy it across to the new document
- if the status is unpaid (or anything beginning with U), we must ask whether payment has been received and update the status only if it has

The basic condition we need to test here is whether status begins with a P, ie. if **status="P\*"**. We could state the condition just like this, but instead we store **P\*** as the item **paid** (using **paid="P\*"** in our 'definitions') and test the condition **status=paid**. Using **paid** rather than **"P\*"** has two advantages: firstly, it makes the master generally easier to read; secondly, it avoids the problem of extra double quotes in the program units in this master document.

If the current status is paid, we simply have to copy **status** across, so the first of our conditional instructions is:

```
#status=paid:<:status:>
```

The action for unpaid records is slightly more complicated. When the status is unpaid (ie. **#status=unpaid**, with **unpaid="U\*"** set up in our 'definitions'), we must prompt whether payment has been received or not. If it has, the status must be updated. If not, the status remains the same. If the answer is anything else, we need to ask again, so that we only change the record when absolutely certain.

The group of instructions we use to do this is a variation of the PROGUNIT.IMP building block. This was used in NEWSALES (Section 13.2) and INVOICE (Section 13.3) to control the repetition of a program unit. The difference here is that instead of setting **finish**, we either insert a **P**, or copy over the current **status**, or ask again depending on the answer to the prompt:

```
"reply=?;Please type Paid or press ENTER"
→ #reply=paid <:endinner=0:(-Mail)P(+Mail)
→ >#reply=null <:endinner=0:status
→ ><:endinner=1 >
→ "&endinner"
```

If the response to the prompt begins with P, the value of **endinner** is set to 0 and the letter **P** is inserted into the new data file. If the response is null, the current **status** is copied and **endinner** set to 0, and if the response is anything else, the prompt is repeated – as required. (The null item is set up in our 'definitions' by **null=""**).

There are two important things to note about this group of instructions:

- The **status** in the new data file will be either P or the current value of **status**. These details are inserted directly by LocoMail instruction, so you won't get records with mistyped status values!

- The group of instructions includes a test of whether the reply given is **paid**. This really tests whether the reply starts with P, and so we could have expressed it as **#reply="P\*"**. However, this would have given us a string within a string, and we would have needed extra Mail codes to ensure that LocoMail interpreted it correctly. Using the item-name **paid** avoids the problem altogether and is therefore strongly recommended.

The instructions to copy and update the records then finish with ones to add the end-of-record marker (stored under **page**), and move on to the next record:

```
page
$+
```

Notice how including the inner group of instructions in loop gives us a string (picked out by double quotes) within a string. This means we have to ensure that LocoMail doesn't take either of the inner double quotes as the end of the program unit. In this case, all is well – because the Mail codes at the beginning and end of the program unit put the inner pair of double quotes at a different level to the ones surrounding loop. LocoMail is therefore able to match the double quotes correctly:

```
loop="(+Mail)
→ Title...
→ "reply=...
→ "&endinner
...
(-Mail)"
Double quotes
surrounding
loop
```

(Embedding strings within strings is explained in Chapter 10.)

Updating the data file is now a case of repeating loop for all the records. As described in Section 13.2, we can conveniently ensure that all records are copied by using the fact that when LocoMail reaches the end of a data file, it automatically sets all values to null. So to repeat the program unit we use the instruction:

```
$loop@name
```

Loop will now be repeated until the last record has been copied, provided you have no 'nameless' records.

Putting these section together to form the RECEIPTS master is straightforward. We start with the initial group of 'definitions', then use a **(-Mail)** code before the 'fixed' text that is to be inserted (the record pattern). After this text, the rest of the master consists only of LocoMail instructions, so we put a **(+Mail)** code before the instructions.

The loop instruction is followed by a **(-Mail)** code and so loop will be performed as if it followed this **(-Mail)** code. The **(+Mail)** at the beginning of the program unit, which makes LocoMail active at the start of the loop, is balanced by the **(-Mail)** at the end of the program unit.

### An enhancement to the master

The master document we've described searches through a data file looking for all records where the status is unpaid. You can easily change this master to search for more specific details such as a customer's name, as follows.

The first step is to include an initial instruction prompting for the name of interest, and store it (eg. as **customer**):

```
(+Mail) customer=?; Customer's name? (-Mail)
```

The conditional instructions in the Loop program unit can then be modified so that they only offer the chance of updating records which are made out to this customer and unpaid:

```
#status=paid OR name<>customer: <:status<
→ ><%"reply" = ?...
→ ...
→ @endinner<
→ >
```

Notice that we've used the IF...THEN...ELSE type instruction, so the operation of the conditional instruction is different; all records that have a status of paid or that are made out to other customers, will have the current **status** copied. Only records that are unpaid and with the right name will give the prompt now.

If you want to allow for the case where you don't know exactly how the name is specified, add wildcards to the details stored in **customer** before making the comparison:

```
(+Mail) customer=?; Customer's name?<
searchname="*" & customer & "*" (-Mail)
```

The conditional instruction described above then uses **name<>searchname**.

## 13.5 Reminders for overdue invoices

Another document that is needed all too regularly is a letter requesting payment of overdue invoices! Obviously, we won't (normally) want to produce these reminder letters for every record in the sales records data file. Instead, the letters must only be produced for selected records – such as those with a sales date before a specific date, and where the status of the record is still unpaid.

The task of producing letters only when certain conditions are satisfied is essentially a form of 'selective' mailshot, and the master that will help you produce these letters is the REMINDER master.

To see the REMINDER master document in action merge it with the ALLSALES data file.

```
(+Mail) produce reminders for all sales unpaid at given date:
Fetch date of interest
Convert typed date and stored date to numbers
Abandon version if status is paid or if date is after date of interest
Otherwise continue with letter

Your company's name and address

To: (Mail) (name) (Mail) (align) (Mail) (Mail) (Mail) (Mail) (Mail)
From: (Mail) (name) (Mail)
Subject: (Mail) (Mail) (Mail) (Mail) (Mail) (Mail) (Mail) (Mail) (Mail) (Mail)
I note from our records that our invoice number (Mail) (number) (Mail), for goods supplied on (Mail) (day) (month) (year) (Mail), is overdue for payment.
The amount invoiced is £ (Mail) (amount) (Mail).
Please make arrangements to send us this amount immediately, or contact our Sales Department if there is some difficulty with the order.
Yours faithfully
Your name
```

As with any 'selective' document, there are two fundamental steps to producing the correct finished document:

- first, work out whether the current sales record needs a reminder, and if not, abandon this version
- produce the reminder letter

The REMINDER master is made up of two distinct parts, carrying out the steps in the same order.

### Working out whether a letter is required

To work out whether a version is required or not, LocoMail first needs to know the date of interest. All records before this date (and where the status is unpaid) will be considered overdue. You need to type these dates in; for example, if you want to produce reminder letters for sales still unpaid after 30 days, you would type the date 30 days ago.

Essentially, we just need *item-name*=? type instructions to gather the day, month and year that you want to compare the invoice dates against. However, the dates you type need to be used in arithmetic and numeric-type comparisons, so they must be numeric. Also, you don't want to have to enter the same details on each pass, so the actual instructions are:

```
!checkday=?#; Produce reminders ...↵
!checkmonth=?#; ...↵
!checkyear=?#; ...
```

The next step is to get these details into a format suitable for comparison with the values in the record.

You cannot just check whether the typed *checkday* is before the value stored in *day*, and the typed *checkmonth* is before the value stored in *month*, etc. – because this would give you the wrong result in certain circumstances (such as the end of years). The test needs to be a little more complicated. There are essentially two ways of going about this:

- either – work out a sequence of comparisons that will check first the year; then depending on the result of that test, the month; then, depending on the result of that, the day.
- or – combine the day, month and year details to give a number, the magnitude of which is related to the date.

In this case, we will use the latter approach. What we need to do is express both the typed date and the sales record date as numbers, which will give the right results when we compare them. For the later of these two dates, the number must be greater; then, a simple comparison between the two values will tell us whether the sales date precedes the date of interest or not. The way we do this is as follows.

Multiply the 2-digit year figure by 10000, and the month figure by 100. If you add these two values to the day, you will always end up with a six-digit number. For example, day 28 of the month 7 and the year 88 will give you the result 880728 (880000 + 700 + 28). Any date before this will give you a number smaller than 880728, and any date after it will give a larger number (try it!). So you can easily tell whether the date stored is before or after the date typed.

(Note: You could use concatenation to create a six-digit number for the date, but you would have to be very careful about how the values were typed in and handled. Instead, we recommend using arithmetic as we've described – it is much more straightforward and safe for the job we're doing.)

In REMINDER, we apply this arithmetic to the values typed in (*checkday*, etc.) and the values in the data record (*day*, etc.), and store the result as *reminderdate* and *salesdate*:

```
reminderdate=[checkyear*10000+checkmonth*100+checkday]
salesdate=[year*10000+month*100+day]
```

and we test whether *salesdate*>*reminderdate* to see whether the reminder letter is to be abandoned.

The other condition under which the reminder letter is to be abandoned is when the status is paid. The two conditions therefore give us the conditional instruction:

```
#salesdate>reminderdate OR status="P*":<:*>
which says "if salesdate is after the date of interest, OR if the current status is paid, then abandon this version.
```

If neither condition is met, a reminder letter is necessary.

### Producing the letter

The rest of the master document is the letter. A typical reminder letter might be:

Your company  
name and  
address

To:  
Mr A Brown  
Brown House  
Brown Lane  
Browntown

3rd February 1988

Dear A Brown

I note from our records that our invoice number 1, for goods supplied on 1/1/88, is overdue for payment.

The amount invoiced is £46.00.

Please make arrangements to send us this amount immediately, or contact our Sales Department if there is some difficulty with the order.

Yours faithfully

Your name

This letter is created just as you would create any LocoScript standard letter, using the simple *(+Mail)item-name(-Mail)* and *(+Mail)!item-name(-Mail)* type instructions to insert details. But note how the slash character, used in the text of the letter to separate the day, month and year, has been stored under a suitable item-name using the instruction:

```
slash="/"
```

It is then inserted from memory. This same technique was used in the earlier sections of this chapter.

Putting these instructions together to form REMINDER is quite straightforward. After the initial group of instructions, we precede the 'fixed' text of the letter with a *(-Mail)* code. At the points in this text where details are to be inserted, we use *(+Mail)item-name(-Mail)* as usual.

## An enhancement to the master

One enhancement you might make to this master is to change the way the dates are collected and compared to produce different sets of reminder letters. Because of the way in which REMINDER processes typed dates, such changes are easy to achieve.

For instance, you might want to produce reminder letters for unpaid invoices made out on a specific date. In this case, the only change to REMINDER needed is to the numeric comparison in the conditional instruction:

```
#salesdate<>reminderdate OR status="P*":<:*:>
```

Similarly, you might want to produce letters for sales made in a certain week. To do this, you would simply collect the dates at the start and end of the week:

```
!checkday1= ?#;Enter day at start of week
!checkmonth1= ?#;Month
!checkyear1= ?#;...
and
```

```
!checkday2= ?#;Enter date at end of week
!checkmonth2= ?#;...
```

**Then convert these to six-digit numbers using:**

```
reminderdate1=[1000*checkyear1+...
reminderdate2=[1000*checkyear2+...
```

The conditional instruction to abandon unwanted letters must then check whether the salesdate is before the first date or after the second or the status is 'paid':

```
#salesdate<reminderdate1 OR salesdate>reminderdate2 OR
status="P*":<:*>
```

### 13.6 List of debtors

The data file of sales records contains details of the status of each sale. To keep a check on the monies owed, typically you want to produce a list of all the bad debtors, total of unpaid sales, etc., such as this:

**LIST OF ALL UNPAID SALES**

| Invoice               | Customer | Sales date | Amount  |
|-----------------------|----------|------------|---------|
| 1                     | A Brown  | 1/1/88     | 46.00   |
| 3                     | C Jones  | 3/1/88     | 10.23   |
| 5                     | E Bloggs | 5/1/88     | 17.99   |
| 6                     | F Green  | 4/1/88     | 27.25   |
| TOTAL DUE (incl VAT): |          |            | £101.47 |

The master document that will do this for us is the UNPAID.INV master. To see UNPAID.INV in action, merge it with the ALLSALES data file.

[illegible]

The techniques used by UNPAID.INV, both to insert the fixed details and to work through every record in the data file, are similar to the methods used in Section 13.2. It starts with a short group of 'definitions' to set up the characters used in the master.

Including the fixed pieces of information in the finished document is simply a case of using LocoScript text, codes, tab characters etc. to give the correct appearance, as we have done in the previous examples.

The task of checking through all the records in the data file, and inserting details from the unpaid records into the finished document is a repetitive one. UNPAID.INV therefore uses a program unit to look for records in which the status is unpaid, copy the details from this record into the finished document, then advance to the next. The program unit is repeated for every record in the data file and so it picks out all the unpaid records. This program unit is called **loop**.

## The program unit

The first job of the program unit is to check the status of the current record: if the status is paid (or anything beginning with P), then we can immediately move on to the next record. If the status isn't paid, then we need to insert the details from the current record. So the first instruction is:

```
#status="p*":<:$+:X...
```

which says "if the status is anything beginning with P, move on to the next record. Else..."

The next task of the program unit is to insert one line of details, as set out in the design of the finished version:

| order number | name     | tab | day    | month | year | amount | carriage return |
|--------------|----------|-----|--------|-------|------|--------|-----------------|
| 3            | C Jones  |     | 3/1/88 |       |      | 10.23  |                 |
| 5            | E Bliggs |     | 5/1/88 |       |      | 17.99  |                 |

The main details (the order number, name, etc.) are inserted straight from the data file. So we just need to use the appropriate item-names for these details. To insert the tab and carriage return characters between these details, we use the same technique used in the INVOICE master – storing the characters under suitable item-names (ie. `tab` and `cr`). These characters are stored in our initial 'definitions' group of instructions at the top of the UNPAID.INV master.

We could similarly store the slash character under an item-name and insert it between the day, month and year in the same way, as we have been doing in the previous sections. However, to improve the layout of the line of instructions that will print one line of the invoice, we concatenate the `day`, `month` and `year` values into a single item `date`:

```
date=day & "/" & month & "/" & year
```

and use `date` in the instructions to insert the details into the finished document:

```
order_number:tab:name:tab:date:tab:amount:cr←
```

Notice how this has made the line much shorter and easier to follow than if we used `...tab:day:slash:month:slash:year:tab...`. You should always look out for opportunities like this as you create masters.

The final task of the program unit is to maintain the running total of unpaid sales and to move on to the next record. Adding the amount from the current record to the value stored under an item-name (`total`) is done using the instruction:

```
total=[total+amount]
```

The value of `total` is initially set to 0 at the start of the master. Moving to the next record is done with the `$+` instruction.

Notice how the comparison and the concatenation instructions within `loop` give us double quote marks within a string. This means we have to ensure that LocoMail doesn't take any of the embedded double quotes as the end of the program unit. In this case, Mail codes at the beginning and end of the program unit put all the embedded double quotes at a different level to the ones surrounding `loop`. So LocoMail is able to match the double quotes correctly:

```
loop=0(+Mail)←
#status="p*"...
..."/"..."..."
Double quotes
surrounding
loop
```

```
(-Mail)←
```

(Embedding strings within strings is explained in Chapter 10.)

Now we've set up `loop`, we need to repeat it until the end of the data file is reached. As in previous sections, this is done using the instruction:

```
#loop@name
```

which means that `loop` will be repeated until the last record has been checked (assuming you have no 'homeless' records in the data file).

The last part of our required list consists of ruled lines, a heading and the running total of all unpaid invoices. These details are inserted using LocoScript text and the LocoMail instruction `(+Mail)total(-Mail)` to insert the total.

The remaining aspect of UNPAID.INV to look at is the join between the sections we've described.

The first join is where the initial 'definitions' meets the 'fixed' text of the headings. We finish the definitions with a `(-Mail)` code to make sure the fixed details appear in the finished document.

At the end of the first piece of fixed text, we return to LocoMail instructions. The `loop` instruction `#loop@name` is followed by a `(-Mail)` code and so `loop` will be performed as if it followed this `(-Mail)` code. The `(-Mail)` at the end of the program unit means that when LocoMail finishes looping it will be inactive. So the next piece of 'fixed' text (beginning with the `(+UL)` code and tabs) will be inserted straight into the finished document.

## 13.7 Quarterly VAT totals

In Section 13.6 we described a master to print out specific records from your data file – records made out on a certain date that are unpaid. Another typical example of listing out specific records is in producing lists of all sales made within a specific quarter, for VAT purposes. For example, you may need a list of all invoices issued in the quarter commencing 6/88:

### INVOICES ISSUED IN QUARTER COMMENCING 6/88

| Invoice | Customer  | Sales date | Amount | VAT  |
|---------|-----------|------------|--------|------|
| 1       | A Brown   | 1/6/88     | 46.00  | 6.00 |
| 3       | C Jones   | 3/6/88     | 10.23  | 1.34 |
| 5       | E Bloggs  | 5/6/88     | 17.99  | 2.35 |
| 6       | F Green   | 4/7/88     | 27.25  | 3.56 |
| 8       | H Andrews | 7/7/88     | 7.43   | 1.11 |
|         | ...       |            |        |      |

### TOTALS :

£108.90      14.36

This finished document is fairly similar to the one produced in the previous section, and as you might expect the master document (TOTAL.INV) looks pretty much the same. The main differences are in the way TOTAL.INV handles the date information.

To see the TOTAL.INV in action, merge it with ALLSALES.

**IMPORTANT:** The TOTAL.INV master document performs simple VAT calculations on the VAT-inclusive figures stored in `amount`. If you are using this master for accounting purposes, you should satisfy yourself that the calculations and the nature of the information are correct for your own particular circumstances.

**(Mail)** \_\_\_\_\_

Start month = first month of quarter (1 - 12)  
Start year = "Y" and year (88-99)

Start quarter = [2 \* start year + start month]  
Start quarter = Start quarter % 4

Total = 0 ; total vat = 0  
do while : slash = "/" or "

(Mail) (Bold) (Itl) ✓  
INVOICES ISSUED TO CUSTOMER CONCERNING (Mail) (Bold)

(CUL) Invoice Customer Sales date Amount VAT (CUL) ✓

(Mail) ✓

loop: (Mail) ✓ sales date = [2 \* year + month]  
start quarter = sales date / end quarter  
date-day & slash & month & year  
vat = (amount - amount / 71.51211) ✓  
order\_number-tab-name-tab-date-tab-amount-tab-vat  
total = total amount + total vat = total vat + vat ✓

(Mail) ✓

X loop: next (Mail) (CUL) ✓

(Bold) TOTALS ++ + £ (Mail) (Bold) (Mail) (Bold)

Fetch dates Convert dates to suitable format Advance to next record

If sales record is in quarter of interest, print out details

Program unit set up

The finished document consists of details from every record where the stored sales date is within three months of the specified date. So the first job of the master is to fetch details of the quarter of interest. Next, the fixed pieces of information and text (at the top and bottom of the finished document) are inserted, which we do using LocoScript text. Finally we produce the list, using a program unit called `loop`.

We will now look at the main steps in TOTAL.INV.

## Working out the dates of interest

The first group of instructions in TOTAL.INV prompt for the start of the quarter of interest, as a month and year. Essentially, we just need the *item-name*=? type instructions; however, to compare the typed values with the stored month and year, we will need to use the typed values in arithmetic and numeric-type comparisons. So the instructions are:

```
startmonth = ?#; First month of quarter? (1 - 12)
startyear = ?#; and the year?
```

Unfortunately, we cannot check, just by simple comparison between the typed and stored details, whether a stored date is within the quarter commencing at this date. We need to make sure that a sales record dated 1/89 will be included in the list when the quarter commences in 1/88.

As you saw with the REMINDER master (Section 13.5), probably the most convenient way of comparing dates when you have to make allowances for year-ends, etc. is to express the dates as numbers. The numbers must be larger for later dates; then, it is possible to see if the sales record is between the start and end of the quarter of interest using a 'range' type comparison.

We put the typed month and year values in a suitable format by multiplying `startyear` by 12 and then adding `startmonth`. This essentially converts the year and month to the number of months since the year 00 (ie. the turn of the century):

```
startquarter=[startyear*12+startmonth]
```

The last month in the quarter is the `startquarter` value + 2, so we store this value in `endquarter`:

```
endquarter=[startquarter+2]
```

## Setting up the program unit

**The instructions within the program unit (Loop):**

- work out whether the date in the current record is in the quarter of interest
- if it is, insert the appropriate details from the current record (the name and amount), then work out the vat included in the amount, and add the amount and vat to running totals
- move on to the next record

To check whether the current record is of interest, the stored date is converted to the same format as the quarter of interest, with the instruction:

salesdate=[12\*year+month]

We then check if the date of this record is within the quarter of interest (ie. between the values stored in **startquarter** and **endquarter**), with the conditional instruction:

```
#startquarter<=salesdate<=endquarter<:...>:$.+
```

If **salesdate** is between the start and end of the quarter, the instructions within the angle brackets are carried out before moving on to the next record (the \$+ instruction). If it isn't, LocoMail ignores the instructions within the brackets and steps straight to the next record.

The instructions within the angle brackets insert the details into the finished document as follows::

- the vat element of the amount is calculated and stored under **vat** using: **vat=[amount-[amount/1.15|2]]** (Note that the value is truncated to 2 decimal places. You should satisfy yourself that this truncation is correct for your own circumstances.)
- the details from the current record are inserted into the finished version: **date=day & slash & month & slash & year**  
**order\_number:tab:name:tab:date:tab:amount:tab:vat:cr**  
(Note that we've built up **date** and used it as suggested in Section 13.6.)
- the sales amount and vat for this line are added to the running totals;

**Total** and **totalvat** must be set to zero before the list is created, so we've used the instructions **total=0:totalvat=0** in the 'definitions' part of the master.

Once the Loop program unit is set up, we can repeat it until we have checked every record using the instruction:

```
%loop@name
```

This is the same method used in the previous masters – as long as you have no nameless records in the data file, Loop will be repeated until the last record has been checked.

The last part of our required list is ruled lines, a heading and the running totals. These details can be simply inserted using LocoScript text and the LocoMail instruction `(+Mail)total:tab:totalvat(-Mail)` to insert the totals under the correct columns.

The joins between these sections of text and instructions are exactly the same as described at the end of Section 13.6.

*• If your sales records are kept in strictly chronological order, you might improve the TOTALINV master document so that it will produce a list of invoices within a given quarter, stopping only when it reaches a sales record outside the quarter. To do this, you need to change the test which controls the loop; the variable becoming zero when you first find a salesdate greater than endquarter.*

*As a further improvement, you might amend a copy of TOTALINV so that it will print out a sequence of documents, one for each quarter of a particular year, and printing one such document per pass. In this case, TOTALINV must again work through your ordered records, stopping when the first record is found from the next quarter and using a \$- instruction to hold on to this record for the next pass.*

## Chapter 14

# Enquiries system

The LocoMail master document and data file described in this chapter provide a simple 'enquiries' system. Each record in the data file is a list of information about a particular product, service, etc., finishing with a piece of descriptive text about that product. The master document is designed to ask for the particular requirements of the enquirer, then print out the details from each record in the data file that matches.

These example documents are currently in group 5 on your 'LocoMail example documents' disc. The data file and master document have been set up for an estate agency – the data file (HOUSE.EG) holds details of properties, and the master document (RESPONSE) prints out the details of all the properties within a specific price range, district, number of bedrooms, etc. However, you should find it easy to adapt the documents to suit other needs.

In the following sections, we describe how the information in the data file is used, and how and why the different techniques of Parts II and III have been used to achieve the desired results. We recommend you to study these. Once you have followed these descriptions, you will be in a better position to adapt these documents or create your own masters.

## 14.1 The data file

The design of the data file is dictated by the details which are to be stored. The sort of details which might be stored for a property are the price, location, number of bedrooms, year of construction, and a description.

The other thing you should consider when setting up the data file is that enquirers typically ask for details within ranges – eg. a price range, a minimum of 3 bedrooms, built before a certain date, etc. So if the master document is to be allowed this flexibility in the comparisons it makes, the corresponding details must be stored as numeric information.

The data file, HOUSES.EG, stores the following details about each property:

|               |   |                                                                |
|---------------|---|----------------------------------------------------------------|
| <b>area</b>   | - | area (town, city, country)                                     |
| <b>type</b>   | - | type of property (detached, semi, flat, terraced, etc)         |
| <b>beds</b>   | - | the number of bedrooms (numeric)                               |
| <b>year</b>   | - | year the property was built (numeric, and a 4-digit number)    |
| <b>price</b>  | - | price (numeric)                                                |
| <b>clause</b> | - | a description of the property, typically including the address |

The one condition that your data file must obey is that the **type** item must not be left blank in any records. This is because the master document repeats a 'program unit' of instructions (see Chapter 9) until the **type** is null, which is automatically the case when the end of the data file is reached. If any of the records have a blank **type**, the master will stop at this record!

The record pattern and separators for the data file are:

area, type, beds, year, price ↓  
clause ↓

You can of course add any other items to this record pattern, as long as the new item-names don't clash with any of these.

■ 3 2 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038 1039 1040 1

## 14.2 Producing the response

As always, what dictates the structure of the master document is the required finished document you want the master to produce. Here, our finished document is to be a list of properties meeting the requirements of a potential customer:

**Requirement profile**

|                 |               |
|-----------------|---------------|
| House type:     | Semi          |
| Seats required: | 3             |
| Age:            | any           |
| Built after:    | 1970          |
| Built before:   | 1968          |
| Price range:    | £41250-£55000 |

.....

A 3 bedroom town semi-detached built in 1966. £ 47 500  
Clark's Crescent is in the select Northenders' estate, close to schools and shops.

.....

A 3 bedroom town semi-detached built in 1977. £ 52 000  
The 3rd floor is in the bustling town of Bartonwood. New bathroom and kitchen suites: a real bargain.

There are three fundamental steps needed to produce this finished document:

- collect up the requirements of the enquirer
- print the summary of requirements
- work through every record in the data file, comparing the details with the requirements; whenever a matching record is found, print the details

The RESPONSE master document we use also consists of three parts, addressing each of these steps in turn. To see the RESPONSE master document in action, merge it with the HOUSES.EG data file.

```

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000

```

## Prompting for the requirements

The requirements of the enquirer are collected by using an initial information-gathering block of instructions of the *item-name*=? type.

The details that can be specified are simply those held in the data file – the type of property, the area, age, number of bedrooms and maximum price. Where the typed details are to be compared with numeric values in the data records, we must accept only numeric input by making the instructions `item-name=?#`. We must

also allow these requirements to be 'blank' (for instance, when there are no particular requirements for age of property): the prompts tell you what to type when there are no particular requirements:

```
(+Mail) <
findtype= ?; Type of House (just press ENTER for any) <
firstyear= ##; Constructed after (0 if not important) <
lastyear= ##; Constructed before (9999 if not important) <
findarea= ?; Location (just press ENTER for any) <
highprice= ##; Price limit? <
findbeds= ##; How many bedrooms? <
(-Mail)
```

### Laying out the requirements

The second part of the master lays these typed details out neatly, as in the required finished document.

The headings and correct spacing simply use the appropriate LocoScript text, codes, tabs characters, etc. The requirements themselves are inserted from memory using instructions of the type `(+Mail) item-name (-Mail)`.

However, we won't necessarily want to simply slot in exactly what was typed. For example, if no particular type of property was specified (ie. if just `[ENTER]` was pressed to the prompt `Type of property?`), then we want the finished document to display the word 'any' alongside the heading, rather than just an empty space. If the dates of construction are of no interest, we want to omit these lines altogether. Also, we need to calculate the lower limit of the price range as we only type an upper limit.

To allow for these possible variations in the content, we use conditional instructions. The ways of using conditionals to vary the contents of a document were described in Chapter 7.

#### • Putting in 'any'

The lines for the type of property and the area need to include either the typed response or, if just `[ENTER]` was pressed to the prompt, the word 'any'. So to insert the appropriate text alongside the heading House Type, we use:

```
#findtype="":<(-Mail) any(+Mail) ><:findtype:>
```

which says 'if `findtype` is null, insert the text 'any', otherwise insert what is stored in `findtype`'. The instruction that puts in the `findarea` works in exactly the same way.

#### • Missing out lines

The lines for the preferred years of construction are a little more complicated. If there are no particular preferences for dates (ie. the responses were 0 or 9999 to the respective prompts), then we don't need to include *any* of the line – no heading, no date and no carriage return after the details. So again, we use conditional instructions, but this time we put the whole line into the conditional instruction.

Putting whole lines or paragraphs into conditional instructions can give you spurious extra `<-s` (ie. blank lines) in your finished documents if you're not careful. The problem is keeping track of which `<-s` in your master document will be inserted into the finished document (since some act as command separators), and under what circumstances they are inserted. The technique `RESPONSE` uses is to replace all the `<-s` which may be inserted by the item-name `cr`, and to structure the master so that the `<-s` you need to make the master document readable actually act as command separators. This makes it very much easier to ensure that the carriage return at the end of any optional paragraph is handled correctly.

The conditional instructions to insert the preferred dates are therefore:

```
(+Mail) #firstyear<>0:<<
(-Mail) Built after:> (+Mail) firstyear:cr:><
(-Mail)
```

'if firstyear is not zero, insert the line of details and a carriage return'.

```
(+Mail) #lastyear<>9999<
(-Mail) Built before:> (+Mail) lastyear:cr:><
(-Mail)
```

'if lastyear is not 9999, insert the line of details and a carriage return'.

#### • Calculating the price range

The last line of the summary of requirements gives the acceptable price range. The only value requested when prompting for information was the maximum price limit, so we calculate a suitable lower price limit from this using LocoMail's arithmetic instructions.

In this case, we want the minimum price to be 75% of the maximum figure. We could calculate this with the instruction:

```
lowprice=[highprice/4*3]
```

However, simply dividing `highprice` by 4 then multiplying by 3 will give us a `lowprice` value that is exactly 75% of the `highprice` – perhaps even including pence! House prices are typically given to the nearest £100 or so, so we need to produce a suitably rounded figure, and to do this, we need slightly more sophisticated arithmetic. For example, we can calculate a `lowprice` which is 75% of `highprice`, but rounded to the nearest £750, by using:

```
lowprice=[(highprice/1000|0)*750]
```

Within the inner set of square brackets, `highprice` is divided by 1000 and the result rounded to 0 decimal places – so the result is a whole number (eg. 55100/1000 becomes 55, or 37750/1000 becomes 38). When we multiply this number with 750 (to give the required overall scaling of 75%), the result of this multiplication can only be a whole multiple of 750.

This is the calculation `RESPONSE.EG` uses. By changing how we express the calculation, and the format we use, we can change what we round the result to. For example, to round to the nearest £250 we would use `[highprice*3/1000|0]*250`. Notice how in each case the number we are rounding to appears at the end of the calculation, multiplying a number that we round to 0 decimal places.

## Producing a list of suitable properties

The final part of the master document produces the list of all records in the data file that meet these requirements. To produce the list, LocoMail must check whether the current record matches or not, print the details if it does, then move on to the next record and repeat the steps.

Repetitive 'looping' tasks like this are ideally suited to using a program unit. The instructions in the program unit carry out the checking, printing and advancing operations; we then use a loop instruction to make LocoMail repeat the program unit as often as necessary (in this case, until all the records have been checked). The program unit is **print**.

### • Checking for matches

The first task is to check whether the typed requirements match the details stored in the current record. We need to check whether:

- area matches the required area (**area=findarea**)
  - type matches the required type of property (**type=findtype**)
  - price is between the min. and max. values (**lowprice<=price<=highprice**)
  - there are at least enough bedrooms (**beds>=findbeds**)
  - the property was built in the right period (**firstyear<=year<=lastyear**)
- The only problem with straightforward comparisons like these is that they don't allow any 'slack' in the comparison. For instance, if the required property type was entered as **semi**, then this won't match when the **type** item holds **semi-detached**. Similarly, if you had just pressed **[ENTER]** to indicate any type or area, the comparisons will be false when they should be true. So to add some flexibility to the comparisons for type of property and area, we add 'wildcard' characters to the typed details:

```
searchtype=findtype & "*"
searcharea=findarea & "**"
```

Now if you just press **[ENTER]**, the comparison **type=searchtype** will effectively be **type="\*\*"**. As \* represents any string of characters, this comparison will be true however the property type is described. Similarly, if you typed **semi**, or even just **s**, you will get the right result when the **type** item holds **semi-detached**.

Also, notice that typing 0 for **firstyear** and/or 9999 for **lastyear** (as described in the prompts) gives the right results because we use a 'range' comparison.

Because all the details must match, these individual comparisons are combined into a single comparison using **AND**:

```
area=searcharea AND type=searchtype AND
lowprice<=price<=highprice AND beds>=findbeds AND
firstyear<=year<=lastyear
```

which is only true if all these conditions are met. In that case, we need to print out details then move on to the next record. If any aren't met, then we simply move straight to the next record.

The complete instruction to do this is essentially:

```
#conditions:<...print details...>: $+
```

with the **\$+** instruction being used to advance to the next record. The instructions within the **<>** angle brackets are performed only when all the details match - and their job is to print out the details of the current record neatly:

### • Printing the details

```
A 3 bedroom town semi-detached built in 1986: £ 47,500
1 Clerk's Crescent is in the select Northenders' estate, close to schools
```

The first line of these details is built up of a combination of details from the data record, and pieces of LocoScript text which contain all the required spaces. The line is initially created and stored using a concatenation type instruction:

```
description="A " & beds & " bedroom " & area & " "
& type & " built in " & year & ": £" & price
```

The second part of the property description is simply the information stored in **clause**, starting on a new line. To print the new details, we use:

```
cr:description:cr:clause:cr
```

The **crs** simply insert carriage return characters, giving a blank line above the description and finishing the description and clause.

### • Repeating the program unit

To copy every record in the data file, we need to ensure that **print** is repeated for all the records. We probably don't know how many records there are in the data file, but this is not a problem. We can simply control how often the program unit is performed by using the fact that when LocoMail reaches the end of the data file, it automatically sets all values it takes from the data file to null. So to repeat **print** for all records, we use the instruction:

```
$print@type
```

This says 'perform the program unit **print** until **type** is zero or null'. That's why we said earlier that the **type** item shouldn't be blank in any records - if LocoMail meets such a record it will stop searching for matches.

- The remaining aspect of **RESPONSE** to look at is the joins between the sections we've described - an important point to consider in any LocoMail document.

The first join to think about is the one between the initial block of instructions (where we collect information, add wildcards, calculate **lowprice** and store the carriage return character), and the part laying out the summary of requirements. The latter comprises fixed text, so the initial block of definitions and comments finishes with a **(-Mail)** code.

At each point where LocoMail instructions appear in the fixed text, we use **(+Mail)** and **(Mail)** codes around the instruction.

After the last of the fixed details, the rest of the master document is made up entirely of LocoMail instructions. So there's a **(+Mail)** code after the fixed text.

However, notice how including the double quote marks of the concatenation instruction within `print` gives us strings (picked out by the double quotes) within a string. This means we have to ensure that LocoMail doesn't take any of the embedded double quotes as the end of the program unit. In this case, LocoMail interprets the double quotes correctly because of the `(+Mail)` at the start of the program unit and the `(-Mail)` at the end of the program unit. These codes put the embedded double quotes at a different level to the ones surrounding `print`. LocoMail is therefore able to match the double quotes correctly:

```
print="((+Mail) ↵
#area=...
... "A" ... " bedroom " ...
$+↵
(-Mail) " " ↵
```

*Double quotes  
surrounding print*

(Embedding strings within strings is explained in Chapter 10.)

## Chapter 15

# Continuous labels

One of the most common uses of LocoMail is in producing address labels for mailshots. Labels tend to be supplied on special continuous labels stationery, which has either a single label across the width of the paper ('1-across'), or more than one ('n-across'). You need a different type of LocoMail master document for each of these two cases; a very simple master will handle the 1-across case, but a more complicated master is necessary for n-across labels.

In addition to including the LocoMail instructions in your master document, you will probably have to set up the document to use the special continuous stationery rather than one of the standard Paper Types.

In this chapter we describe two LocoMail master documents, one for each type of labels stationery. Section 15.1 describes the data file used by both master documents; Section 15.2 then describes the simple master document suitable for 1-across labels, and how you set up the document for a special type of paper. (You can use the same method to set up any of your masters for a particular type of paper.) Section 15.3 describes the more complicated master that will print labels on n-across stationery.

The data file and the n-across labels master described in this chapter are the documents LABELDATEG and LABELDOCEG respectively, in group 6 of your LocoMail example documents' disc. The simple master document has not been supplied – if you want, you can create this master as you work through Section 15.2.

## 15.1 The data file

In creating the data file of names and addresses, you need to think of the way you want to use the information in the finished document. You can, for example, store the whole of the address under the single item-name **address** – but this will only allow you to include the details in your finished documents in one way. Or you can use a separate item-name for different parts of the address and include the details in your masters as and when you want. You just need to ensure that the record pattern defines the item-names you want.

To give yourself full control over how you can use the information, you might split the address into components like Room, Building, Street, Village, Town, etc. Using a separate item-name for each component of the address also allows you to choose ←s as the separators, giving you a record pattern like:

Name ↵  
Room ↵  
Building ↵  
Street ↵  
Village ↵  
Town ↵  
County ↵  
Postcode ↵  
Country ↵

urse, each of the real names  
pattern. For example, Willi  
William Smith ↵  
Flat 10 ↵  
West Lodge ↵  
223 The Broadway ↵  
Ameswell ↵  
Gloucester ↵  
Glos ↵  
GL23 7HE ↵  
England ↵

Of course, each of the real names and addresses in the data file has to follow this record pattern. For example, William Smith's record might look like:

William Smith↵  
Flat 10↵  
West Lodge↵  
223 The Broadway↵  
Ameswell↵  
Gloucester↵  
Glos↵  
GL23 7HE↵  
England↵

An example of this data file is stored in the LABELS group on your 'LocoMail example documents' disc, as LABELDATE.G.

If, for any reason, you want to leave the name blank in any record, remember to put a space character in the `name` item. You need to do this because LocoMail always ignores any returns in a record before the first item of information. Putting in the space effectively tells LocoMail 'the name is blank in this record'.

## 15.2 Setting up a master for 1-across labels

All a master document that produces 1-across labels has to do, is fetch each item of information from the data file and arrange it on the page – putting in spaces, commas and carriage returns between items where necessary. So you just need a number of the simple (+Mail) *item-name* (-Mail) instructions, within a framework of the LocoScript spacing characters to give it the layout you want.

**So to get labels like:**

William Smith  
Flat 10, West Lodge  
223 The Broadway  
Ameswell  
Gloucester  
Glos  
GL23 7HE  
England

**You simply need a master document like:**

```
(+Mail) Name (-Mail) ↵
(+Mail) Room (-Mail) ↵
(+Mail) Street (-Mail) ↵
(+Mail) Village (-Mail) ↵
(+Mail) Town (-Mail) ↵
(+Mail) County (-Mail) ↵
(+Mail) Postcode (-Mail) ↵
(+Mail) Country (-Mail) ↵
(+Mail) Building (-Mail) ↵
```

Note that all the returns, etc. are outside the LocoMail instructions. Ensuring that these are correct is just as important as getting the LocoMail instructions right!

This master isn't supplied on your example documents disc – so if you need such a master, you'll need to type it in. As you do this, remember to think about the Paper Type your master document is set up for.

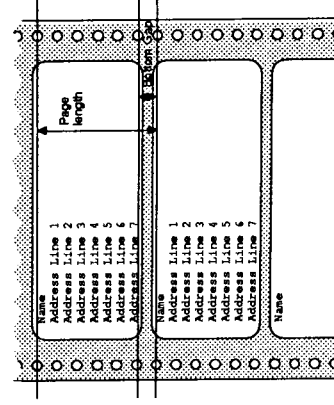
When you create this master document, it will be automatically set up for the same paper as the corresponding TEMPLATE.STD document. Unless the template has already been set up for your labels stationery, you will need to set up the type of paper to be used. How to do this is described below.

## Setting the paper type

As with any LocoScript document, each separate document (ie. each label) that LocoMail produces will be printed on a separate page, and the distance the printer moves to the top of the next page before printing the next label depends on the details set up in the Current Paper Type for the printer. So to make sure that each label is printed in the right place on the label, you need to set up the details of the new Paper Type for the master document (see Session 19 of the LocoScript 2 User Guide, or Session 20 of the PCW9512 User Guide).

The important details you need to know when setting up a document for a particular type of paper are:

- the repeat distance (ie. the distance from the top of one label to the top of the next, so that LocoScript will move on the correct distance)
- the bottom gap (ie. the size of the gap between consecutive labels, so that LocoScript doesn't print anything on the backing paper): because you can position the labels exactly where you want, printing to begin, we set the top gap to zero
- the left offset (ie. the distance from the natural start position of the printhead, so that nothing will be printed off the side of the labels)



**Note:** The repeat distance and the bottom gap are both measured in standard lines (%th of an inch); the left offset is measured in 1/60ths of an inch.

For the purpose of this example, we shall set up the master document to print correctly on 1½" continuous labels, with a bottom gap of 1 line (ie. ¼" – to clear the backing paper), and a left offset of 10 (ie. 1" from the natural start position to the printhead) to the correct position on label). You can use the same technique to set up the document for any size of stationery (both continuous and single sheet), substituting the gap and offset values appropriate for the paper you intend to use.

Before you set up the details for your own paper type, however, you will need to determine a suitable value for the left offset. This may require a little trial and error until the position is exactly right on the printed version. Practise by trying out different offsets in Direct Printing – with the labels stationery in the printer, set the offset manually (using the Left Offset menu in Printer Control State) and print an H.

The steps to set up the paper type for the master document are as follows:

- 1 If the master document isn't currently displayed on the screen, display it. Then press **[F1]** to display the Actions menu, select **Document setup** and press **[ENTER]**.
- 2 Press **[F5]** to display the Page menu.
- 3 Select **Paper Type** (it should already be highlighted) and press **[ENTER]**.

The list of standard paper types is displayed. To add your new paper to this list, choose the paper type which is nearest to your labels (choose **11" continuous** if this is the only type of continuous paper you've got; it is always best to start from the same fundamental type of paper) and press **[E]**.

- 4 Move the Menu cursor down to **Show Paper Type** and press **[ENTER]**. The details of the current Paper Type will now be displayed.

Press **[E]** to clear away the old name. Type in a new name for the labels stationery (eg. **1½" labels**), then change the other values in the menu to match the new paper as below. Each time you change a value on the menu, press **[ENTER]**.

The **Height** you will require is the repeat distance, in standard 6-pitch lines. As the repeat distance is 1½" and there are 6 lines per inch, the **Height** you need is 1½ times 6 = 9.

The **Top gap** and **Bottom gap** need to be set in such a way that no line of an address is printed on the backing paper. For labels, you might set:

- **Top gap** of 0 (as you can position the labels in the printer exactly where the first label is to begin)
  - **Bottom gap** of 1 (as you can see the backing paper at the last line of each 'page', and you must avoid printing on this backing paper)
- Set the **Left offset**, for example, to 10 (ie. printing to start 1" from the leftmost position of the printhead).

| Paper Labels          |    |
|-----------------------|----|
| Single sheet          |    |
| Continuous stationery | ✓  |
| Height                | 9  |
| Left offset           | 10 |
| Top gap               | 0  |
| Bottom gap            | 1  |
| Ignore paper sensor   |    |

When you have set these values, press **[ENTER]** to return to the Paper Type menu. You will see your new paper type on the list of those available, with a ? beside it. The ? indicates that the Paper Type is special to this document.

Move the cursor to **Use Paper Type** and press **[ENTER]**.

You must now set the Page Layout details for your new paper. If an Alert message appears telling you that the page does not fit on the paper, select **Alter Page Layout** and press **[ENTER]**. If no message appears, select the **Page Layout** option in the Page menu and press **[ENTER]**.

What you set in the Page Layout menu is the **Header zone** and the **Footer zone**. For straightforward labels, no header or footer is required (in fact, there isn't room for them). So set both of these values to 0, pressing **[ENTER]** after you type each value. This gives you a **Page body** of 8 lines.

When you've set the details on the Page layout menu, press **[ENTER]** to return to the Page menu.

Once you have set up the *Paper type and Page layout*, you don't need the *Page menu any more*, so press **[EXIT]** to move down to the **Exit** option and press **[ENTER]**.

- 5 Delete any header or footer text on the Pagination screen if any text is shown (since you have set the header and footer zones to zero).

Press **[EXIT]** to leave Document setup.

*The master document is now set up to print on 1½" continuous labels stationery. Merge the document with the LABELDATEG data file, and check that the labels are printed at the correct positions – with each name on the top line of a new label, and no details printed on the backing paper.*

**IMPORTANT:** If you intend to use the new paper type with any other documents, you should store it as one of your 'standard' paper types in the SETTINGS.STD file on your Start-of-day disc. Now is the ideal time to do this because, provided you do this straight away, these paper type details will be the ones displayed if you select the Create new Paper Type option in the Settings menu. How to do this is described in Session 19 of your LocoScript 2 User Guide (or Session 20 of your PCW9512 User Guide).

### 15.3 A master document for n-across labels

Labels are often supplied in groups of two or three across the backing paper:

William Smith  
Flat 10 West Lodge  
223 The Broadway  
Crimeswell  
Gloucester  
Glos  
GL23 7HE  
England

To use this type of stationery, you need a rather more complex master document. The important thing to remember, as you look at the labels, is that when LocoScript is printing the finished labels it cannot print all of one label, then rewind the paper and move across to print the second label, then move back up and across to print the third. Instead, the printer has to print all three names on one line, with tabs between. Then it can print the first line of each address with tabs between, then the second line, and so on.

So the job of the master is to collect up the details in groups of 3 (or however many labels there are across the paper), and insert tabs between the details. The labels can only be printed when these lines of details are complete

What the master needs is a number of item-names; each one will be used to store a line of information. To collect the details from a number of records and store the details under the item-names, the master uses a program unit – repeating this program unit three times (in this case) stores the right details. When the lines of information are complete, the labels can be printed.

The master document is stored in LABELDOC.EG, and is shown opposite.

The first group of instructions in the master document sets up the item-names that will hold the details, and makes them blank:

```
(+Mail) Nameline="" ↵
Addline1="" : Addline2="" : Addline3="" ↵
Addline4="" : Addline5="" : Addline6="" : Addline7="" ↵
```

The next two instructions store the characters that will be used to get the right appearance for the finished labels:

**Tab="→       ":Spaces="↵**

The next section of this master is the part that collects up the three names and addresses from the records and stores them under the item-names. It essentially consists of a program unit (called **Loop**) which is designed to collect up the details from a record, and a pair of instructions designed to make sure that **Loop** is carried out three times.

**Making the 'line'**  
items initially blank

**Program unit,  
adding details from  
the file to each line**

**Printing out the completed lines**

The instructions that will carry out a program unit three times are:

**Loopcount=3** ↴

```
@Loopcount&Loop
```

**Loopcount** is used to ensure that **Loop** is only executed three times. What actually happens is that the LocoMail instructions stored as the program unit **Loop** are obeyed while **Loopcount** is non-zero. So to make sure that we only repeat **Loop** three times, **Loop** must include an instruction that makes **Loopcount** count down 3, 2, 1, 0. (See Chapter 9.)

The instructions stored as **Loop** are:

```
(+Mail) Nameline=Nameline & Name & Tab↵
Addline1=Addline1 & Rm & Spaces & Building & Tab↵
Addline2=Addline2 & Street & Tab↵
Addline3=Addline3 & Village & Tab↵
Addline4=Addline4 & Town & Tab↵
Addline5=Addline5 & County & Tab↵
Addline6=Addline6 & Postcode & Tab↵
Addline7=Addline7 & Country & Tab↵
loopcount=[loopcount-1]↵
#loopcount<0:<:$+:>↵
#name="":<:loopcount=0:>↵
(-Mail)
```

and as you can see, these commands are grouped into a program unit by the `Loop="..."` command.

The first of these adds the name from the current record to the righthand end of whatever is stored in `nameline`, then adds a tab character to space out ready for the next name. The second instruction does a similar operation on the first address line, but it also adds some spaces between the room number and the building name before finishing with the tab character. The other Addlines work similarly.

The remaining lines in `Loop` set up the information necessary to make sure that the correct amount of details are collected – ie., that `Loopcount` becomes zero when loop is repeated 3 times. The first of these lines subtracts one from the value of the variable `Loopcount` – so each time `Loop` is performed, `Loopcount` is decremented. The next instruction tells LocoMail to move on to the next record only if `Loopcount` is not zero – because `Loopcount` is zero when you have collected up the three sets of details. The final thing you need to consider while collecting up details is what happens when you reach the end of the data file. When LocoMail uses the last record, it sets the value of all the *item-names* to null. So at this point, you want to stop repeating the program unit. The instruction that will do this is the `#name="":<:Loopcount=0:>` instruction.

`Loop` will therefore collect up details from records, and repeating this `Loop` three times therefore has the required effect of storing three names in `nameline`, three rooms and buildings in `addline1`, etc.

Now all that is left is to print out the lines. This is done in much the same way as the master in Section 15.2 printed the lines of the label – a combination of LocoMail instructions and LocoScript characters as follows:

```
(+Mail)Nameline (-Mail) ␣
(+Mail)Addline1 (-Mail) ␣
(+Mail)Addline2 (-Mail) ␣
(+Mail)Addline3 (-Mail) ␣
(+Mail)Addline4 (-Mail) ␣
(+Mail)Addline5 (-Mail) ␣
(+Mail)Addline6 (-Mail) ␣
(+Mail)Addline7 (-Mail)
```

(Note that the layout you're using in the master document must contain the suitable tab positions to make sure the details are laid out correctly on the paper.)

So the master document will now do what you want – collect up the details from three records and print the details correctly across the labels.

Note that you can easily change this master to use 2-across labels, say, by changing the number of times the master carries out the program unit `Loop`. All you have to do is set `Loopcount` to the number of labels across the paper, and adjust the positions of the tabs in the document layout.

**Note:** This master document will not work with a data file in which addresses are stored under a single item-name. It has to handle the individual lines of the address separately. To apply it to such a list of names and addresses, you have to at least change the record pattern to give a separate item-name for each line.

## Appendix I

# Technical description of LocoMail

This appendix gives the formal technical description of all the various types of LocoMail instruction and of the structure of the data files from which LocoMail can take information. References are given to the relevant sections of this User Guide.

## Conventions and definitions

The following conventions are used throughout:

|                                   |                                                                                                                        |
|-----------------------------------|------------------------------------------------------------------------------------------------------------------------|
| <b>*</b> , <b>&lt;</b> , <b>:</b> | an element of the syntax that is to be included at this point                                                          |
| <i>placeholder</i>                | italicised text represents a description of the item that is to be inserted at this point, rather than the item itself |
| <i>[option]</i>                   | italicised square brackets represent an element that can be included or omitted                                        |
| <b>return</b>                     | in LocoScript documents is a carriage return (↵)<br>in ASCII files is a CR/LF or LF/CR                                 |
| <b>tab</b>                        | in LocoScript documents is a tab (→) or an indent tab (⇨)<br>in ASCII files is a TAB                                   |
| <b>form feed</b>                  | in LocoScript documents is an end-of-page character (⇩)<br>in ASCII files is an FF                                     |
| <b>unit</b>                       | in LocoScript documents is a Unit code (Unitr)<br>it cannot appear in ASCII files                                      |

## 1.1 LocoMail data files (Chapter 2)

LocoMail data files may be either LocoScript documents or ASCII files. Data files are treated by LocoMail as a sequence of records, each divided into named items. As far as LocoScript and other programs are concerned, however, data files have no special properties, and can be created and changed in the usual ways.

## The Record pattern

At the beginning of a data file LocoMail expects to find a 'record pattern' which defines the layout of records in the file. The record pattern takes the form:

*[return...][item-name separator][item-name separator...][terminator]*

where:

*item-name* is any combination of latin alphabetic characters and numeric characters (starting with an alphabetic character) giving the name of a data item. Greek and cyrillic characters cannot be used. Accented characters and ß are treated as alphabetic. Underscore is treated as numeric. Upper and lower case characters are not distinguished. Only the first sixteen characters of an *item-name* are significant.

*separator* is any printing character which cannot appear in an *item-name*  
or: **return**  
or: **return return**  
or: *end-of-record*

*terminator* is: **/return**  
or: *end-of-record*  
*end-of-record* is: **form feed**  
or: **unit**  
or: **end-of-file**

### Special rules

Spaces and tabs around objects in the above syntax are not significant.

In the record pattern all codes are ignored.

A *separator* of *end-of-record* acts as a *terminator*.

The names **AND**, **OR** and **NOT** are reserved and may not be used as *item-names*.

### Notes

**/** can be used as a *separator* in the normal way; it will only be recognised as a *terminator* after the *separator* has been read.

If the pattern ends with an *end-of-file* then the user will be prompted to select the actual data file before the Merge proceeds. This allows the use of existing data without having to alter it to add the record pattern.

## The data records

The records which follow the record pattern have a similar form, but where the pattern has *item-names* the records have the item values. An item value is any sequence of characters or codes followed by either the defined *separator* for the item or by *end-of-record*, viz:

*[return...][value separator...]*

### Notes

An item value which is to be terminated by the **return** *separator* is not terminated by a single **return**. For two returns to be taken as a *separator*, there must not be any other character or code between them.

Leading returns are ignored at the start of a record, but are significant between items (they start the next item's value).

Leading tabs and spaces (which occur before an item's value) are ignored. Tabs and spaces which occur within the item or before the *separator* form part of the item's value. The *separator* never forms part of the item. An item's value may contain codes as well as characters and format effectors.

**Form feed, unit and end-of-file** terminate records explicitly and unconditionally.

Note that in the absence of an explicit terminator, a data record ends when the number of data items matches the number of *item-names* in the record pattern.

## 1.2 Master Documents (Chapter 3)

A master document is a standard LocoScript document, with LocoMail commands embedded in it. LocoMail commands are preceded by a **(+Mail)** code and end in a **(-Mail)** code. Between the codes is ordinary text, which has meaning to LocoMail but has no special significance when the document is edited. The **(+Mail)** **(-Mail)** codes have the effect of **(+Rev)** **(-Rev)**, so that LocoMail commands stand out.

LocoMail sections of the master document take the form:

**(+Mail)** *[[statement]separator...]* *[statement]* **(-Mail)**  
where

*statement* is: *[command]; comment*

*command* is one of the LocoMail commands described below

*comment* is any number of codes and characters starting with a semicolon, ending at the end of document, or at the next **return** or **(-Mail)** code

*separator* is: colon

or: **return**

or: **(-Mail)** any characters and codes **(+Mail)**

### Notes

Between **(+Mail)** and **(-Mail)** all LocoScript codes are ignored. Spaces and tabs may be freely used between items to improve readability, except in *comments* and *strings* (defined below).

When a master document is processed by LocoMail the commands within it are obeyed, producing a result document. This operation is called a 'pass'. Each time LocoMail is used there may be several passes down the master. Each pass starts by loading a new copy of the original document.

## The commands

**Note:** commands will be described as if they always occurred between their own (+Mail) and (-Mail) codes. This is for convenience only, as is clear from the structure of a LocoMail section of document, which has just been described.

(+Mail) ? [#] *item-name* [: *comment*] (-Mail)

Take keyboard input and insert it into the result document. This command is performed on every pass through the master document. If the # is present the input must be a valid number. If the *item-name* is present then the input is also stored under that name. (Part II, Section 3.3.2.)

(+Mail) ! ? [#] *item-name* [: *comment*] (-Mail)

Take keyboard input and insert it into the result document. The keyboard input is also stored under the *item-name* given. This command is ignored if there is already information stored under the name. If the # is present the input must be a valid number. (Part II, Section 3.3.2.)

When data is taken from the keyboard the cursor is positioned immediately after the command, with only the prompt visible to guide the operator. The optional *comment* field is used as the prompt text. If the *comment* is not present, a prompt of the data *item-name* (upshifted and to a maximum of 16 characters) will be used instead. Upon completing the input the prompt is removed.

? may also be used in some of the following commands, as it is a form of *value*. In such a case it will prompt in exactly the same way as just described, but the characters typed will not be left in the result document.

(+Mail) *item-name* [: *comment*] (-Mail)

Insert the information stored under *item-name* into the result document. (Section 3.3.1.)

(+Mail) *item-name* = *value* [: *comment*] (-Mail)

The *value* is stored under the given name. (Section 3.3.3.)

(+Mail) !*item-name* = *value* [: *comment*] (-Mail)

If there is already information stored under *item-name* then this command is ignored. Otherwise, the *value* is stored under the given name. (Section 3.3.2.)

(+Mail) +*item-name* [: *comment*] (-Mail)

The information stored under *item-name* must be a valid number. Its value is incremented by one and then inserted into the result document. (Section 8.2.)

(+Mail) *item-name*+ [: *comment*] (-Mail)

The information stored under *item-name* must be a valid number. It is inserted into the result document and its value is then incremented by one. (Section 8.2.)

(+Mail) -*item-name* [: *comment*] (-Mail)

The information stored under *item-name* must be a valid number. Its value is decremented by one and then inserted into the result document. (Section 8.2.)

(+Mail) *item-name*- [: *comment*] (-Mail)

The information stored under *item-name* must be a valid number. It is inserted into the result document and its value is then decremented by one. (Section 8.2.)

(+Mail) [*expression*] [*format*] (-Mail)

The *expression* is evaluated, formatted and then inserted into the result document. (Chapter 8.)

(+Mail) \* [: *comment*] (-Mail)

Discard the current result document immediately. (Chapter 6.)

(+Mail) \$+ [: *comment*] (-Mail)

Fetch the next record immediately from the data document, re-assigning all datafile record pattern items accordingly. If merging with a LocoFile datafile, the next record is the next in the current index. If there are no more records then all items are set null. (Section 6.2.)

(+Mail) \$- [: *comment*] (-Mail)

Do not fetch a new record at the start of the next pass. All datafile record pattern items will start the next pass set to their current values. (Section 6.2.)

(+Mail) \$= *value* [: *comment*] (-Mail)

Change the current index to the one specified, which must be set up in the current datafile. If *value* is null, the record number index is used. The command also moves to the first record in the specified index unless the index specified is already the current index, in which case the command does nothing. In the absence of any \$= commands, the record number index is used. In Fill mode the command is ignored. (See LocoFile User Guide.)

(+Mail) \$\$ *value* [: *comment*] (-Mail)

Using the current index, move to the first record whose key has the same or a greater *value*. If no such record exists, move to the end of the file and set all items to null. If the first *value* matches the main key precisely, then the second *value* is used for comparison against any sub-key. In Fill mode the command is ignored. (See LocoFile User Guide.)

(+Mail) #*condition* :< *separator*[[*statement*] *separator*...]

/>#*condition* :< *separator*[[*statement*] *separator*...]

/>< *separator*[[*statement*] *separator*...]

> (-Mail)

If the first *condition* is true, then the first portion of commands and text will be actioned up to the next corresponding > command; the rest of the command will be discarded from the document.

If the first *condition* is false, then the first portion of commands and text will be discarded up to the corresponding > command; the next *condition* (if any) will then be evaluated. If it is true the following portion of commands and text will be actioned up to the next > command; the rest of the command will be discarded from the document.

The process of discarding and evaluating continues until a >< command (if any) is met. If all *conditions* have been false, then the commands and text following this are actioned.

This provides a form of IF *x* THEN.../ELSE IF.../ELSE.../ENDIF structure. This structure may be nested. (Chapter 7.)

**(+Mail) &value[&item-name] [ ; comment] (-Mail)**

When this command is encountered LocoMail reads up to and including the command termination (including any comment): ie. it reads up to the next **(-Mail)** or **return**, or (in the absence of a comment) to the next colon or to the end of document. The source of command fetching is then changed and the contents of *value* are executed as if they were present in the master document. If no *item-name* is specified or the specified *item-name* has been set to a null string or has the numeric value zero then when *value* is exhausted the command is complete and LocoMail returns to fetching from after the original **&** command. Otherwise LocoMail starts fetching from the beginning of *value* again. (Chapter 9.)

**(+Mail) &item-name&value [ ; comment] (-Mail)**

This command works in exactly the same way as the previous one, except that if the *item-name* is null or zero then *value* is never executed at all. (Chapter 9.)

**In the above:**

|                  |                                                                                                                                                    |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>condition</i> | is: <i>logatom</i> [ <i>logop</i> <i>logatom</i> ...]                                                                                              |
| <i>logatom</i>   | is: <i>condition</i><br>or: <b>NOT</b> ( <i>condition</i> )<br>or: ( <i>logatom</i> )<br>or: [ <i>value relation</i> ] <i>value relation value</i> |
| <i>logop</i>     | is: <b>AND</b><br>or: <b>OR</b>                                                                                                                    |
| <i>relation</i>  | is any one of: < <= << = => > >= >> ≥ ≠ ≤                                                                                                          |
| <i>value</i>     | is: ?<br>or: <i>number</i><br>or: <i>chatom</i> [ <b>&amp;</b> <i>chatom</i> ...]                                                                  |
| <i>number</i>    | is: [ <i>sign</i> ] <i>digits</i> [ . [ <i>digits</i> ]]<br>or: [ <i>sign</i> ] <i>digits</i> . <i>digit</i>                                       |
| <i>chatom</i>    | is: <i>item-name</i><br>or: <i>quoted-string</i><br>or: [ <i>expression</i> ] [ <i>format</i> ]                                                    |

*quoted-string* is: "characters and codes"

*item-name* is any combination of latin alphabetic characters and numeric characters (starting with an alphabetic character) giving the name of a data item. Greek and cyrillic characters cannot be used. Accented characters and German Scharfes S (ß) are treated as alphabetic. Underscore is treated as numeric. Upper and lower case characters are not distinguished. Only the first sixteen characters of an *item-name* are significant.

|                   |                                                                                                                                                                                                                                                                                                     |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>expression</i> | is: <i>numatom</i> [ <i>numop</i> <i>numatom</i> ...]                                                                                                                                                                                                                                               |
| <i>numatom</i>    | is: + <i>numatom</i> numeric unary plus<br>or: - <i>numatom</i> numeric unary minus<br>or: [ <i>expression</i> ] [ <i>format</i> ]<br>or: <i>number</i><br>or: <i>item-name</i>                                                                                                                     |
| <i>numop</i>      | is: +      numeric addition<br>or: -      numeric subtraction<br>or: *      numeric multiply<br>or: /      numeric divide                                                                                                                                                                           |
| <i>format</i>     | is: [ <i>sign</i> ][ <i>thousand</i> ][ <i>point</i> ][ <i>decimals</i> ][ <i>truncate</i> ]                                                                                                                                                                                                        |
| <i>sign</i>       | is: +    to precede positive numbers with a plus.<br>or: -    to precede positive numbers with a space.                                                                                                                                                                                             |
| <i>point</i>      | is: .    decimal point<br>or: ,    decimal comma                                                                                                                                                                                                                                                    |
| <i>thousand</i>   | If omitted then the current layout's decimal marker is used<br>is: /    if commas (or points) are wanted as thousands separators.<br>or: \    if spaces are wanted as thousands separators.                                                                                                         |
| <i>decimals</i>   | If omitted then there are no thousands separators.<br>is: <i>digit</i> (in the range 0...9)                                                                                                                                                                                                         |
| <i>truncate</i>   | If omitted then the maximum number of decimal places is used of any operand in the expression, or the maximum number of decimal places to avoid losing precision (whichever is the higher).<br>is: !    to truncate to the given number of decimal places.<br>If omitted then the value is rounded. |

**Notes**

In Merge it is an error to use an *item-name* (as a *value*) which has not yet had any information assigned to it. This does not mean that null information is unacceptable (such as may occur when a field has no characters in it), but rather that undefined *item-names* are rejected. This is intended to trap spelling errors and some forms of logic error within the master document.

In Fill it is not an error to use an undefined *item-name* (except as the control variable for an **&** or **@** loop). If an undefined *item-name* is used in an expression, or elsewhere, the user will be prompted to supply the information from the keyboard (just as if a ? command had been met for the name). This information will then be stored under the *item-name*. The name will thereafter be treated exactly as if it had occurred in a pattern (thus it cannot be re-assigned to

and the information will be discarded at the end of the pass). This allows a document designed for Merge to be used (or just tested) in Fill. If the undefined *item-name* is immediately followed by a prompt then this is used, otherwise (as in many cases the syntax will not allow a prompt) the name is used instead.

The information stored under an *item-name* is unaltered at the end of pass except when the name occurs in the datafile record pattern (when the information changes to reflect the new record) or if the information was placed there by a *?item-name* command (when the name will be completely forgotten). Note that copies of datafile pattern items or *?item-name* items (created by *item-name* = commands) will not be altered at the end of the pass.

It is an error to attempt to change the contents of a name defined in the data file record pattern (by *+item-name*, *item-name*= or any other command).

Numbers are processed internally using fixed point arithmetic, with up to 18 digits before the decimal point, and up to 9 digits after the decimal point.

'Decimal points' in number constants will be commas if the master document requests this. The other character (of point and comma) will be assumed to be a thousands separator and will be ignored.

A quoted string ("...") may contain any LocoScript characters or codes. In order to allow it to contain LocoMail commands (as used in advanced LocoMail programming techniques) it may contain (+Mail)...(-Mail) or (-Mail)...(+Mail) sections within which a double quote mark will not count as a terminator. (Chapter 10.)

The form *value relation value* compares the first and second values, then the second and third values and yields true only if both yield true. For example, the comparison  $-5 < x \leq 6$  is true if  $x$  is greater than  $-5$  and if  $x$  is less than or equal to  $6$ .

When comparing named items taken from the data document LocoMail will compare them as numbers, provided both items are of suitable form, otherwise they will be compared as strings. If an explicit number was defined in the master document then the other side of the comparison must be a number.

Only the first 100 characters are significant in string comparisons.

When comparing text strings:

- only equality and inequality operators are permitted.
- upper and lower case characters are treated as equal.
- any number of the 'white space' characters (space, tab and return) are treated as one space. This avoids problems with minor layout differences.

The wildcards ? and \* may be used in strings which originally came from the master document or the keyboard (they are treated as normal characters in strings which originally came from the data document). The character ? will match any character, the character \* will match any number of characters. It is an error to have wildcards on both sides of a string comparison. (Chapter 6.)

The & command concatenates the items it links. (Chapter 8.)

## Appendix II

# The LocoMail Master disc

The following lists the files and documents supplied on the LocoMail Master disc and explains their use.

### Group 0 - LocoMail

On the LocoMail Master disc, this group contains:

LOCOMAILJOY The LocoMail program  
(plus any 'Read-me' documents supplied on the disc.)

On the LocoMail Examples Master disc, this group is empty.

**Note:** LOCOMAILJOY must be present on the Start-of-day disc you start up from when you want to use LocoMail.

### Group 1 - Tutorial

This group contains the example LocoMail document used in the tutorial section of this User Guide, and in Part II.

ADDRESS.LST

Example address list for use with tutorial examples, to show how to merge a data file with a master document.

ADDRESS2.LST

A sparse address list to demonstrate how little information a data file need contain. (Tutorial Session 5)

FILLDOC.EG

A simple master document to show the use of Fill mode. (Tutorial Session 3)

LABEL.EG

A simple master document to show the generation of labels, using the names and addresses from an address list. (Tutorial Session 1)

MAILSHOT.EG

A simple mailshot letter, using names and addresses from a data file. (Tutorial Session 1)

MASTER.EG

Letter used to demonstrate how to prepare a master document. (Tutorial Session 4)

MERGEDOC.EG1

A simple document used to show typing in details while merging a document. (Tutorial Session 1)

MERGEDOC.EG2

A document used to show how typed-in details can be re-used. (Tutorial Session 2)

MERGEDOC.EG3

A document used to show information-gathering at the start of the document. (Tutorial Session 2)

MERGEDOC.EG4

An improved version of MERGEDOC.EG3. (Tutorial Session 2)

SAMPLE.EG

A document which is used to demonstrate the correct choice of LocoMail instructions for various blanks. (Chapter 3)

## Group 2 – Advanced

This group contains sample LocoMail documents that illustrate the advanced features of LocoMail described in Part III of this User Guide.

|                                  |                                                                                                                             |
|----------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| ACCOUNT.EG                       | Illustrates the use of arithmetic within a master document. (Chapter 8)                                                     |
| ADVANCE.EG                       | Illustrates how to insert information from more than one data record in a single document. (Chapter 6)                      |
| CLAUUSE.EG                       | Illustrates how to keep paragraph or clause numbers consecutive when one or more of the paragraphs are omitted. (Chapter 8) |
| LOOP.EG1<br>LOOP.EG2<br>LOOP.EG3 | } Show the use of Loop instructions to repeat a group of LocoMail instructions a given number of times. (Chapter 9)         |
| NOTPFORM.EG                      |                                                                                                                             |
| OMIT.EG                          |                                                                                                                             |
| OPTION.EG                        | Used with PERFORM.EG to illustrate how the readability improves when Perform instructions are used. (Chapter 9)             |
| PERFORM.EG                       | Illustrates how sections of a master document are marked for inclusion only under certain conditions. (Chapter 7)           |
| REPEAT.CND                       | Illustrates the inclusion of a series of alternative sections of text. (Chapter 7)                                          |
| REPEAT.NUL                       | Used with NOTPFORM.EG to illustrate how the readability improves when Perform instructions are used. (Chapter 9)            |
| SELECT.EG                        | Illustrates using a conditional instruction to control the repetition of a group of instructions. (Chapter 9)               |
| STRINGS.EG                       | Shows how to use <b>[ENTER]</b> to control the repetition of a group of instructions. (Chapter 9)                           |
| SUCCESS.EG                       | Illustrates how to make a mailshot selective. (Chapter 6)                                                                   |
|                                  | Illustrates concatenation of information. (Chapter 8)                                                                       |
|                                  | Illustrates how to retain the current data record for use in preparing the next version of the document. (Chapter 6)        |

## Group 3 – Club

This group contains a number of LocoMail documents to help with running a club. The documents are described in Chapter 12.

|              |                                                                                                   |
|--------------|---------------------------------------------------------------------------------------------------|
| CLUBFILE     | Data file document containing all membership details.                                             |
| FEESLABL     | Produces address labels only for those members receiving a reminder letter.                       |
| FEESLETR     | Produces reminder letters for the fees due in an month.                                           |
| MEMBER.LST   | Neatly formats and prints out every record in the data file for a complete membership list.       |
| NEWSLETR     | Produces customised letters, where the text depends on the member's interests.                    |
| SPECIFIC.LST | Prints out the names and phone numbers of members with specific interests and/or membership type. |
| UPDATE       | Produces a new version of the data file, complete with new member's details.                      |

## Group 4 – Invoices

This group contains a number of sample LocoMail documents suitable for maintaining sales records of a business. The documents are described in Chapter 13.

|            |                                                                                     |
|------------|-------------------------------------------------------------------------------------|
| ALLSALES   | Data file document containing a list of example sales records.                      |
| INVOICE    | Produces invoices for all unpaid sales on a particular day.                         |
| NEWSALES   | Produces a new version of the data file, complete with new sales details.           |
| RECEIPTS   | Updates status of selected sales records in the data file from unpaid to paid.      |
| REMINERS   | Produces reminder letters for overdue unpaid sales.                                 |
| TOTAL.INV  | Produces a list of all sales in a particular quarter, with totals of sales and vat. |
| UNPAID.INV | Produces a list of all unpaid sales, with totals.                                   |

## Group 5 – Enquiry

This group contains sample LocoMail documents for storing details of properties on an estate agency's books, and for listing properties meeting a purchasers requirements. The documents are described in Chapter 14.

|           |                                                                |
|-----------|----------------------------------------------------------------|
| HOUSES.EG | Data file containing details of houses, flats, etc.            |
| RESPONSE  | Prints out details of all properties meeting enquirer's needs. |

## Group 6 – Labels

This group contains sample LocoMail documents to produce labels on special labels stationery. The documents are described in Chapter 15.

|             |                                                                                   |
|-------------|-----------------------------------------------------------------------------------|
| LABELDAT.EG | Data file containing example names and addresses suitable for LABELDOC.EG master. |
| LABELDOC.EG | Master document which prints labels on 3-across continuous stationery.            |

## Group 7 – Building Blocks

This group contains 'building blocks' – groups of LocoMail instructions to perform specific tasks. They are described in Appendix III.

|              |                                                                                                                                          |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------|
| CALCVAT1.BDG | Calculates a VAT-inclusive amount from a VAT-exclusive amount.                                                                           |
| CALCVAT2.BDG | Calculates a VAT-exclusive amount from a VAT-inclusive amount.                                                                           |
| FINDNAME.BDG | A master document which searches through a data file, printing out records which match a particular name (one matching record per pass). |
| LISTALL.BDG  | A master document which searches through every record in a data file and lists details from all records matching a typed name.           |

|              |                                                                                                                                                                                         |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| OMITPARA.BDG | Simple conditional instructions to select (manually) which pieces of text are inserted.                                                                                                 |
| PROGUNIT.GEN | A simple program unit 'shell', repeated until 'No' is typed. Instructions for any repetitive task can be inserted into this program unit                                                |
| PROGUNIT.IMP | An improved program unit 'shell', which prompts the user whether to repeat the program unit and accepts only specific responses.                                                        |
| PROGUNIT.LIN | The improved program unit shell, complete with instructions to build up lines of an invoice.                                                                                            |
| PROGUNIT.SUM | A program unit which adds a number to a running total, and is repeated until 0 is typed. The total is displayed.                                                                        |
| PROGUNIT.TOT | Combines the actions of the previous two blocks – producing lines on an invoice and giving the total sum when the list is complete. Uses the improved response checking of PROGUNIT.IMP |

## Appendix III

# Building blocks

This appendix contains a number of 'building blocks' – self-contained groups of instructions that you might use to perform 'standard tasks'. All of the building blocks described are supplied on your LocoMail example documents disc, in the BUILDING group.

You can embed any of these building blocks directly into a master document you are creating. Just change the item-names in the building block to fit with the ones you use elsewhere in your master. In particular, change any item-names in the building block that you use for different information in the master document, or that clash with the item-names in the data file you want to use. Where the building blocks use program units of instructions, change the program unit name and/or the test variable controlling the looping action to suit your own circumstances.

Each section of this appendix includes a listing of a building block together with a brief description of the structure.

In general, you can see how each block works by filling it using LocoMail's Fill mode. The exceptions to this are the master documents described in Section III.2. These masters are designed to search through a data file; the best way to see them in action is to merge them with the ADDRESS.LST data file in the TUTORIAL group.

The methods used in each case are by no means the only solution to the task in hand – you may well think of a better way, or one more suited to the way you've arranged your data files, etc. Nevertheless, with a little adjustment to prompts and item-names, the basic building blocks can be put to a number of uses.

Many of the principles shown in these building blocks are employed in the example documents of Part IV.

## III.1 Simple VAT calculations

These short building blocks will quickly convert VAT-exclusive prices of single items to VAT-inclusive prices, and vice versa. They use the arithmetic and formatting instructions described in Chapter 8. This means that they can only work with strictly numeric information (ie. values must not contain any characters other than a decimal point, a + or – sign and the numbers 0 – 9). *Note that if you want to perform VAT calculations for multiple items, you should refer to a suitable reference text (eg. "HM Customs & Excise Notice 700 – The VAT Guide") for the precise methods of rounding and truncation required by law.*

## Calculating VAT-inclusive prices from exclusive prices

CALCVAT1.BDG operates by taking the value currently stored under **vat\_exc** and adding the current rate of **vat**, storing the result as **vat\_inc**.

|                                                                                                                                                    | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|----------------------------------------------------------------------------------------------------------------------------------------------------|---|---|---|---|---|---|---|---|
| $\begin{aligned} & \text{float } \text{water} = 0.19 \\ & \text{float } \text{not\_excited} = \text{water} / 2.0 \\ & \text{float } \end{aligned}$ |   |   |   |   |   |   |   |   |

The calculated vat-inclusive amount is truncated to two decimal places (ie. rounded down to the nearest penny). You can insert the contents of `vat_inc` in your master documents using a simple `(+Mail)vat_inc(-Mail)` instruction.

## Calculating VAT-exclusive prices from inclusive prices

CALCVAT2.BDG operates by taking the value stored under **vat\_inc** and deducting the vat from this value, storing the result as **vat\_exc**.

|                                                                   |   |   |   |   |   |   |   |   |   |
|-------------------------------------------------------------------|---|---|---|---|---|---|---|---|---|
| 0                                                                 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| <pre> (=ail)rate=1/9 val exc=(val inc/(1+rate/12)* (5-51)) </pre> |   |   |   |   |   |   |   |   |   |

The vat-exclusive amount is rounded to the nearest penny. As before, you can insert the contents of **vat\_exc** using a simple **(+Mail)vat\_exc(-Mail)** instruction.

### III.2 Searching through a data file

The following building blocks can be used to search through all the records in a data file, looking for and printing out records that match a particular name (or part of a name) that you type. FINDNAME.BDG works through the data file producing one match for each pass; if you need to produce all the matches in one pass (such as when you want to include a list of all matching records into a document you are preparing) you should use the LISTALL.BDG building block.

Both these building blocks have been set up to work with a simple address list data file such as ADDRESS.LST. This is purely so that you can quickly see how they work. By changing the item-name that these building blocks search through from **name** to something else, changing the items that are printed out and changing the prompts that request the name (or part of name) to look for, you can use these building blocks to search for almost anything in your data files – people whose appointments are due in October, the phone numbers of everyone with the title Dr., a list of telephone numbers for all 1st team players, etc.

Both these building blocks require that there be no 'nameless' records in the data file; in each case the program unit that searches for matches will stop when the name becomes null. If you use them to search through your own data files, be sure to change the test variable that controls the repetition of the program unit to an appropriate one for your data file (ie. to an item that is never left blank in your data records).

## Finding one matching record per pass

When the FINDNAME.BDG building block is merged with a suitable data file (ie. one with corresponding *item-names*), the user is prompted for the particular name to be searched for. 'Wildcard' characters are then appended to give some flexibility in the searching, so that even if you only know part of the name the match will still be found. FINDNAME.BDG then checks through the records.

When a LocoMail finds a record where the name information matches what you typed, the value held in **count** is incremented and the **name**, **address** and **country** details for that record are displayed – the pass is now complete. If no match is found, LocoMail will stop when it reaches the end of the data file and display a suitable message. Each time you start a new pass, LocoMail will carry on working through the data file from its current position.

```

1 Mail[search]=type name (or part of name) to look for
2 findstring="w" & search & "w", append wildcards, to widen scope of search
3
4 count=0; the count variable records the number of finds made, set to 0
5
6 loop=1; only on the first pass
7
8 now step through records until match found (or end of data file)
9
10 findstring=ance OH name=nl r c c count=c+1; if found
11
12 if match is found, then display it
13
14 if not found, then
15 count=count+1
16
17 Mail=line address r (Mail)name={Mail} is:w
18
19 Mail=address({Mail})w
20
21 Mail=countw
22
23 ELSE if no matches now open count down until count=0
24
25 count=0; reset the count
26
27 Mail=1; start at the beginning of the dataset
28
29 Mail=1

```

If you want to see `FINDNAME.BDG` in action, you can merge it with the `ADDRESS.LST` data file, in the `TUTORIAL` group on your LocoMail example documents disc.

Using program units of instructions is described in Chapter 9.

## Listing all matching records

The LISTALL.BDG building block works through the whole data file in one pass, printing out details from each record where the name matches the one you type in. It does this by repeating a program unit of instructions. The instructions within the Loop program unit compare the typed information (with suitable wildcards appended, to find matches when the whole name isn't known) with the information in the `name` item in a record if the two match, then the value held in

**count** is incremented and the details are printed. A **\$+** instruction is then used to move on to the next record.

This program unit is repeated until every record in the data file has been checked, using the instruction `@name%loop`.

```

1 (Mail)P
2 findstringP Name (or part of name) to search for'
3 findstringP & findstring & "P" ; append wildcards to match with parts of
4 nameP
5 count=0 ; 'count' makes the number of matches found - set to 0 initiallyP
6 ; now define the program unit to build up list of matchesP
7 loop=1
8 findstringName := (count=countP)
9 names(Mail)P replace the next line with the item-names you want to displayP
10 names(Mail)P ; (Mail)countP ("Mail")P
11 (Mail)P ;
12 ; now step on to next recordP
13 (Mail)P
14
15 repeat loopP
16 until (Mail)P
17 (Mail)P
18 (Mail)P
19 (Mail)P
20 (Mail)P
21 (Mail)P
22 (Mail)P
23 (Mail)P
24 (Mail)P
25 (Mail)P
26 (Mail)P
27 (Mail)P
28 (Mail)P
29 (Mail)P
30 (Mail)P
31 (Mail)P
32 (Mail)P
33 (Mail)P
34 (Mail)P
35 (Mail)P
36 (Mail)P
37 (Mail)P
38 (Mail)P
39 (Mail)P
40 (Mail)P
41 (Mail)P
42 (Mail)P
43 (Mail)P
44 (Mail)P
45 (Mail)P
46 (Mail)P
47 (Mail)P
48 (Mail)P
49 (Mail)P
50 (Mail)P
51 (Mail)P
52 (Mail)P
53 (Mail)P
54 (Mail)P
55 (Mail)P
56 (Mail)P
57 (Mail)P
58 (Mail)P
59 (Mail)P
60 (Mail)P
61 (Mail)P
62 (Mail)P
63 (Mail)P
64 (Mail)P
65 (Mail)P
66 (Mail)P
67 (Mail)P
68 (Mail)P
69 (Mail)P
70 (Mail)P
71 (Mail)P
72 (Mail)P
73 (Mail)P
74 (Mail)P
75 (Mail)P
76 (Mail)P
77 (Mail)P
78 (Mail)P
79 (Mail)P
80 (Mail)P
81 (Mail)P
82 (Mail)P
83 (Mail)P
84 (Mail)P
85 (Mail)P
86 (Mail)P
87 (Mail)P
88 (Mail)P
89 (Mail)P
90 (Mail)P
91 (Mail)P
92 (Mail)P
93 (Mail)P
94 (Mail)P
95 (Mail)P
96 (Mail)P
97 (Mail)P
98 (Mail)P
99 (Mail)P
100 (Mail)P
101 (Mail)P
102 (Mail)P
103 (Mail)P
104 (Mail)P
105 (Mail)P
106 (Mail)P
107 (Mail)P
108 (Mail)P
109 (Mail)P
110 (Mail)P
111 (Mail)P
112 (Mail)P
113 (Mail)P
114 (Mail)P
115 (Mail)P
116 (Mail)P
117 (Mail)P
118 (Mail)P
119 (Mail)P
120 (Mail)P
121 (Mail)P
122 (Mail)P
123 (Mail)P
124 (Mail)P
125 (Mail)P
126 (Mail)P
127 (Mail)P
128 (Mail)P
129 (Mail)P
130 (Mail)P
131 (Mail)P
132 (Mail)P
133 (Mail)P
134 (Mail)P
135 (Mail)P
136 (Mail)P
137 (Mail)P
138 (Mail)P
139 (Mail)P
140 (Mail)P
141 (Mail)P
142 (Mail)P
143 (Mail)P
144 (Mail)P
145 (Mail)P
146 (Mail)P
147 (Mail)P
148 (Mail)P
149 (Mail)P
150 (Mail)P
151 (Mail)P
152 (Mail)P
153 (Mail)P
154 (Mail)P
155 (Mail)P
156 (Mail)P
157 (Mail)P
158 (Mail)P
159 (Mail)P
160 (Mail)P
161 (Mail)P
162 (Mail)P
163 (Mail)P
164 (Mail)P
165 (Mail)P
166 (Mail)P
167 (Mail)P
168 (Mail)P
169 (Mail)P
170 (Mail)P
171 (Mail)P
172 (Mail)P
173 (Mail)P
174 (Mail)P
175 (Mail)P
176 (Mail)P
177 (Mail)P
178 (Mail)P
179 (Mail)P
180 (Mail)P
181 (Mail)P
182 (Mail)P
183 (Mail)P
184 (Mail)P
185 (Mail)P
186 (Mail)P
187 (Mail)P
188 (Mail)P
189 (Mail)P
190 (Mail)P
191 (Mail)P
192 (Mail)P
193 (Mail)P
194 (Mail)P
195 (Mail)P
196 (Mail)P
197 (Mail)P
198 (Mail)P
199 (Mail)P
200 (Mail)P
201 (Mail)P
202 (Mail)P
203 (Mail)P
204 (Mail)P
205 (Mail)P
206 (Mail)P
207 (Mail)P
208 (Mail)P
209 (Mail)P
210 (Mail)P
211 (Mail)P
212 (Mail)P
213 (Mail)P
214 (Mail)P
215 (Mail)P
216 (Mail)P
217 (Mail)P
218 (Mail)P
219 (Mail)P
220 (Mail)P
221 (Mail)P
222 (Mail)P
223 (Mail)P
224 (Mail)P
225 (Mail)P
226 (Mail)P
227 (Mail)P
228 (Mail)P
229 (Mail)P
230 (Mail)P
231 (Mail)P
232 (Mail)P
233 (Mail)P
234 (Mail)P
235 (Mail)P
236 (Mail)P
237 (Mail)P
238 (Mail)P
239 (Mail)P
240 (Mail)P
241 (Mail)P
242 (Mail)P
243 (Mail)P
244 (Mail)P
245 (Mail)P
246 (Mail)P
247 (Mail)P
248 (Mail)P
249 (Mail)P
250 (Mail)P
251 (Mail)P
252 (Mail)P
253 (Mail)P
254 (Mail)P
255 (Mail)P
256 (Mail)P
257 (Mail)P
258 (Mail)P
259 (Mail)P
260 (Mail)P
261 (Mail)P
262 (Mail)P
263 (Mail)P
264 (Mail)P
265 (Mail)P
266 (Mail)P
267 (Mail)P
268 (Mail)P
269 (Mail)P
270 (Mail)P
271 (Mail)P
272 (Mail)P
273 (Mail)P
274 (Mail)P
275 (Mail)P
276 (Mail)P
277 (Mail)P
278 (Mail)P
279 (Mail)P
280 (Mail)P
281 (Mail)P
282 (Mail)P
283 (Mail)P
284 (Mail)P
285 (Mail)P
286 (Mail)P
287 (Mail)P
288 (Mail)P
289 (Mail)P
290 (Mail)P
291 (Mail)P
292 (Mail)P
293 (Mail)P
294 (Mail)P
295 (Mail)P
296 (Mail)P
297 (Mail)P
298 (Mail)P
299 (Mail)P
300 (Mail)P
301 (Mail)P
302 (Mail)P
303 (Mail)P
304 (Mail)P
305 (Mail)P
306 (Mail)P
307 (Mail)P
308 (Mail)P
309 (Mail)P
310 (Mail)P
311 (Mail)P
312 (Mail)P
313 (Mail)P
314 (Mail)P
315 (Mail)P
316 (Mail)P
317 (Mail)P
318 (Mail)P
319 (Mail)P
320 (Mail)P
321 (Mail)P
322 (Mail)P
323 (Mail)P
324 (Mail)P
325 (Mail)P
326 (Mail)P
327 (Mail)P
328 (Mail)P
329 (Mail)P
330 (Mail)P
331 (Mail)P
332 (Mail)P
333 (Mail)P
334 (Mail)P
335 (Mail)P
336 (Mail)P
337 (Mail)P
338 (Mail)P
339 (Mail)P
340 (Mail)P
341 (Mail)P
342 (Mail)P
343 (Mail)P
344 (Mail)P
345 (Mail)P
346 (Mail)P
347 (Mail)P
348 (Mail)P
349 (Mail)P
350 (Mail)P
351 (Mail)P
352 (Mail)P
353 (Mail)P
354 (Mail)P
355 (Mail)P
356 (Mail)P
357 (Mail)P
358 (Mail)P
359 (Mail)P
360 (Mail)P
361 (Mail)P
362 (Mail)P
363 (Mail)P
364 (Mail)P
365 (Mail)P
366 (Mail)P
367 (Mail)P
368 (Mail)P
369 (Mail)P
370 (Mail)P
371 (Mail)P
372 (Mail)P
373 (Mail)P
374 (Mail)P
375 (Mail)P
376 (Mail)P
377 (Mail)P
378 (Mail)P
379 (Mail)P
380 (Mail)P
381 (Mail)P
382 (Mail)P
383 (Mail)P
384 (Mail)P
385 (Mail)P
386 (Mail)P
387 (Mail)P
388 (Mail)P
389 (Mail)P
390 (Mail)P
391 (Mail)P
392 (Mail)P
393 (Mail)P
394 (Mail)P
395 (Mail)P
396 (Mail)P
397 (Mail)P
398 (Mail)P
399 (Mail)P
400 (Mail)P
401 (Mail)P
402 (Mail)P
403 (Mail)P
404 (Mail)P
405 (Mail)P
406 (Mail)P
407 (Mail)P
408 (Mail)P
409 (Mail)P
410 (Mail)P
411 (Mail)P
412 (Mail)P
413 (Mail)P
414 (Mail)P
415 (Mail)P
416 (Mail)P
417 (Mail)P
418 (Mail)P
419 (Mail)P
420 (Mail)P
421 (Mail)P
422 (Mail)P
423 (Mail)P
424 (Mail)P
425 (Mail)P
426 (Mail)P
427 (Mail)P
428 (Mail)P
429 (Mail)P
430 (Mail)P
431 (Mail)P
432 (Mail)P
433 (Mail)P
434 (Mail)P
435 (Mail)P
436 (Mail)P
437 (Mail)P
438 (Mail)P
439 (Mail)P
440 (Mail)P
441 (Mail)P
442 (Mail)P
443 (Mail)P
444 (Mail)P
445 (Mail)P
446 (Mail)P
447 (Mail)P
448 (Mail)P
449 (Mail)P
450 (Mail)P
451 (Mail)P
452 (Mail)P
453 (Mail)P
454 (Mail)P
455 (Mail)P
456 (Mail)P
457 (Mail)P
458 (Mail)P
459 (Mail)P
460 (Mail)P
461 (Mail)P
462 (Mail)P
463 (Mail)P
464 (Mail)P
465 (Mail)P
466 (Mail)P
467 (Mail)P
468 (Mail)P
469 (Mail)P
470 (Mail)P
471 (Mail)P
472 (Mail)P
473 (Mail)P
474 (Mail)P
475 (Mail)P
476 (Mail)P
477 (Mail)P
478 (Mail)P
479 (Mail)P
480 (Mail)P
481 (Mail)P
482 (Mail)P
483 (Mail)P
484 (Mail)P
485 (Mail)P
486 (Mail)P
487 (Mail)P
488 (Mail)P
489 (Mail)P
490 (Mail)P
491 (Mail)P
492 (Mail)P
493 (Mail)P
494 (Mail)P
495 (Mail)P
496 (Mail)P
497 (Mail)P
498 (Mail)P
499 (Mail)P
500 (Mail)P
501 (Mail)P
502 (Mail)P
503 (Mail)P
504 (Mail)P
505 (Mail)P
506 (Mail)P
507 (Mail)P
508 (Mail)P
509 (Mail)P
510 (Mail)P
511 (Mail)P
512 (Mail)P
513 (Mail)P
514 (Mail)
```

To amend this building block to suit your own requirements, change the item-name that is compared in the conditional instruction and similarly change the item-names that are displayed to the ones you require (if necessary, change the test variable as well).

### III.3 Selecting text to be included

You may want a master document to offer you the option of including certain pieces of text, paragraphs or clauses in the current version. A very simple way to do this is to get *LocoMail* to ask the user whether each of the optional paragraphs are to be included or not, as shown in *OMITPARA.BDG*.

```

(Header)parameterN-1; set up numbering so that first paragraph to appear
parameterN-1; will be number 1
parameterN-1;
parameterN-1; include text of paragraph 1 (type yes or NO)
parameterN-1; parameterN-1; insert text of paragraph 1 here
(parameterN-1)
parameterN-2; include text of paragraph 2 (type yes or NO)
parameterN-2; parameterN-2; insert text of paragraph 2 here
(parameterN-2)
parameterN-3; include the text of paragraph 3 (type yes or NO)
parameterN-3; parameterN-3; insert text of paragraph 3 here
(parameterN-3)
(parameterN-3)

```

In this simple case, each piece of text (complete with a paragraph number, indent tab and with a  $\hookleftarrow$  after the text) is the action in a single conditional instruction. The more sophisticated methods of getting LocoMail to omit and include text automatically are described in Chapter 7. The use of the LocoMail counting instructions to number the paragraphs correctly was described in Chapter 8.

### III.4 Repeating an action

A very common requirement in filling in versions of a document is to repeat an action a number of times, but where the number of times you perform the actions depends on the document you're creating. So you need some way of repeating the action until you decide you've finished, then of telling LocoMail to stop.

PROGUNIT.GEN shows a simple method of using a program unit (called 'loop') to do this, with the user controlling how often loop is performed. In fact, the loop will be repeated, and whatever action you have included in loop performed, until the user types **No** (or anything beginning with **N**). Repeating program units of instructions like this is described in Chapter 9.

```

(Shell)loop=(Shell)
insert the action you want to perform in place of this comment line
coltype No to finish, Yes to continue
loop=[if-ik<{finish=0};{finish=1;}
(Shell)
%loopfinish(Shell)
here is some text

```

If you include this building block into your own master document, replace the line beginning with the semi-colon with the action you want to carry out. Note that a quoted string has been included within the program unit. To avoid any possible problems with the embedded string, we have used **(+Mail)** and **(-Mail)** codes to shift the content of the program unit 'down a level' and back 'up a level', as described in Chapter 10.

## Rejecting invalid responses

In the above example, the program unit `loop` is repeated until the user types `No` (or any response beginning with `N`). Typing anything but this will repeat the action, which could be annoying if the response was merely mistyped (eg. `Bo`) and the ensuing action inserts extra details in the current version. It is possible to use a more sophisticated approach to this task, so that specific responses are needed to repeat or stop the loop, and errors (or deliberate mistyping) will have no effect. `PROGUNIT.IMP` is the improved building block that will do this task.

`PROGUNIT.IMP` is based around the `PROGUNIT.GEN` – a program unit (`mainloop`) is set up and will be repeated until a test variable **finish** becomes zero.

(Mail)mainloop:(Mail)\*  
 Insert the action you want to perform here.  
 Now use a group of instructions to work out whether mainloop is to be repeated.  
 resword = "w"; Store the acceptable responses under names.  
 word = "w"; Note onward - yes' response can be anything beginning with 'w'  
 word = "w"; No' response can only be the word 'no'.  
 wordly = "y"; Love no to finish, yes to continue.  
 wordly = word; If user types 'w', mainloop finishes.  
 wordly = word; If user types 'y', mainloop is repeated.  
 X: endinner = 1; and if anything else, response is ignored.  
 wordinner = "X";  
 end of mainloop (Mail)\*  
 Next appears before 'mainloop' is performed.  
 (Mail)mainloop:(Mail)\*  
 This text appears after the program unit has been performed.

Where `PROGUNIT.IMP` differs is that the `mainloop` program unit contains another group of instructions. This inner group is repeated until another test variable, `endinner`, becomes zero, using an instruction of the type introduced in Section 9.6:

```
%"...@endinner
```

The instructions in the inner group (ie. within these double quote marks) are designed to ask whether the program unit needs to be repeated, and from the response to control the value of **endinner** and **finish**. The acceptable responses are set up under item-names before they are used in the group of instructions, as recommended in Chapter 10; the item-names can then be used within the group of instructions without using Mail codes to move between further levels.

The only response you want to accept as "Yes" is stored in **yesword** and the only response you want to accept as "No" is stored in **noword**. Any other responses will have no effect on the **mainloop** program unit. Note that wildcards can be used when defining **yesword** and **noword**, to add flexibility to the comparisons.

The group of instructions is then set up. The first of the instructions in this group supplies the prompt **Type No to finish, Yes to continue**. The subsequent response is stored in **reply**. This response is then used to work out whether to repeat **mainloop** or not, or (if the response wasn't one of the acceptable ones specified above) the user is prompted again. The instruction that does this is of the **IF... ELSE IF... ELSE IF... ELSE...** form described in Chapter 7:

```
#reply=noword : { endinner=0 if finish=0 ; if user types No, mainloop finishes?
#reply=yesword : { endinner=0 if finish=1 ; if user types Y..., mainloop is
repeated?
} { endinner=1 ; } and if anything else, response is ignored.
```

- If the response is No (or whatever you have stored in **noword**) then the user has finished and so the variables that control this inner group of instructions (**endinner**) and that control the **mainloop** program unit (**finish**) are both set to zero. LocoMail will then carry on with the rest of the document.
  - If the response is Yes (or whatever you have stored in **yesword**) then this group of instructions is complete so **endinner** is set to 0. However, **mainloop** is to be repeated, so **finish** is set non-zero.
  - If the response is neither yes or no, then the user must be prompted to type the response again; **endinner** is set non-zero, meaning that this inner group of instructions is performed again.
- Because **endinner** is used to tell LocoMail whether to repeat the group of instructions or not, this inner group of instructions is repeated until the response to the prompt matches either **yesword** or **noword**.

Because **endinner** is used to tell LocoMail whether to repeat the group of instructions or not, this inner group of instructions is repeated until the response to the prompt matches either **yesword** or **noword**.

This more sophisticated method of controlling a repeating loop is used in many of the following building blocks.

## Adding lines to an invoice list

One of the most common applications of a repeated group of instructions is to insert details of items into an invoice. Some invoices might only include one or two lines, some might include ten! So you would use a loop to repeat an action, just as shown in the previous building block.

In this case, the action that you need to carry out each time through the loop is to insert the description of the item, the number of that item purchased, and its unit cost. LocoMail then works out the sub-total for this line. To repeat this sequence of instructions, inserting line after line into the invoice, PROGUNIT.LIN is based on the PROGUNIT.IMP program unit, so once again only the responses stored in **noword** and **vesword** will affect the main program unit:

```

<Mail>mainloop=<Mail>";
 ; prompt for details for one line of invoice";

 # number ; number of items<Mail>.<Mail>";
 ; description of item<Mail>.<Mail>";
 # item unit ; item cost<Mail>.<Mail>subtotal=unit*number<Mail>.<Mail>";
 # item cost<Mail>.<Mail>";
 ; response = "yes" ; "Yes" response can only be the word yes";
 ; response = "no" ; "No" response can only be the word no";

 ; Now use group of instructions to work out whether mainloop is to be repeated";
 ; reply = 0 ; More lines on invoice"; type No to finish, Yes to continue";

 #reply=noword ; <end inner> if finish=<";
 >#reply=noword ; <end inner> if finish=<";
 }<end inner>=<";

 <end inner>";

 <end of mainloop(<Mail>)<";

 <Mail>";
 <Mail>mainloop(finish<Mail>)<";

```

Note that to carry out the calculation for the subtotal, **number** and **unit** must be entered as true numbers (no £ signs, etc.). **Subtotal** is rounded to 2 decimal places (ie. pounds and pence) before printing. Using LocoMail to carry out arithmetic and to format results is described in Chapter 8.

After each line has been printed, the user is prompted to say whether there are any more entries, as described for the PROGUNIT.IMP building block.

This building block puts tabs between the items on each line. The tab positions, headings for the columns, etc. should all be included within the master document. To align the calculated subtotals on each line, you should set up a decimal tab in your master document at the appropriate position. (Decimal tabs are described on page 120 of the LocoScript 2 User Guide, and page 154 of the PCW9512 User Guide.)

## Keeping a running total of typed numbers

PROGUNIT.SUM shows another simple use of repeating a group of instructions, where LocoMail keeps a running total of the numbers typed in. The value of the variable used to store the total (**total**) is initially set to zero. A program unit **loop** is then set up; the instructions in this program unit fetch and store a number from the keyboard, insert it into the document and add it to **total**.

```

%Mail) first, set the running total to zero
Total=0;
% now set up the instructions to ask for values and add to running total
; and group in "Loop"
do
 100=1; %Mail)
 % % number : type number (type 0 to finish)
 ; add what is typed to the running total
 Total=Total+number;
until (Mail)=0
%Mail) when (Mail)=0
%Mail) return(Mail)
end

```

The typed value is stored in a variable called **number**; this value can then be used to control how often the program unit is performed using the instruction **\*loop@number**. So **loop** is repeated until a zero is typed.

Note that the program unit is carried out immediately after a (**-Mail**) code; LocoMail is therefore inactive when it performs **loop**, and so the **←**, **+** and **→** at the start of **loop** are inserted into the document. To ensure that these characters are always inserted into the document however many times **loop** is repeated, the program unit finishes with a (**-Mail**) code (as described in Chapter 9).

The program unit `loop` is repeated, adding more values to `total`, until 0 (or a null) is entered. The value stored in `total` is then printed out. To align the numbers and the total in your finished document, you should set up a decimal tab in your master document at the appropriate position and delete the one in `PROGNIT.SUM`:

## Producing an invoice

The building block `PROGUNIT.TOT` combines the calculated subtotals (as produced for each line in the `PROGUNIT.LIN` building block) and the running addition of values (as shown in the `PROGUNIT.SUM` building block) within a program unit to produce an invoice. `PROGUNIT.TOT` also uses the method of controlling the program unit described in the `PROGUNIT.IMP` example.

Note that the tab positions, column headings, etc. must all be set within the master document. To align the prices and totals in the finished document, you should include decimal tabs in your master document at the appropriate points (and clear the tabs set in the building block).

[illegible]

## Troubleshooting

This appendix covers what to do and what may have gone wrong when something unexpected happens while you are using LocoMail. It covers only those problems which might arise due to LocoMail – more general problems (either with your PCW or with your use of the LocoScript 2 software) are described in either your LocoScript 2 User Guide or your PCW User Guide.

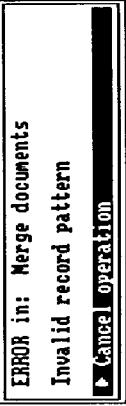
### IV.1 Trouble when selecting documents and data files

- The selection menu doesn't include the Automatic Merge option.

Your printer is not connected to your PCW or it is not available to LocoScript.

First press **[FTR]** to enter Printer Control State. From here you can check which printer LocoScript is expecting to print on, the status of the printer, etc. If you cannot select the printer you require, switch your PCW off and then check the printer connections as described in your PCW User Guide. You must not do this while the machine is switched on. Then switch back on, load LocoScript and use Printer Control State to check that your printer is now available to LocoScript.

- The following Alert message appears after you have selected the master document and data file, but before LocoMail starts producing a version:



ERROR in: Merge documents  
Invalid record pattern  
▶ Cancel operation

If merging with a LocoMail datafile (or a datafile dressed up as a LocoMail datafile), this suggests either that the file has no record pattern at all, or the record pattern has no terminator or it contains illegal item-names. Edit the data file you want to use and check that it does have a record pattern and that this has the structure specified in Part II, Chapter 2. Then try again.

If merging with a LocoFile datafile, consult the troubleshooting section in the LocoFile User Guide.

**Note:** This message will appear if you have stored the record pattern for the data file in a separate document but you select your data file first. You must always select the document holding the record pattern first.

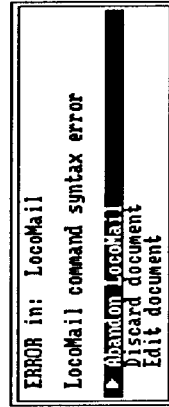
- You select a master file and a data file but LocoMail doesn't start the merge LocoMail is waiting for you to select the file containing the data – there is a message to this effect on the screen. It is waiting because it only found a record pattern in the first data file you selected. If the file contains data as well, the record pattern hasn't been given a proper record pattern terminator.

## IV.2 Trouble as LocoMail fills a document

**Note:** Many of the error messages that appear at this stage include a menu of options. What these options offer is as follows:

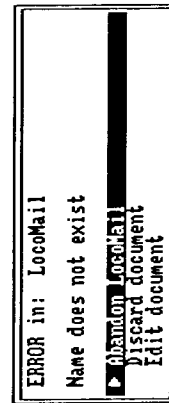
- **Abandon LocoMail** returns you immediately to the Disc Manager Screen, discarding the work you have done so far. You can then edit the document and correct the LocoMail instruction that caused the problem. Select this option if you can see what the trouble was and feel you know how to correct it.
- **Discard document** throws away the document being prepared at the moment and lets you start another. This is not an option to use as a general rule because it offers a stop-gap solution to the problem, allowing you to re-use a document that contains errors. Choose this option only if the error occurred in a section of the document that is only included conditionally and you know that other versions you want to prepare won't use this section of the document.
- **Edit document** stops the filling operation but keeps the current document on the screen. This is useful for seeing precisely where the trouble occurred because the cursor is left at the point in the LocoMail instruction where the error was detected. It can also be used to save sections of the document that have been processed, if required.

- The following message appears:



The instruction that LocoMail is currently processing is not a legal LocoMail instruction, ie. it doesn't have precisely the form specified in this User Guide.

- The following message appears:

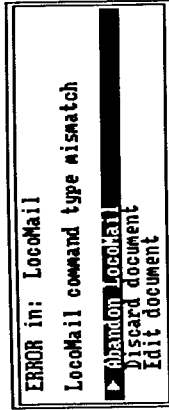


**Either:** You have asked for information to be entered from the data file but the item-name you have used doesn't appear in the record pattern. The chances are that you have misspelled one of the names.

**Or:** You are trying to use a name that has not been set as the test in a loop. (Chapter 9)

**Or:** You are trying to execute a program unit that hasn't been defined. (Chapter 9)

- The following message appears:

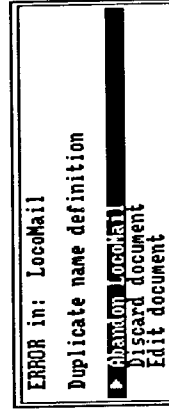


**Either:** One or more of the values in an arithmetic expression is not a pure number. A probable cause is the inclusion of a character like £ in a price or % in a discount rate. (Chapter 6)

**Or:** One of the conditions you have specified is trying to compare a text item with a number or you've forgotten to put the quotes around the text in the comparison. Strings in comparisons must begin and end with double quote marks. (Chapter 6)

**Or:** You are using a wildcard on both sides of a string comparison. Wildcards can only be used on one side of a string comparison. (Chapter 6)

- The following message appears:

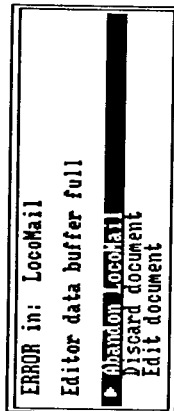


This normally indicates that you are trying to alter a value that has either been set through your data file's record pattern (Merge mode) or that LocoMail is treating it as if it came from a record pattern (Fill mode) because you asked to insert the value before you had set it.

In Merge mode, this error probably means that you have forgotten that this information is available from the data file. In Fill mode, you have probably forgotten to include this piece of information in your initial block of information gathering instructions. (Chapter 3)

If you are merging with a LocoFile datafile, consult the troubleshooting section of the LocoFile User Guide.

- The following message appears:



Either: The master document you are working on is too big or too complicated to be held in the available memory. Simplify the document, for example by replacing multiple tabs with one tab.

Or: Your LocoMail instructions are storing too much information for the available memory. Check that you are not accidentally concatenating information repeatedly, and that you have initialised any variables before using them.

- The information is inserted in the wrong blanks.

The data file has not been set up correctly. Check carefully that it obeys the rules given in Chapter 2. In particular check the use of separators in the record pattern against their use in the data records themselves, remembering that errors will occur if the character chosen as separator appears in the information it is being used to mark the end of. Pay particular attention to your use of `↵` in the record pattern and in the data records.

For example, a record pattern like this:

```
name ↵
address,postcode ↵
country ↵
```

used for a data record like this:

```
Fred Smith ↵
The House, ↵
The Town, AX1 20XY ↵
England ↵
```

will give you an address of just **The House** (up to the first comma), a blank postcode (because this comma is immediately followed by a carriage return – the separator marking the end of the postcode) and a country of **The Town, AX1 20XY** ←England.

Also, if you have specified `↵` as the first separator in your record pattern, then check that every data record has at least one character (typically a space character, if not real data) before the first return. All leading returns in a record (ie. returns before the first character of any type) are ignored by LocoMail.

- Only one line of the address is entered in the slot for the address and the remaining lines are used as the next item.

You probably have a record pattern that specifies the separator after the address as a `↵` but you are also using `↵` itself in the address. Use a different separator. (Chapter 2)

- The first version fills perfectly but the second is entirely blank.

You probably have both a separator and a record pattern terminator at the end of each of your data records. You should have either a separator or a record pattern terminator in the data records – NOT both.

- The usual Exit menu does not appear after all the LocoMail instructions have been processed. Instead the document is left on the screen with the cursor after the last instruction that was processed.

As the document was filled, you pressed `[ENTER]` more times than LocoMail expected or used. This may mean that some of the information has been put into the wrong slots, and to give you the chance to correct this, you have been put straight into editing mode.

Press `[EXIT]` when you have made the necessary changes. The Exit menu is then displayed, offering the usual options. If the document is too badly damaged to repair, return to the Exit menu and select **Discard result**.

- The Exit menu appears but doesn't include either **Print result** or **Save and Print result**.

Your printer is not connected to your PCW or it may be faulty. Save your work for now; then when you have finished, switch off and check the printer connection as described in your PCW User Guide. You must not do this while the PCW is switched on.

- A document 'disappears' without any output.

If the last line is a LocoMail instruction, make sure that it ends with a `↵` or `(-Mail)` code. Make sure that any complex structures (choices, loops or program units) finish properly (with a `:>` or `"`).

If there are any loops or program units in the master, make sure that the number of `(+Mail)` and `(-Mail)` codes balance, within each pair of double quote marks.

- Some instructions are not being obeyed, even though displayed.

Check whether they follow a comment or prompt. If this is the case, they must be separated by a `↵` or `(-Mail)` `(+Mail)`; a colon does not act as a separator after a comment or a prompt.

- Some instructions are simply inserted as text.

LocoMail is not 'active' when it processes these instructions. You probably need another `(+Mail)` code in your document.

Work through the part of your master document that is affected, checking where `(+Mail)` codes are making LocoMail active and where `(-Mail)` codes are making LocoMail inactive, and put in or take out Mail codes as necessary.

If the section of the document that is affected contains a program unit, the changes you need to make won't necessarily be easy to spot. Refer to Part III, Section 9.5 for further advice.

- If you are having difficulties with program units or loops. Start by reading the appropriate part of Section 9.5.
  - Check that the number of (+Mail) and (-Mail) codes balance, within each pair of double quote marks.
  - Check that there is a separator at the end of any program unit you use.
  - If a program unit includes embedded double quote characters, make sure that these are surrounded by (+Mail) and (-Mail) codes.
  - Try executing the program unit directly using a Perform instruction (&program-name) - if it works correctly when performed just once, the problem when repeating may be due to a mismatch of separators at the start or end of the loop.
- If LocoMail appears to have stopped working through the document.
  - LocoMail may be waiting for you to type something.
  - There may be a loop executing which cannot finish (because the test variable controlling it is never set to 0 or null). Alternatively, LocoMail may be executing a loop which takes a long time to do its task!

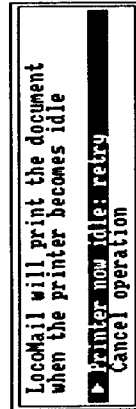
### IV.3 Trouble while printing the finished version

- You select Print result or Save and Print result and nothing is printed.

Your printer is either Off-line or Waiting for paper.

Check that there is paper in the printer and that the bail bar is back on the platen. If necessary, enter LocoScript's Printer Control State by pressing [PTE] and use the menus there to put the printer On-line. Details are given in your LocoScript 2 User Guide or the PCW9512 User Guide.

- You select Print result or Save and Print result and the following message box appears on the screen:



The message is asking you to wait while LocoMail finishes printing the current document. Only one document can be being prepared or waiting to print while another is being printed.

If you wish, you can use the Cancel option to abandon printing the document you have just prepared. Alternatively, you can press [PTE] and use the Printer Control State menus to abandon the current document.

- Other troubles you can experience while printing are explained in the troubleshooting section of your LocoScript 2 or PCW User Guide.

## Important Notice

THE SOFTWARE CONTAINED IN THE DISKETTE PACKAGE IS SUPPLIED TO YOU ON THE TERMS AND CONDITIONS INDICATED BELOW. THE OPENING OF THIS PACKAGE INDICATES YOUR ACCEPTANCE OF THESE TERMS AND CONDITIONS. IF SUCH TERMS AND CONDITIONS ARE NOT ACCEPTED BY YOU, YOU MUST RETURN THE UNOPENED PACKAGE TO THE PLACE OF PURCHASE AND YOUR MONEY WILL BE REFUNDED. NO REFUNDS WILL BE GIVEN WHERE THE PACKAGE HAS BEEN OPENED UNLESS THE PRODUCT IS FAULTY AND SUCH REFUND BECOMES PAYABLE UNDER CLAUSE 7 BELOW

In this notice, the terms:

'Locomotive' means Locomotive Software Limited

'The Program' means the program known as LOCOMAIL.JOY on the diskette supplied in the diskette package.

### 1. Copyright

Material within The Program is copyright Locomotive.

Locomotive grants to the purchaser of this package a non-exclusive right to use The Program in accordance with these terms and conditions. Such Licence may be transferred only in accordance with Clause 3 below. Any other use or dealing not expressly authorised by these terms and conditions is strictly prohibited.

### 2. Use

The Program may only be used on a single machine or terminal at any one time but may be copied in support of that use. Any such copying is subject to there being no modification of The Program and in particular to the copyright notices of Locomotive being preserved in the copy. Save for copying as aforesaid, any other operations (including modification or translation from machine readable form) are expressly prohibited.

### 3. Transfer

The Program may be transferred to a third party provided the original Program and all copies are transferred or otherwise destroyed and provided further these terms and conditions are produced to that third party and prior to the transfer that party agrees and undertakes to observe and continue to observe the same. Without such transfer and undertaking any application of The Program or copies thereof by any other person will not be authorised by Locomotive and will be in breach of Locomotive's copyright and other proprietary rights.

#### 4. Documentation

The documentation accompanying The Program is copyright Locomotive. However, no right to reproduce that documentation in part or in whole is granted by Locomotive. Should additional copies of the documentation be required for whatever reason, application must be made in writing to Locomotive which will be considered in its discretion.

#### 5. Breach

If the user for the time being acts in breach of any of these terms and conditions it shall indemnify Locomotive against all loss suffered (including loss of profits) and the licence granted hereunder shall be deemed to be terminated forthwith. On termination the user shall deliver up to Locomotive all infringing and lawful copies of The Program.

#### 6. Exclusions

Neither Locomotive nor any person authorised by it gives warranties or makes representations that The Program is error free or will meet functions required by the user. It shall be the responsibility of the user to satisfy itself that The Program meets the user's requirements. The Program is supplied on an 'as is' basis and save as expressly provided in these conditions all warranties of any nature (and whether express or implied) are excluded.

#### 7. Liability

Locomotive warrants that the diskette on which The Program is stored is free from material defect and through normal use will remain so for a period of 90 days after purchase. In the event of any breach of this warranty (or statutory warranty or conditions incapable of exclusion by these conditions) the responsibilities of Locomotive shall be limited to replacing the enclosed program or to returning the price paid for the same as they shall determine.

As the sole exception to the foregoing Locomotive will accept liability for death or personal injury resulting from its negligence. In no circumstance shall Locomotive be liable for any indirect or consequential costs damages or losses (including loss of business profits, operating time or otherwise) arising out of the use or inability to use the enclosed program and diskette and whether or not the likelihood of damage was advised to Locomotive or its dealer.

This notice does not affect your statutory rights.

## INDEX

- Abandon LocoMail, 32, 114
- Abandoning a merge, 102
- Abandoning current version, 114, 126
- ACCOUNT.EG, 152, 154
- Active status, 116
- Addition, 147
- ADDRESS.LST, 7, 11
- ADVANCE.EG, 129
- Advancing to next record, 129
- Alert messages, 41, 273
  - Command syntax error, 95, 274
  - Command type mismatch, 275
- Duplicate name definition, 96, 275
- Editor data buffer full, 276
- Invalid record pattern, 74, 273
- Name does not exist, 75, 274
- ALLSALES, 209
- Alternative pieces of text, 136
- AND instruction, 123
- Arithmetic, 147
  - addition, 147
  - combining arithmetic expressions, 153
  - division, 147
  - multiplication, 147
  - subtraction, 147
- Automatic merge, 99, 101
- Blank records
  - appearing unexpectedly, 68, 75
- Blanks
  - choosing suitable instructions, 88
  - filled from the keyboard, 22, 102
  - putting in master document, 38, 82
- Building blocks, 263
- CALCVAT1.BDG, 264
- CALCVAT2.BDG, 264
- Carriage return
  - as command separator, 92
  - as item separator, 61
- Carrying information over
  - between passes, 23, 86
- CLAUSE.EG, 144
- CLUBFILE, 191
- Codes
  - Mail codes, 10
  - Styling codes, 19
- Colon (as command separator), 92
- Combining data, 153
- stored under item-names, 155
- strings, 153
- Comma
  - as decimal point marker, 151
  - as thousands marker, 151
- Command separators, 92
- Command syntax error, 95, 274
- Command type mismatch, 275
- Comments, 83
  - definition, 253
- Comparisons
  - comparing numeric values, 122
  - comparing text, 121
  - inverting comparisons, 124
  - multiple comparisons, 123
  - wildcards in comparisons, 124
- Concatenation, 153
- Conditional instructions, 120
  - nesting conditional instructions, 179
  - testing, 125
  - with text, 133

- Conditions, 121  
definition, 256  
Converting existing data files, 69  
more complex cases, 71  
Counting, 143
- Data files, 54, 57  
adapting, 48  
advancing to the next record, 129  
ASCII files, 57, 69  
blank records, 68  
converting an existing file, 69  
definition, 251  
end-of-record markers, 63  
errors in data files, 74  
items in data files, 13, 57, 59  
keeping record for next pass, 130  
LocoScript files, 57  
matching to master documents, 60  
omitting items from records, 66  
planning, 45  
record pattern terminators, 63  
records, 57, 252  
separators, 13, 61  
setting up from scratch, 59  
spacing out items, 62, 64  
structure of, 57  
testing, 73  
types of data file, 57  
typing, 46, 64
- Decrementing numbers, 143  
Discard result, 114  
Duplicate name definition, 96, 275
- Edit result, 31, 113  
Editor data buffer full, 276  
End-of-record markers, 57, 63  
definition, 252  
separators as  
end-of-record markers, 68
- Errors  
in conditional instructions, 126  
in data files, 74  
in master documents, 95  
in program units, 175  
Troubleshooting, 273
- Examples  
choosing LocoMail  
instructions, 89  
club membership system, 191  
enquiries system, 235  
example documents, 259  
initial block of instructions, 90  
n-across labels, 248  
numbering clauses, 144  
sales and invoicing system, 209  
selective Mailshot, 126  
setting up special paper type, 245  
simple labels, 14, 243  
simple mailshot, 7, 10  
Exit menus, 113
- FEESLABL, 203  
FEESLETR, 201  
Fill again, 32  
Fill again/Finish menu, 32, 111  
Fill mode, 107  
end of each pass, 110  
selecting document to fill, 108  
when to use, 54
- FILLDOC.EG, 30, 33  
Filling a merge document, 33, 94  
FINDNAME.BDG, 265  
Formatting numbers, 149  
Formatting parts of a calculation, 149
- Grouping instructions, 157
- HOUSESE.EG, 235
- Improving the master document, 91  
Inactive status, 116  
Incrementing numbers, 143  
Installing LocoMail, 1  
Invalid record pattern, 74, 273  
Inverting comparisons, 124  
INVOICE, 216  
Invoices  
building blocks, 269, 271  
example invoice, 216  
keeping running totals, 270
- Item-names, 13, 54, 57, 59  
definition, 252, 256  
invalid names, 59  
rules for selecting, 43, 59  
suitable item-names, 59, 81  
using two-word names, 59  
Items, 13, 54, 57  
omitting items from records, 66
- LABEL.EG, 14  
LABELDAT.EG, 244  
LABELDOC.EG, 248  
Leaving items out of data records, 66  
LISTALL.BDG, 265  
LocoFile,  
Introduction, 51, 57, 99, 101  
LocoMail  
how it works, 53, 116  
technical description, 251  
LocoMail command type  
mismatch, 275
- LocoMail instructions  
choosing instructions, 88  
comment text, 83  
conditional instructions, 120  
copying information, 87  
definition, 253  
grouping instructions, 157  
improving layout, 91  
inserting in master document, 38  
nesting instructions, 179  
program units, 157  
prompt text, 83  
repeating instructions, 163, 177  
setting up information, 87  
storing data from keyboard, 85  
summary, 254  
taking data from data file, 84  
taking data from keyboard, 85  
taking numbers from keyboard, 85  
LocoMail Master disc, 259  
LocoMail status, 116  
Loop instructions, 163  
controlled by conditional  
instructions, 171  
controlling number of repeats, 166  
fixed number of repeats, 167
- repeated until null or zero  
typed, 169  
how Loop instructions work, 164  
problem solving, 175  
when followed by : or cr, 173  
LOOP.EG1, 168  
LOOP.EG2, 168  
LOOP.EG3, 168
- Mail codes, 10  
appearing in finished version, 96  
inserting in master  
document, 39, 82  
Mailshot example, 7  
MAILSHOT.EG, 7, 10  
Manipulating text and numbers, 141  
Manual merge, 99, 101  
Master documents  
alternative pieces of text, 136  
creating from existing  
document, 78  
creating from scratch, 78  
definition, 253  
designing, 183  
errors in master document, 95  
improving layout, 91, 185  
manipulating text and  
numbers, 141  
omitting sections of text, 133  
planning, 78, 183  
testing, 94, 111, 187  
where to use LocoMail  
instructions, 80
- MASTER.EG, 36  
MEMBER.LST, 193  
Menus  
Exit menu, 105, 113  
Fill again/Finish, 32, 111  
Merge mode, 99  
abandoning a merge, 102  
end of each pass, 104, 105  
Exit menu, 105  
number of passes made, 54  
selecting documents to merge, 100  
typing in extra information, 102  
when to use, 54

- MERGEDOC.EG1, 17, 21  
 MERGEDOC.EG2, 22  
 MERGEDOC.EG3, 26  
 MERGEDOC.EG4, 29  
 Multiple comparisons, 123
- Name does not exist, 75, 274
- Nested instructions, 179
- Nested strings, 180
- NEWSALES, 210
- NEWSLETR, 198
- NOT instruction, 124
- NOTPFORMEG, 162
- Numbers  
   decrementing numbers, 143  
   definition, 256  
   formatting, 149  
   formatting large numbers, 151  
   incrementing numbers, 143  
   rounding, 150  
   signed numbers, 150  
   specifying number of decimal places, 150  
   truncating, 150
- Numeric information, 142  
   definition, 142
- OMIT.EG, 134
- OMITPARA.BDG, 266
- Omitting sections of a document, 133
- building blocks, 266
- OPTIONEG, 138
- OR instruction, 123
- Passes, 54
- Pattern, see Record pattern
- Perform instructions, 160
- problem solving, 175  
   testing, 161  
   when followed by : or cr, 173
- PERFORM.EG, 162
- Print result, 113
- Printer now idle, 106, 278
- Program units, 157  
   building block, 267  
   controlled by conditional instructions, 171
- controlling number of repeats, 166
- defining a program unit, 158
- performing once, 160
- problem solving, 175
- repeated fixed number of times, 167
- repeated to end of data file, 170
- repeated until null or zero typed, 169
- repeating program units, 162  
   when followed by : or cr, 173
- PROGUNIT.GEN, 267
- PROGUNIT.IMP, 267
- PROGUNIT.LIN, 269
- PROGUNIT.SUM, 270
- PROGUNIT.TOT, 271
- Prompts, 22, 83
- Question mark  
   as instruction, 22, 85  
   in strings, 124
- Re-using information, 22, 85
- RECEIPTS, 220
- Record pattern, 54, 57  
   definition, 252  
   in a separate file, 58, 101  
   order of items, 60  
   planning and creating, 59  
   terminator, 57, 63
- Records, 12, 54, 57  
   definition, 252
- REMINDER, 224
- REPEAT.CND, 173
- REPEAT.NUL, 170
- Repeating groups of instructions  
   building blocks, 267  
   outside program units, 177  
   using program units, 162
- Repeating program units  
   a specific number of times, 167  
   controlled by conditional instructions, 171  
   problem solving, 175  
   until end of data file, 170  
   until null or zero typed, 169
- RESPONSE, 237
- SAMPLE.EG, 88
- Save and print result, 32, 114
- Save result, 31, 113
- Searching through a data file  
   (building block), 264
- SELECT.EG, 128
- Separators, 13, 57  
   acceptable characters, 61  
   as end-of-record markers, 68  
   choosing suitable separators, 61  
   definition, 252  
   return return, 63
- Signed numbers, 150
- Spaces  
   between item-names, 62
- SPECIFIC.LST, 195
- Stored information, 22, 26  
   combining, 153  
   lifetime of typed information, 55  
   options for changing, 55  
   storing data from keyboard, 85  
   using in arithmetic, 148
- String concatenation, 153
- Strings, 87, 117, 180  
   definition, 256  
   nested strings, 180  
   wildcards in strings, 124
- STRINGS.EG, 156
- SUCCESS.EG, 131
- Tabs  
   between item-names, 62
- Terminator, 57, 63  
   definition, 252
- Testing  
   conditional instructions, 125  
   data files, 73  
   documents, 94, 111, 187  
   perform instructions, 161  
   Text strings, 87  
   Thousands separator, 151  
   TOTAL.VAT, 231  
   Troubleshooting, 273
- Typing in information, 18, 102, 109  
   adding styling codes, 19, 104  
   correcting errors, 18  
   finishing a response, 18  
   multi-line information, 19, 103  
   when response is rejected, 103, 110
- Unit code (in a data file), 63
- UNPAID.INV, 228
- UPDATE.EG, 204
- VAT calculations, 150  
   building blocks, 263
- Wildcard characters, 124
- ! in format, 150  
 ! instruction, 24, 86  
 \$+ instruction, 129  
 \$- instruction, 130  
 \$= instruction, 255  
 \$\$ instruction, 255  
 % instruction, 160  
 %@ instruction, 163  
 @% instruction, 163  
 ? instruction, 22, 85  
 ? as wildcard, 124  
 ?# instruction, 85  
 , in format, 151  
 = instruction, 27  
 <- as command separator, 92  
 \* as arithmetic operator, 147  
 \* as wildcard, 124  
 \* in strings, 124  
 \* instruction, 126  
 + as arithmetic operator, 147  
 + as instruction, 143  
 - as arithmetic operator, 147  
 - as instruction, 143  
 / as arithmetic operator, 147  
 / in format, 151  
 \ in format, 151  
 | in arithmetic, 149  
 < instruction, 120  
 > instruction, 120  
 [...] instruction, 147