

**Reality**  
V10.0

---

**General Utilities and Printing**

All trademarks including but not limited to brand names, logos and product names referred to in this document are trademarks or registered trademarks of Northgate Information Solutions UK Limited (Northgate) or where appropriate a third party.

This document is protected by laws in England and other countries. Unauthorised use, transmission, reproduction, distribution or storage in any form or by any means in whole or in part is prohibited unless expressly authorised in writing by Northgate. In the event of any such violations or attempted violations of this notice, Northgate reserves all rights it has in contract and in law, including without limitation, the right to terminate the contract without notice.

© Copyright Northgate Information Solutions UK Limited, 2003.

Document No. UM70006045F  
August 2003

Northgate Information Solutions UK Limited  
Boundary Way  
Hemel Hempstead, Hertfordshire,  
HP2 7HU

Tel: +44 (0)1442 232424

Fax: +44 (0)1442 256454

[www.northgate-is.com](http://www.northgate-is.com)

# Comment Sheet

Please give page number and description for any errors found:

Page	Error

Please use the box below to: describe any material you think is missing; describe any material which is hard to understand; enter any suggestions for improvement; provide any specific examples of how you use your system which you think might be useful to readers of this manual

Continue on a separate sheet if necessary.

If you would like someone to contact you about documentation, please tick this box.

*Important:* Please enter your name, address and telephone number on the back of this form before returning.

**Manual:** Reality General Utilities and Printing,  
UM70006045F, August 2003.

Name of Sender: \_\_\_\_\_

Company Name / Address: \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

Telephone: \_\_\_\_\_

Date: \_\_\_\_\_

FOLD B

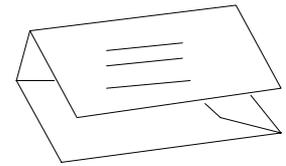
BUSINESS REPLY SERVICE  
Licence No. HH698

2



F  
O  
L  
D  
A

Technical Publications Department  
Northgate Information Solutions UK Limited  
Boundary Way  
Hemel Hempstead  
Hertfordshire HP2 7HU



Fold along fold A,  
then B and C. Tuck in  
so name and address  
are on outside.

FOLD C

---

# Contents

## Chapter 1 About this Manual

Purpose of this Manual.....	1-2
Contents .....	1-3
References .....	1-4
Conventions.....	1-5
User Comments.....	1-7

## Chapter 2 User Exits and Utilities

Introduction.....	2-2
Verb Utilities;.....	2-2
User Exits .....	2-2
Contents of this Chapter .....	2-2
Relocated Utilities.....	2-4
Withdrawn Utilities .....	2-5
Obsolescent Utilities .....	2-7
Summary of Current Utilities .....	2-9
Proc Utilities.....	2-12
Character Set Conventions .....	2-12
Definitions .....	2-12
U50BB - Get Process and Account .....	2-14
U10DD - Get System Serial Number.....	2-14
U315E - Number of Ports .....	2-14
U0190 - Polish String Evaluation.....	2-14
Straight-through Printing.....	2-16
U119B - Straight-through On.....	2-17
U219B - Buffer Printer Output To Serial Printer .....	2-17
U319B - Straight through Off.....	2-17
U41A4 - Check Tape Attached.....	2-18
U147F - Get System Time/Date .....	2-19
U247F - Get Line Number and Account Name .....	2-20
U347F - Error Handling Routine .....	2-20
U447F,U547F - Disable/Enable Terminal Break Key.....	2-21
U647F,U747F - Clear/Set R Flag .....	2-21

- U847F - Get Number of Items Selected..... 2-22
- U44AF - Number of Tape Units ..... 2-22
- English Utilities..... 2-23
  - U319F - Running Total of Multi/Sub Values ..... 2-23
  - U419F - Running Totals of Attributes/Values ..... 2-23
  - UA070 - Referencing a Specified Multi-Value ..... 2-24
- DataBasic Utilities ..... 2-25
  - U40E0 - Toggle Warning Message Printing ..... 2-25
  - U31A4 Strip/pad Trailing Spaces..... 2-25
  - Overpunched Numeric Fields ..... 2-26
  - U01A7 - Overpunched To/From System Numeric..... 2-26
  - U11A7 - Overpunched To/From System Numeric..... 2-27
  - U21A7 - Overpunched To/From System Numeric..... 2-27
  - U41AC - Input and Reduce Characters ..... 2-27
  - U047F - Strip/Pad Leading Zeros ..... 2-28
- Conversion Utilities ..... 2-29
  - U019F - Full Stop <-> Comma <-> Space ..... 2-29
  - U119F - Full Stop <-> Comma..... 2-30
  - U519F - <> (MD conversion) <-> ()..... 2-31
  - U919F - Decimal/Hexadecimal -> Binary..... 2-31
  - U41A9 <> (MD Conversion) to () ..... 2-32
  - U31AC - Reduce Repeated Chars To One..... 2-32
  - U51AA - Reversing a String for Output..... 2-33
- Verb Utilities..... 2-35
  - Table Input ..... 2-36
  - TABLE-LOAD Verb ..... 2-38
  - TABLE-CHECK Verb ..... 2-38
- ICL Utilities..... 2-39
  - ICL Read/Write-1900 Verbs..... 2-39
  - Use of Files ..... 2-39
  - WRITE-1900 Verb..... 2-39
  - READ-1900 Verb ..... 2-40
  - Options..... 2-40
  - Label and Sentinel Processing ..... 2-41
  - Guidelines for Handling Tape Files..... 2-44
  - ICL Read and Write 1900 Routines ..... 2-45

U0200 - Read/Write Data Block .....	2-45
U0201/ U1201 - Read/Write Start-of-Data or End-of-Data Sentinel .....	2-47
Guidelines for Label and Sentinel Processing.....	2-48
Guidelines for Programs Handling Data .....	2-48
U2201 - Access/Update Current Tape Block Size .....	2-49
ICL 1900 General Utilities.....	2-50
U01A4 - ASCII <-> ICL 1900 7-Track Code .....	2-50
U11A4 - Decimal <-> ICL 1900 Binary .....	2-50
ICL READ/WRITE - 2900 Verbs.....	2-52
WRITE-2900 Verb .....	2-55
READ-2900 Verb.....	2-56
Options .....	2-57
Error Messages .....	2-59
Bull Utilities .....	2-60
U219A - ASCII <-> Bull 7-Track Code.....	2-60
U319A - ASCII <-> Bull 7-Track Code.....	2-60

### Chapter 3 Terminal Printing Techniques

Introduction.....	3-2
Terminal Executive .....	3-2
Printing Facilities .....	3-3
Print Screen.....	3-3
Direct Printing .....	3-3
Straight-through Printing.....	3-3
Hardcopy Printing .....	3-3
Autoprint .....	3-3
Spooled Printing .....	3-4
Print Screen.....	3-6
Direct Printing .....	3-8
Straight-through Printing .....	3-10
Line Protocol Codes .....	3-11
Sample Programs.....	3-12
Hardcopy Printing.....	3-20
Autoprint .....	3-21
Spooled Printing .....	3-24

## Chapter 4 Printer Programming

Introduction .....	4-2
Sending Control Codes to a Printer .....	4-2
Sending Escape Sequences to a Printer .....	4-2
Variables in Escape Sequences .....	4-2
Model 11115/11115-X and 11130 150 and 300 LPM Line Printers .....	4-3
Control Codes .....	4-3
Cataloged Page Length/EVFU Program.....	4-3
To Load the EVFU from Program .....	4-4
To Utilise the EVFU from Program .....	4-5
Changing the Number of Lines Per Inch.....	4-6
Printable Characters .....	4-6
Model 11161/11161-X 600 LPM Line Printers.....	4-7
Control Codes .....	4-7
Cataloged Page Length/EVFU Program.....	4-7
To Load the EVFU from Program .....	4-8
To Utilise the EVFU from Program .....	4-10
Printable Characters .....	4-11
Model 11180/11180-X/11199/11199-X High Volume Line Printers.....	4-12
Control Codes .....	4-12
Cataloged Page Length/EVFU Program.....	4-12
To Load the EVFU from Program .....	4-13
To Utilise the EVFU from Program .....	4-15
Printable Characters .....	4-15
Printable Characters .....	4-17

## Appendix A Decimal, Hexadecimal, and ASCII Table

### Index

### List of Figures

Figure 2-1. Single File Structure .....	2-53
---	------

**List of Tables**

Table 2-1.	Obsolescent Utilities .....	2-7
Table 2-2.	User Exits.....	2-9
Table 2-3.	Verb Utilities .....	2-11
Table 3-1.	Keyboard Controlled Printing Facilities.....	3-4
Table 3-2.	Program Controlled Printing.....	3-5
Table 3-3.	Effecting Print Screen .....	3-6
Table 3-4.	Codes to Control Direct Print .....	3-8
Table 3-5.	Controlling Straight-through Printing.....	3-11
Table 3-6.	Controlling Hardcopy Printing .....	3-20
Table 3-7.	Controlling Autoprint .....	3-22

---

# Chapter 1

## About this Manual

This chapter describes the different sections of this manual and any conventions used.

## **Purpose of this Manual**

This manual describes special facilities available to Reality software users that are not described elsewhere in the Reality reference manuals. They include:

- User exits
- Printers attached to terminals
- Special programming for system printers

## Contents

**Chapter 2, User Exits and Utilities**, describes utilities that can be called from Procs, English dictionaries, DataBasic programs and/or as verbs to perform tasks that would otherwise be difficult, inefficient or inconvenient to execute.

**Chapter 3, Terminal Printing Techniques**, describes techniques available for output to printers attached to terminals, including: Print Screen, Spooled Printing, Direct Printing, Hardcopy Printing, Straight-through and Autoprint (those supported depend on the model of terminal.).

**Chapter 4, Printing Programming**, describes the programming interface from DataBasic and Proc for control of selected printers.

**Note:** Programming information for printers not covered by this chapter is given in the relevant printer manual.

**Appendix A, Decimal Hexadecimal and ASCII Table.**

## References

The following manuals contain relevant information:

*Reality Reference Manual Volume 1: General*

*English Reference Manual*

*Proc Reference Manual*

*DataBasic Reference Manual*

For a complete list of Reality reference manuals together with their document numbers, refer to the *Document Directory* in *Reality Reference Manual Volume 1*.

## Conventions

This manual uses the following conventions:

**Text**                    Bold text in this typeface shows input to be typed at the terminal.

Text                      Text in this typeface shows text that is output to the screen.

**Bold text**              Bold text in syntax descriptions represents characters typed exactly as shown. For example

**WHO**

*text*                      Characters or words in italics indicate parameters which must be supplied by the user. For example in

**LIST** *file-specifier*

the parameter *file-specifier* is italicized to indicate that you must specify an actual file.

*Document Title*        Italic text also indicates titles of documents referenced.

{Braces}                 Braces enclose options and optional parameters. For example in

**BLIST** *file-specifier item-id* {(options)}

- *file-specifier* and *item-id* must be supplied
- one or more single-letter options can be included, as defined for the command; these must be preceded by an open parenthesis, can be given in any order, and are not separated by spaces. Any number of options can be used except where specified otherwise in text.

[param | param]        Parameters shown separated by vertical lines within square brackets in syntax descriptions indicate that at least one of these parameters must be selected. For instance,

[**THEN** *statements* | **ELSE** *statements*]

	indicates that either a THEN clause or an ELSE clause must be included (or both).
...	In syntax descriptions, indicates that the parameters preceding can be repeated as many times as necessary.
<i>file-specifier</i>	The parameter <i>file-specifier</i> represents the following:  <i>{file-modifier} filename{,data-section-name}</i> <ul style="list-style-type: none"><li>• <i>file-modifier</i> can be <b>DICT</b> to indicate the dictionary section of the file. With appropriate English Query Language verbs, it can also be <b>ONLY</b>, <b>TAPE</b>, or <b>WITHIN</b>.</li><li>• <i>data-section-name</i> can be used to specify a data section other than the default. In this case, <b>DICT</b> must be omitted.</li></ul>
SMALL CAPITALS	Small capitals are used for the names of keys such as RETURN.
CTRL+X	Two (or more) key names joined by a plus sign (+) indicate a combination of keys, where the first key(s) must be held down while the second (or last) is pressed. For example, CTRL+X indicates that the CTRL key must be held down while the X key is pressed.
Enter	To enter means to type text then press RETURN. For instance, 'Enter the WHO command' means type WHO, then press RETURN.  In general, the RETURN key (shown as ENTER or ↵ on some keyboards) must be used to complete all terminal input unless otherwise specified.
Press	Press single key or key combination but do not press RETURN afterwards.
X'nn'	This denotes a hexadecimal value.

## User Comments

A Comment Sheet is included at the front of this manual. If you find any errors or have any suggestions for improvements in the manual please complete and return the form. If it has already been used, then send your comments to the Technical Publications Manager at the address on the title page, or email [techpubs@northgate-is.com](mailto:techpubs@northgate-is.com).

---

## **Chapter 2**

# **User Exits and Utilities**

This chapter describes the utilities that can be called from Procs, English dictionaries, DataBasic programs and/or as verbs to perform tasks that would otherwise be difficult, inefficient or inconvenient to execute.

## Introduction

User exits enable you to perform tasks that would otherwise be difficult to implement or inefficient in execution. Alternatively, some utilities provide convenience and ease of system operation.

### Verb Utilities

Some utilities are provided as verbs. In these cases different functions can sometimes be called by option characters. You use these verbs in the same way as ordinary TCL verbs.

### User Exits

Utilities provided for use in Procs and English conversions and DataBasic programs are known as 'user exits'. These are called by specifying a unique four digit hexadecimal number prefixed by the letter U.

### Contents of this Chapter

This chapter provides detailed information on utilities under the following main headings:

- [\*Relocated Utilities\*](#)

Lists a number of user exits that have been moved to new locations.

**Note:** Ensure that, if you have used any of the utilities listed here as 'relocated', you replace them with the new identity.

- [\*Withdrawn Utilities\*](#)

Lists a number of user exits that have been withdrawn over a period of time as their functionality has become obsolete or available within the standard software.

**Note:** Ensure that if you have used any of the utilities listed here that you remove them and replace them with the equivalent standard functions.

- [\*Obsolescent Utilities\*](#)

Lists a number of user exits that are still available, but have been completely or partially replaced in functionality by new standard functions. Also detailed is the standard Reality functions that should be used in stead.

Those which have been completely replaced in functionality are not now described in detail. If you need this information, you must keep your old Reality documentation.

- [\*Summary of Current Utilities\*](#)

This heading precedes two tables listing all currently available utilities.

In the first table, each user exit's identity is given with a brief description and an indication of the language(s) from which it is used.

In the second table, each verb utility's command name is listed with a brief description.

- [\*Proc Utilities\*](#)

Contains detailed information about Proc-only utilities.

- [\*English Utilities\*](#)

Contains detailed information about English-only utilities.

- [\*DataBasic Utilities\*](#)

Contains detailed information about DataBasic-only utilities.

- [\*Conversion Utilities\*](#)

Contains general and specific information about Conversion Utilities; that is, utilities which can be called from Proc, English or DataBasic.

- [\*Verb Utilities\*](#)

Contains detailed information about general verb and associated utilities.

- [\*ICL Utilities\*](#)

Contains detailed information about various types of utility specifically designed for interfacing with ICL tape formats.

- [\*Bull Utilities\*](#)

Contains detailed information about various types of utility specifically designed for interfacing with Bull tape formats.

## Relocated Utilities

The following user exits have been relocated. Attempting to execute any of these via the old identity will result in an illegal opcode abort. If you are still using the old location identifiers you must change to the new ones as follows:

Old ID	Utility	New ID
U819F	Get Number Of Items Selected	<a href="#">U847F</a>
U21A4	Strip/pad Leading Zeros On Tape Records	<a href="#">U047F</a>
U21A8	Get System Time/Date	<a href="#">U147F</a>
U31A8	Get Line Number + Account Name	<a href="#">U247F</a>
U51A8	Error Handling Routine	<a href="#">U347F</a>
U61A9	Disable BREAK Key	<a href="#">U447F</a>
U71A9	Enable BREAK Key	<a href="#">U547F</a>
U81A9	Set "R" Flag	<a href="#">U747F</a>
U91A9	Clear "R" Flag	<a href="#">U647F</a>

**Notes:**

1. These utilities are documented under the new identities shown.
2. Utilities U647F and U747F were documented incorrectly on some previous Reality releases.

## Withdrawn Utilities

The following utilities are no longer supplied. They are either obsolete or replaced by features added to the software over the last few years. Any attempt to execute one of these will result in an 'Illegal Opcode' abort. Therefore, you should rework any Procs, DataBasic programs or English dictionary attribute definition items which still use them.

**Note:** User exits U21AC and U947F were withdrawn on RealityX Release 3.1. U947F, if entered, returns zero. All other user exits in the list were withdrawn on Reality Release 7.0 or earlier, and are also not available in RealityX and Reality 8.0 or later.

U3070	Get PCB-FID
U0079	Get PIB status/return stacks
U0191	Set special lock byte
U1191	Unset special lock byte
U2191	Unset special lock byte
U3191	Unset special lock byte
U0192	Output formatting
U0198	European date format
U01A2	N-way branch on value
U11A2	Pad secondary input value
U21A2	Delete items
U31A2	Limited subroutine capability
U41A2	Limited subroutine capability
U51A2	Limited subroutine capability
U61A2	Alternate system LISTFLAG
U91A2	Proc to Proc control transfer
UA1A2	Delete values
U01A5	Upper case to lower case
U01A6	Display and cursor control
U11A8	Concatenate parameters

U41A8	Parameter comparison in PQ Proc
U61A8	Provides formatted error routines
U51A9	Enable/disable BREAK key
U21AC	Translate using pre-defined table
U01AD	Get system serial number
U947F	Boot device unit number
U148C	Spooler line number

## Obsolescent Utilities

A number of DataBasic and Conversion utilities have been replaced by equivalent functionality within English, Proc, or DataBasic. These are still supported but it is recommended that you use the new equivalent functions rather than the utility in any new functions you create.

Table 2-1 lists each obsolescent utility with an indication of the new functions that replace it.

**Table 2-1. Obsolescent Utilities**

Utility	Function	Replacement
U307A	Set process to sleep until "time"	PERFORM "SLEEP HH:MM"
U407A	Set process to sleep for 'n' seconds	PERFORM "SLEEP n"
U50BB	Get process and account	PORT = SYSTEM(18)
U10DD	Get system serial number	NUMBER = SYSTEM(22)<1>
U00E0	Check whether program cataloged	CAT = SYSTEM(20) (Result CAT = 1 if program cataloged, 0 if not.)
U10E0	Get run-time options	OPTIONS = SYSTEM(15)
U20E0	Get TCL Input Statement	STATEMENT=SENTENCE()
U30E0	Check for active select list	L.ACTIVE = SYSTEM(11) (Result 1 if list active, 0 if not.)
U60E0	Get terminal page width	TWIDTH = SYSTEM(2)
U70E0	Set terminal echo 'on'	ECHO ON
U80E0	Set terminal echo 'off'	ECHO OFF

Utility	Function	Replacement
U90EO	Concatenate character 't'	VALUE = "t": VALUE
U719F	Decimal <-> hexadecimal	Use MCDX or MCXD conversion, e.g. (in DataBasic): HEX = OCONV(DEC,"MCDX") DEC = OCONV(HEX,"MCXD")
U01A9	Strip Spaces from Variable	In DataBasic programs, use: STRING = TRIM(STRING," ","A") In English and Proc, use the conversion code MCC; ;" to change all spaces to nulls.
U11A9	Strip Specified Character	Similar to U01A9, but using character to be stripped instead of a space. For example, if character to be stripped is *, use in DataBasic: STRING = TRIM(STRING,"*", "A") In English/Proc: MCC:*;"
U21A9	Strip All Except Alphanumeric	Use MCB code. For example, in DataBasic: STRING = ICONV(STRING,"MCB")
U31A9	Replace Character with Another	In DataBasic, use: STRING = CHANGE (STRING, "X","Y") where X is the character to be replaced, Y is the character to replace it with. In English and Proc, use code MCC as follows: MCC;X;Y

## Summary of Current Utilities

The following tables summarise the currently supported utilities, showing where they are used and a reference to where you can find detailed information. Utilities are listed in frame number order - that is, in numeric order of the final three digits.

**Note:** If you are unfamiliar with the use of user exits you must read the beginning of the relevant section (refer to the section [Contents of this Chapter](#)) to find out about general principles and conventions.

**Table 2-2. User Exits**

User Exit	Identity and Description	DataBasic	English	Proc
<a href="#">UA070</a>	References a multivalued specified in an English enquiry			X
<a href="#">U50BB</a>	Get process number and account name	X		X
<a href="#">U10DD</a>	Get system serial number	X		X
<a href="#">U40E0</a>	Toggle warning message printing	X		
<a href="#">U315E</a>	Number of ports			X
<a href="#">U0190</a>	Polish string evaluation			X
<a href="#">U219A</a>	ASCII <-> Bull 7-track	X		
<a href="#">U319A</a>	ASCII <-> Bull 7-track	X		
<a href="#">U119B</a>	Straight-through printing "on"			X
<a href="#">U219B</a>	Buffer printer output to serial printer		X	
<a href="#">U319B</a>	Straight-through printing "off"			X
<a href="#">U019F</a>	Full stop <-> comma <-> space	X	X	X
<a href="#">U119F</a>	Full stop <-> comma	X	X	X
<a href="#">U319F</a>	Running totals of multi/sub-values		X	
<a href="#">U419F</a>	Running totals of attributes and multi/sub-values		X	

## Summary of Current Utilities

---

<b>User Exit</b>	<b>Identity and Description</b>	<b>DataBasic</b>	<b>English</b>	<b>Proc</b>
<i>U519F</i>	<> (MD conversion) <-> ()	X	X	X
<i>U919F</i>	Decimal/Hexadecimal -> Binary	X	X	X
<i>U01A4</i>	ASCII <-> ICL 1900 7-track code	X		
<i>U11A4</i>	Decimal <-> ICL 1900 Binary	X		
<i>U31A4</i>	Strip/pad trailing spaces on tape records	X		
<i>U41A4</i>	Check tape attached			X
<i>U01A7</i>	Overpunched to/from system numeric	X		
<i>U11A7</i>	Overpunched to/from system numeric	X		
<i>U21A7</i>	Overpunched to/from system numeric	X		
<i>U41A9</i>	<> (MD conversion) -> ()	X	X	X
<i>U31AC</i>	Reduce repeated characters to one	X	X	X
<i>U41AC</i>	Input and reduce characters	X		
<i>U51AA</i>	Reversing a string for output	X	X	
<i>U0200</i>	Read/write 9-track tape for use with ICL 1900 series	X		
<i>U0201</i>	Read/write start-of-data sentinel	X		
<i>U1201</i>	Read/write end-of-data sentinel	X		
<i>U2201</i>	Access/update current tape block size	X		
<i>U047F</i>	Strip/pad leading zeros on tape records	X		
<i>U147F</i>	Get system time/date			X
<i>U247F</i>	Get line number and account name			X
<i>U347F</i>	Error handling routine			X
<i>U447F</i>	Disable terminal BREAK key			X
<i>U547F</i>	Enable terminal BREAK key			X

<b>User Exit</b>	<b>Identity and Description</b>	<b>DataBasic</b>	<b>English</b>	<b>Proc</b>
<i>U647F</i>	Clear "R" flag			X
<i>U747F</i>	Set "R" flag			X
<i>U847F</i>	Get number of items selected			X
<i>U44AF</i>	Number of tape units			X

**Table 2-3. Verb Utilities**

<b>Verb</b>	<b>Function</b>
<i>TABLE-LOAD</i>	Load ICONV/OCONV translation tables
<i>TABLE-CHECK</i>	Check whether translation tables loaded
<i>WRITE-1900</i>	Write tape for use with ICL 1900 series
<i>READ-1900</i>	Read tape for use with ICL 1900 series
<i>WRITE-2900</i>	Write tape for use with ICL 2900 series
<i>READ-2900</i>	Read tape for use with ICL 2900 series

## Proc Utilities

'Proc only' utilities operate within the framework provided by Stored Procedures (Procs). The Proc concept is extended in that the data source for an operation can be the user database, the primary or secondary output buffers, the primary input buffer, or a special file which is uniquely associated with each port on the CPU. The target (the destination of the result) for an operation can be either output buffer, the primary input buffer, the user's terminal, or the private file.

See the topic [Conversion Utilities](#) for additional routines available from Proc.

**Note:** SYSTEM function can now be used from Proc to obtain system parameters and should be used in preference to user exits. For example SYS(18) returns the current port number and SYS(19) returns the current account. Equivalent user exits documented here are described for maintenance purposes only.

### Character Set Conventions

The following characters have special meanings in Proc:

- % (percent sign) flags a reference to a primary input buffer value
- # (hash sign) flags a reference to a value in the current output buffer (primary if STOFF ('stack off'), secondary if STON ('stack on'))
- ! (exclamation mark) flags a reference to a select buffer
- & (ampersand) flags a reference to a file buffer

These characters should not be used as the initial character of a file name, item-id or attribute name.

### Definitions

The following definitions are used in the detailed descriptions of the Proc utilities:

#### Current Output Buffer

The primary output buffer if selected by a STOFF command; the secondary output buffer if STON was the last buffer selection command.

**Alternate Output Buffer**

The secondary output buffer if STOFF was the most recent buffer selection command; the primary output buffer if STON was last issued. In other words, the currently inactive output buffer.

**Indirect Reference**

An argument in the form:

%nn                    references a value in the primary input buffer.

#nn                    references a value in the current output buffer.

**Target Designation**

The following notation identifies the target or destination for a process:

A = alternate output buffer.

P = primary input buffer.

S = current output buffer.

T = the user's terminal.

V = verifies the item, but does not produce output.

**Substitute Blank Option**

Any of the target designators may be prefixed by the letter 'B' to specify that embedded blanks should be translated to substitute blank (backslash) characters. This is an extremely useful facility when the target is P, S or A.

**Verify Attribute Option**

VA is a valid target as well as V. VA verifies that the attribute is non-null.

**Carriage Return/Line Feed Option**

When the terminal is the specified target (T), carriage return/line feed can be suppressed by appending the character '+' to the target designator (T+).

**Summary of Target Designators**

The following are all valid target designators:

A, BA, P, BP, S, BS, T, BT, T+, BT+, W, BW, V, BV, VA, BVA

### U50BB - Get Process and Account

This routine loads the process number and the account name from which the Proc is being executed into the currently active input buffer. The following Proc command loads process number and account, separated by a space, into the current input buffer parameter:

```
IBH%n:U50BB:
```

If IH is used instead of IBH, account name is loaded into the current parameter and process into the next parameter. In either case, the parameter number *n* is ignored.

### U10DD - Get System Serial Number

This routine loads the system's serial number into the current input buffer parameter. The relevant Proc command is:

```
IBH%n:U10DD:
```

The parameter *n* is ignored.

### U315E - Number of Ports

This routine returns the number of actual ports on the system, that is the number that are physically connected as opposed to the maximum number that the configuration allows. The number is placed in the currently selected input buffer.

#### Example

```
TEST
001 PQN
002 S1
003 U315E
004 T "NUMBER OF PORTS IS ",%1
005 RTN
```

### U0190 - Polish String Evaluation

This utility executes arithmetic, logical and relational operations on a 'Polish string' in the current output buffer of the Proc. The result of the computation can be directed to the terminal, to the primary input buffer, or to either of the output buffers.

The Polish string must be delimited to the left by an arbitrary character which is specified when calling U0190, and to the right by a '?' character.

U0190 evaluates the Polish string, produces the result and directs it to the required destination. At the same time it deletes the Polish string, including delimiters, from the current output buffer.

The destination of the result is known as the 'target'.

### Principle of Operation

U0190 implements a stack arithmetic machine. The character string passed to U0190 consists of operands (numerical values) and operators (see below). Each operand implies a 'LOAD STACK' command which places the value on the top of a push-down stack. Each operator processes one or two values from the top of the stack, and returns the result to the stack. Operators are either unary (the ^ operator - an Attribute Mark) or binary (all other operators). A unary operator replaces the top value in the stack with the result of operating on that value. A binary operator processes the top two stack values and then (1) - deletes the top value from the stack, and (2) - replaces the second value with the result of the operation. The string:

2 3 + 6 \* ?

would result in the following operations:

String Character Operation Stack Values

2 LOAD S1 2

3 LOAD S1 3

S2 2

+ ADD S1 3 + 2 = 5

6 LOAD S1 6

S2 5

\* MULTIPLY S1 6 \* 5 = 30

? ASSIGN S1 30 assign

to target

### Call-Up Sequence

U0190 is invoked by the following calling sequence:

Line n                    U0190

Line n + 1                left-end-delimiter target

**Definition**

Line n	is any line in Proc
Line n + 1	is the line in Proc immediately following Line n.
U0190	is the call-up code
left-end-delimiter	is any single character which defines the start of the Polish string in the output buffer
target	is the required destination of the result. Specified by one of the following characters:
A	is the alternate output buffer.
S	is the current output buffer.
P	is the primary input buffer.
T	is the terminal display.

**Example**

```
...  
007 U0190  
008 @ P  
...
```

Content of current output buffer = @ 6 3 + 4 \* ?

Polish string in current output buffer evaluates to 36. Polish string and delimiters deleted from current output buffer and primary input buffer assigned value of 36.

**Straight-through Printing**

There are three routines which together provide an enhanced facility for the operation of a Terminal Printer (a printer directly connected to a Video Display Terminal). The main advantage of using these routines is that the terminal can be operated at a higher line speed (baud rate) than the printer that is connected to it. Under normal circumstances, the printer could lose characters unless the line speeds are set to the same rate.

A suitable terminal executive must be loaded into the terminal before attempting to print.

The three routines are:

- |       |  |
|-------|--|
| U119B | This routine must be called from a Proc and is used to turn the Straight-through printing function ON.   |
| U219B | This routine is an English utility but it is included here for completeness. It is called via an attribute definition item in the file dictionary and ensures that only the correct amount of data to fill the buffer (as specified in the Proc) is sent to the printer at any one time. |
| U319B | This routine must be called from a Proc and is used to turn the Straight-through printing function OFF.  |

### **U119B - Straight-through On**

This routine must be called from a Proc. Its effect is to send X'16' to the terminal. This turns the Straight-through function ON, disables the terminal's screen and directs all subsequent data (until receipt of X'17') to the printer.

### **U219B - Buffer Printer Output To Serial Printer**

This routine is called from one of the attribute definition items (in the file dictionary) used to define an attribute to be printed. The call-up code is placed in the V/CONV attribute of the attribute definition item. If multi-valued fields are to be printed then the call-up code must be placed in one of these attribute definition items.

U219B employs a parameter which is stored in the Proc on the line following the call-up code for U119B (see example below). The parameter is prefixed by the letter C and comprises a one, two or three digit decimal number which defines the maximum number of print lines which can be handled at any one time (a buffer-full of data). Once this amount of data (normally about 1500 bytes) has been sent to the printer, the CPU waits for confirmation that the buffer has been emptied before sending the next buffer-full and so on. This sequence of events is repeated until all the data has been output. Proc processing then moves to the next line but the straight-through function remains ON until U319B is called.

**Note:** A dummy dictionary definition item containing an AMC of 99 must not be used to contain the U219B call-up code.

### **U319B - Straight through Off**

This routine must be called from a Proc. Its effect is the same as depressing the F3 Function Key on the keyboard - the CPU sends X'17' to the terminal. Having sent the X'17' code, the CPU enters a 'wait' condition. The X'17' is eventually processed as the last character in the buffer and the terminal responds by sending X'15' back to the CPU.

The Straight-through function is then turned OFF and the CPU continues processing the next line of the Proc.

**Example**

```
STOCK-REPORT
001 PQN
002 MV #1 "SORT STOCK BY DESCRIPTION PART
      DESCRIPTION QTY (N)"
003 U119B
004 C10
005 P
006 U319B
```

Line 3 calls up U119B which turns the Straight-through function ON. U219B is called during report formatting and controls the output of data according to the parameter specified in Line 4. This line specifies that 10 print lines will comprise one buffer-full of data (about 1500 bytes). Line 5 causes Line 2 to be processed and the data to be output under the control of U219B. Line 6 calls up U319B and turns the straight-through function OFF.

Any Proc which uses this utility must be in exactly the format shown above (line 3 onwards) or it aborts with the message:

```
INVALID EXIT FORMAT
```

Line 5 in the above example must not be changed to PP or the same message is output.

**U41A4 - Check Tape Attached**

This utility checks whether or not the tape is attached to your port. If it is attached, the next line of the Proc is skipped, otherwise the Proc continues as normal.

**Example**

```
001 PQN
002 HT-ATT
003 P
004 U41A4
005 XTRY AGAIN LATER!
006 HSELECT OUTPUT-FILE
007 STON
008 HRUN PROGFILE TAPE-OUTPUT-PROGRAM
009 P
```

If the T-ATT is successful, the line immediately following the call-up code (005) is skipped. If the T-ATT is not successful, the line immediately following the call-up code (005) is actioned.

In multi-tape environments the tape unit(s) acted upon are the default channel. If statement 002 in the above example is replaced by ASSIGN =TAPE 3 then this gives a default channel consisting of unit 3, and this unit is the one tested for tape attachment.

## U147F - Get System Time/Date

This routine obtains the current time or date, and places it in the primary input buffer. The time or date can be converted if required.

The Proc line immediately following the call-up code (U147F) is used to specify time or date, and any possible conversions. The Proc continues at the line following the specification line.

The format of the specification line is as follows. The first character must be T (for time) or D (for date). Subsequent characters are optional: if they are omitted, internal format is assumed. If present, the internal format is converted to external format. In this case, the T or D must be followed by a space followed by the desired conversion code(s). Multiple conversions must be separated by value marks.

The time or date replaces the parameter pointed to by the input buffer pointer. Note that several parameters may replace one parameter. The effect is analogous to the IH/NIH commands.

### Example

```
TEST
001 PQN
002 RI
003 C NOW PUT INTERNAL TIME IN PARAMETER 1
004 U147F
005 T
006 C NOW PUT CONVERTED TIME IN PARAMETER 2
007 S2
008 U147F
009 T MTHS
010 C NOW PUT INTERNAL DATE IN PARAMETER 3
011 S3
012 U147F
013 D
014 C NOW PUT CONVERTED DATE IN PARAMETERS
    4,5,6
015 S4
```

```
016 U147F
017 D D2
018 D0
019 RTN
```

The primary input buffer now contains:

```
44937^12:28:57PM^3224^28^OCT^89
```

### U247F - Get Line Number and Account Name

This routine obtains the current line number (terminal number, port number) and account name (user name) as in the 'WHO' TCL command, and replaces the current parameter in the primary input buffer with them. Note that two parameters replace one, that is %2 becomes %3, %3 becomes %4 and so on.

#### Example

```
TEST:
001 PQN
002 RI
003 U247F
004 D0
005 RTN
```

The primary input buffer now contains:

```
0^SYSPROG^
```

### U347F - Error Handling Routine

This routine makes reporting of operator errors in Procs easier and quicker. It sounds the alarm, displays an error message, and restarts the Proc at the correct line for re-entry of data.

The error message is placed on the line following the call-up code (U347F), as a comment. The routine sounds the terminal alarm, displays the message followed by a carriage return/line feed, then scans backwards through the Proc looking for a label ending in a zero; Proc execution is continued at this label.

#### Example

```
TEST
001 PQN
002 O
003 10 OENTER ACCOUNT NUMBER+
004 IN?
```

```
005 IF A # (6N) U347F
006 CMUST BE 6 DIGITS
007 XTHAT WAS OK
```

If an invalid account number is entered, the alarm is sounded, the message 'MUST BE 6 DIGITS' is displayed on the next line, and the prompt is repeated on the line after that.

### **U447F,U547F - Disable/Enable Terminal Break Key**

These utilities allow the BREAK key to be disabled and re-enabled from within a Proc. U447F disables the BREAK key, U547F re-enables the BREAK key.

#### **Example**

```
001 PQN
002 U447F
003 0 Break Key now disabled
004 .
    .
    .
031 U547F
032 0 Break Key re-enabled
033 RTN
```

### **U647F,U747F - Clear/Set R Flag**

The "R" (RSTRT) flag, if set, causes END to debugger to re-execute the Logon Proc and thus (with an appropriate Logon Proc) prevent access to TCL via BREAK and END. This is equivalent to R on line 9 of the account definition item in SYSTEM.

#### **Example**

```
001 PQN
...
006 C PREVENT ACCESS TO TCL VIA BREAK AND END
007 U747F
...
021 C ALLOW ACCESS TO TCL VIA BREAK AND END
022 U647F
...
```

## U847F - Get Number of Items Selected

This routine allows a Proc to access the count of items generated by an immediately-preceding SELECT command.

**Note:** The results obtained from a FORM-LIST or GET-LIST are unpredictable and, in most cases, incorrect.

### Example

```
:SELECT GUESTS WITH ROOM "3]"
```

```
5 ITEMS SELECTED
```

```
>REPORT
```

```
Third floor rooms occupied 5
```

The Proc 'REPORT' is as follows:

```
REPORT
001 PQN
002 S1
003 U847F
004 T "Third floor rooms occupied ",%1
```

## U44AF - Number of Tape Units

This user exit returns the tape unit number in a single tape deck environment or 0 if it is a multi-tape environment.

### Example

```
TEST
001 PQN
002 S3
003 U44AF
004 IF %3 #0 GO 10
005 T "Enter tape unit to use"
006 IBP?%3
007 10 MV #1 "ASSIGN =TAPE "**%3
008 P
009 RTN
```

## English Utilities

The following utilities can only be used via English dictionary items. See also the topic [Conversion Utilities](#).

### U319F - Running Total of Multi/Sub Values

This routine is commonly used as a conversion code in attribute 8 of a dictionary definition to enable a running total of Multi/Sub-values to be produced within one record.

#### Example

```
REC1      REC2
10]20]30  20]30]40
```

Using the call-up code U319F in the appropriate attribute of an item in the file dictionary, when a LIST verb for the file is entered, the output is as follows:

	UNCONVERTED	CONVERTED
REC1	10	10
	20	30
	30	60
REC2	20	20
	30	50
	40	90

### U419F - Running Totals of Attributes/Values

This routine is identical to U319F (above), except that the running total is carried over from one record to the next. Hence it can be used for running totals of attributes as well as Multi/Sub-values.

#### Example

Using the same example as given above for U319F, but this time employing the call-up code U419F, the output is as follows:

	UNCONVERTED	CONVERTED
REC1	10	10
	20	30
	30	60
REC2	20	80
	30	110
	40	150

**Note:** This user exit can only be used as a pre-processing conversion code in attribute 8 of the data definition item.

### UA070 - Referencing a Specified Multi-Value

This routine allows you to set up a data definition to access a particular multi-value number, as specified in an English sentence.

#### Example

For example an English enquiry such as:

```
LIST ORDERS ANY.VALUE "3"
```

lists the third value in attribute ANY.VALUE

where the data definition of ANY.VALUE is as follows:

```
001 A
002 0
003 Any ]Value
004
005
006
007
008 A;1(UA070) (1=attribute no.)
009 L
010 5
```

The A; 1(UA070) code takes attribute one (ANY.VALUE) and references the multi-value number three, as specified in the English enquiry.

## DataBasic Utilities

The following utilities can only be used from DataBasic. See also the topic [Conversion Utilities](#).

### U40E0 - Toggle Warning Message Printing

Toggles printing of warning messages during run-time (that is, toggles the 'S' option of the RUN verb.) For example:

```
DUMMY = ICONV(0,"U40E0")
```

### U31A4 Strip/pad Trailing Spaces

This routine was specifically written to ease processing of magnetic tape records in DataBasic programs.

On input conversion, it strips trailing spaces from fields. On output conversion, it pads variable-length fields with trailing spaces to a specified fixed length.

#### Example 1

If the value input to the field NAME is 'FRED '

```
X = ICONV(NAME, 'U31A4')
```

assigns the value 'FRED' to X.

#### Example 2

If the value of NAME is 'WILLIAM'

```
X = OCONV(NAME, 'U31A4,10')
```

assigns the value 'WILLIAM ' to X ready for output

#### Notes:

1. The field length specification (10 in the above example) can be separated from the mode-id by any character except 0-9 and A-F.
2. If the output value is longer than the specified field length, truncation occurs on the right. For example:

```
X = OCONV('ABCDEFGF', 'U31A4*5')
```

assigns the value 'ABCDE' to X.

### Overpunched Numeric Fields

This DataBasic only facility contains three separate routines for converting overpunched numeric fields to/from a System numeral. These routines are normally used in tape input/output applications when communicating with a mainframe. When used with the OCONV function of DataBasic, all the routines convert a System numeral (variable length with optional leading minus sign) to external overpunch format (fixed length with leading zeros and with the sign overpunched on the rightmost digit). When used with the ICONV function the reverse applies.

#### U01A7 - Overpunched To/From System Numeric

This routine performs the required conversion according to the following table:

	Positive	Negative
0	+	-
1	A	J
2	B	K
3	C	L
4	D	M
5	E	N
6	F	O
7	G	P
8	H	Q
9	I	R

#### Examples

`ICONV('0000123D','U01A7')` is '1234'

`ICONV('0009876N','U01A7')` is '-98765'

`ICONV('000+', 'U01A7')` is ''

`OCONV(2211, 'U01A7, 10')` is '00000221A'

`OCONV(-99, 'U01A7, 4')` is '009R'

**Note:** When using the OCONV function, a parameter to define the output length must be specified.

**U11A7 - Overpunched To/From System Numeric**

This routine performs the required conversion according to the following table:

	Positive	Negative
0	0	}
1	1	J
2	2	K
3	3	L
4	4	M
5	5	N
6	6	O
7	7	P
8	8	Q
9	9	R

**U21A7 - Overpunched To/From System Numeric**

This routine performs the required conversion according to the following table.

	Positive	Negative
0	?	}
1	A	J
2	B	K
3	C	L
4	D	M
5	E	N
6	F	O
7	G	P
8	H	Q
9	I	R

**U41AC - Input and Reduce Characters**

The function of this utility is to compact any specified character string so that any consecutively repeated characters are reduced to single character (for example AAABBC will be returned as ABC). The utility then automatically prompts for the next character string to be input (thus eliminating the need for a separate INPUT statement).

### Example

```
NAME = OCONV(35, "U41AC")
```

Instead of a direct or indirect reference to a predefined character string the maximum number of characters to be input is specified (35 in this example). The routine prompts for, and accepts, up to the maximum number of characters specified before performing the compression as defined in U31AC. Note that an input specification value of 0 allows up to 24 characters to be input.

### U047F - Strip/Pad Leading Zeros

This routine eases processing of magnetic tape records in DataBasic programs.

On input conversion it strips leading zeros from fields. On output conversion it pads variable-length fields with leading zeros to a specified fixed length.

#### Example 1

If the value input to the field NUMBER is '001205'

```
X = ICONV(NUMBER, 'U047F')
```

assigns the value '1205' to X.

#### Example 2

If the value of NUMBER is '1234'

```
X = OCONV(NUMBER, 'U047F,6')
```

assigns the value '001234' to X ready for output.

#### Notes:

1. The field length specification (6 in example 2 above) can be separated from the mode-id by any character except 0-9 and A-F.
2. If the output value is longer than the specified field length, truncation will occur on the left, for example,

```
X = OCONV('123456', 'U047F*4')
```

assigns the value '3456' to X.

## Conversion Utilities

Conversion utilities may be used in Procs, English or DataBasic programs. The method of use in each of these applications is shown in the following examples.

### Example 1: In a Proc

Use with the IH and IBH commands. For example,

```
...
004  IBH%1:U31AC:
...
```

where ":" performs output conversion (equivalent to the DataBasic function OCONV) and ";" performs input conversion (equivalent to the DataBasic function ICONV)

### Example 2: In an English Dictionary

Use in attribute 7 of dictionary items unless otherwise specified.

```
...
007  MD2,£<]U41A9
...
```

### Example 3: In a DataBasic program

Use with the ICONV or OCONV function. For example,  
`ICONV(DATA, "U21A9")`

**Note:** All examples given later are for DataBasic programs. However, all conversion utilities can also be used in Procs and in English as shown above.

## U019F - Full Stop <-> Comma <-> Space

When used in conjunction with the DataBasic functions OCONV and ICONV, or the equivalent in Proc or English, this routine has the following effects:

OCONV	Commas are replaced with spaces. Full stops (.) are replaced with commas.
ICONV	Commas are replaced with full stops (.). Spaces are replaced with commas.

**Example 1**

```
VALUE = '987,654.321'  
PRINT OCONV (VALUE,"U019F")
```

The output produced is 987 654,321

**Example 2**

```
VALUE = '987 654,321'  
PRINT ICONV (VALUE,"U019F")
```

The output in this case is 987,654.321

**Note:** This routine is usually used with an 'MD' conversion as follows:

OCONV - 'MD3,]U019F'

ICONV - 'U019F]MD3'

where ']' represents a value mark.

**U119F - Full Stop <-> Comma**

This routine has the following effects:

Commas are replaced with full stops (.).

Full stops (.) are replaced with commas.

**Example 1**

```
VALUE = '987,654.321'  
PRINT OCONV (VALUE,"U119F")
```

The output is 987.654,321

**Example 2**

```
VALUE = '987.654,321'  
PRINT ICONV (VALUE,"U119F")
```

The output is 987,654.321

**Note:** This routine is usually used with an 'MD' conversion as follows:

OCONV - 'MD3]U119F'

ICONV - 'U119F]MD3'

where ']' represents a value mark

**U519F - <> (MD conversion) <-> ()**

This routine is used to replace the negative balance indicators '<>', as generated by the 'MD' conversion, with the more conventional '()'. It has the following effects:

OCNV	'<' is replaced with '(' '>' is replaced with ')'
ICNV	'(' is replaced with '<' )' is replaced with '>'

**Example 1**

```
VALUE = '<987>'
PRINT OCNV (VALUE, "U519F")
```

Output is (987)

**Example 2**

```
VALUE=' (987) '
PRINT ICNV (VALUE, "U519F")
```

Output is <987>

**Note:** This routine is usually used with an 'MD' conversion as follows:

```
OCNV - 'MD3&<]U519F'
```

```
ICNV - 'U519F]MD3&<'
```

**U919F - Decimal/Hexadecimal -> Binary**

This routine accepts a decimal number (on OCNV) or a hexadecimal number (on ICNV) and returns the equivalent binary number.

**Example 1**

```
VALUE = 77
PRINT OCNV (VALUE, "U919F")
```

Output is 1001101

### Example 2

```
VALUE = '3F'  
PRINT ICONV(VALUE, "U919F")
```

Output is 111111

**Note:** If the number input (VALUE) contains an illegal or out-of-range character a null string is returned.

### U41A9 <> (MD Conversion) to ()

This routine is used in conjunction with the MD (Mask Decimal) conversion to replace the <...> sequence on negative amounts with (...).

```
VALUE = '-99999' ; VM = CHAR(253)  
.  
.  
.  
PRINT OCONV(VALUE, 'MD2,£<':VM: 'U41A9')
```

This would print as £(999.99)

The same code sequence can be used in a Dictionary definition, but only as a conversion:

```
007 MD2,£<]U41A9    (] = value mark)
```

### U31AC - Reduce Repeated Chars To One

This routine compacts any specified character string so that any consecutively repeated characters are reduced to a single character (for example AAABBC is returned as ABC). ICONV and OCONV have exactly the same effect.

#### Example

```
VALUE = 'AAABBCDEFFG!!!'  
PRINT OCONV(VALUE, "U31AC")
```

Output is ABCDEFG!

**Note:** In a DataBasic program, where the repeated characters are known, you can use:

```
STRING = TRIM(STRING, VAR, "R")
```

where VAR is the character to be reduced.

## U51AA - Reversing a String for Output.

This routine reverses a complete string, one character at a time.

### Example 1

If the value of STRING is '17AB]5041]AEF1'

```
INV.STRING= OCONV(STRING, 'U51AA')
```

assigns the value '1FEA]1405]BA71' to INV.STRING ready for output.

### Example 2

This second example shows how U51AA can be used as part of an English processing code to access the last value in an attribute.

For example, you could do this using the English enquiry:

```
LIST ORDERS LAST.VALUE
```

where LAST.VALUE has the following data definition:

```
001 A
002 0
003 Last ]Value
004
005
006
007
008 TORDERS;V;;2]U51AA]G 1]U51AA
009 1
010 5
```

The AMC, defined in attribute 2, must be set to 0 (zero) to reference the item-id LAST.VALUE.

The processing code then processes the string '17AB]5041]AEF1', as follows:

1. The T code (attribute 8) searches the ORDERS file for item-id LAST.VALUE and returns the value of attribute 2 (specified in the code) in LAST.VALUE. In this case, attribute 2 contains the string '17AB]5041]AEF1'. The affect of this is to translate the string back to the same file turning value marks into spaces. For example:

```
17AB]5041]AEF1 → 17AB 5041 AEF1
```

2. U51AA reverses the string:  
17AB 5041 AEF1 → 1FEA 1405 BA71
3. The group extraction 'G 1' extracts the first segment:  
1FEA 1405 BA71 → 1FEA
4. U51AA reverses the string:  
1FEA → AEF1

## Verb Utilities

TABLE-LOAD, TABLE-CHECK and U21AC are routines which together provide a complete facility for the translation (that is, conversion) of individual character codes or strings of character codes within a DataBasic program.

These routines employ two separate tables for the translation, or conversion, of submitted character codes. One table is used if the translation is specified via OCONV in DataBasic, the other if translation is via ICONV.

Both tables are initially created and stored as a single file item. The table used by OCONV is held between attributes 1 and 128. The table used by ICONV is held between attributes 129 and 256. Any character code in the decimal range 0 to 127 may be submitted for, or returned from, translation by either of the tables. Each attribute of the tables is used to hold one character code only.

Before being used in the translation process, the required pair of tables is loaded using the TABLE-LOAD verb. Any number of pairs of tables may be held on the system but only the pair currently loaded is actually used. Loading a new pair of tables (item) overwrites any existing tables. The original item remains untouched.

A means of checking whether the required pair of tables is already loaded is provided by the TABLE-CHECK.

**Note:** The verb definition items for TABLE-CHECK and TABLE-LOAD are entered via the EDITOR as described later.

The OCONV table (attributes 1 to 128) is accessed for the translation of a single or a string of character codes via an OCONV statement containing the call-up code U21AC. Each character in a submitted string (starting from the left) is dealt with separately and the result is returned to the program when the entire string has been translated.

The routine first reduces the submitted character to its decimal equivalent according to the ASCII standard. For example, the character A is reduced to decimal 65. This value is then incremented by 1 (to compensate for the fact that ASCII decimal codes start at 000 but the first attribute of the item must be attribute 001). The result of this addition provides the number of the attribute in which the translated code resides. Taking the example of the submitted character A: the decimal value of this character in ASCII is 65. This value is incremented by 1 and the result (66) is the attribute which contains the code to be returned to the program.

The ICONV table (attributes 129 to 256) is accessed via an ICONV statement containing the call-up code U21AC. The process used to arrive at the translated code is very similar to that used for the OCONV table, except that the ASCII decimal value of the submitted character is further incremented by 128 to access the second table. Taking the same example of a character A being submitted: the decimal value of 65 is incremented by 1 and then by a further 128 to give a total of 194. This attribute number then contains the code to be returned to the program.

It is recommended that the DataBasic program TABLE-INPUT should be entered and used to input the translation tables, rather than attempting to input them via the EDITOR.

## Table Input

The following program should be used to input each pair of translation tables required:

```
TABLE-INPUT
001   PROMPT ' '
002   REC = ' '
003   COD = "000"
004   PRINT @(-1)
005   PRINT @(10,2):"Reality CHARACTER CODE TRANSLATION TABLE
      INPUT PROGRAM"
006 5 PRINT @(20,5):SPACE(31):@(10,5): "FILE NAME: ":
007   INPUT FNAME,30_
008   IF FNAME EQ ' ' THEN STOP
009   OPEN ',FNAME TO F1 ELSE
010   PRINT @(4,19):"FILE DOES NOT EXIST - PRESS RETURN TO
      CONTINUE":
011     INPUT C,1
012     PRINT @(4,19):SPACE(50)
013     GOTO 5
014     END
015 10 PRINT @(20,6):SPACE(31):@(10,6): "ITEM NAME: ":
016   INPUT INAME,30_
017   IF INAME EQ ' ' THEN STOP
018   READ REC FROM F1,INAME ELSE GOTO 20
019     PRINT @(4,19):"ITEM ALREADY EXISTS - PRESS RETURN TO
      CONTINUE":
020     INPUT C,1_
021     PRINT @(4,19):SPACE(50)
022     GOTO 10
023 20 PRINT @(10,8):"PLEASE ENTER THE REQUIRED CODES IN
      DECIMAL FORM"
024   FOR I = 1 TO 128
025     PRINT @(30,10):@(-128)
026     PRINT @(4,10):@(-144):"ATTRIBUTE ":I:" (OCONV
      TABLE)":
027 25 PRINT @(5,12):"CODE TO BE RETURNED FOR A SUBMITTED
```

```

CODE OF "
028 PRINT @(49,12):STR('0',3 - LEN(COD)):COD:" : ":
029 INPUT IN1,3_
030 IF IN1 EQ "END" THEN STOP
031 IF NOT(IN1 MATCHES '0N') OR IN1 > 127 OR IN1 EQ ''
THEN
032 PRINT @(1,19):"CODE MUST BE NUMERIC IN DECIMAL
RANGE 0 TO 127 - PRESS RETURN TO CONTINUE"
033 PRINT @(1,20):"OR ENTER END AT ABORT PROGRAM:":
034 INPUT C,3_
035 IF C EQ "END" THEN STOP
036 PRINT @(1,19):SPACE(160) :@(55,12):SPACE(3)
037 GOTO 25
038 END
039 PRINT @(30,15):@(-128)
040 PRINT @(4,15):@(-144): "ATTRIBUTE ":I + 128:" (ICONV
TABLE) "
041 30 PRINT @(5,17):"CODE TO BE RETURNED FOR A SUBMITTED
CODE OF "
042 PRINT @(49,17):STR('0',3 - LEN(COD)) :COD:" : ":
043 INPUT IN2,3
044 IF IN2 EQ "END" THEN STOP
045DR IF NOT(IN2 MATCHES '0N') OR IN2 >127 OR IN2 EQ ''
THEN
046 PRINT @(1,19):"CODE MUST BE NUMERIC IN DECIMAL
RANGE 0 TO 127 - PRESS RETURN TO CONTINUE"
047 PRINT @(1,20):"OR ENTER END TO ABORT PROGRAM:":
048 INPUT C,3_
049 IF C EQ "END" THEN STOP
050 PRINT @(1,19):SPACE(160): @(55,17):SPACE(3)
051 GOTO 30
052 END
053 PRINT @(55,12):SPACE(420)
054 REC = REPLACE(REC,I,0,0,CHAR(IN1))
055 REC = REPLACE(REC,I + 128,0,0,CHAR(IN2))
056 COD = COD + 1
057 NEXT I
058 WRITE REC ON F1,INAME
059 PRINT @(36,22):"FINISHED"
060 END

```

The program first prompts for the file name and item name under which the tables are to be stored. The file must already exist and the item must not. The appropriate prompts are then displayed so that the codes (in decimal) may be input and, once the full 256 codes have been entered, the program will create the specified item. The FINISHED message indicates that the program has been successfully completed.

### TABLE-LOAD Verb

This routine is provided in the form of a TCL-II verb. You must create the following verb definition item in the appropriate Master Dictionary:

```
TABLE-LOAD
001 P
002 2
003 01AC note that the usual 'U' prefix is not used
004
005 U
```

The appropriate file and item names must be input with the verb. For example:

```
TABLE-LOAD TEST A
```

The item specified must not contain more than 256 attributes and each attribute must not contain more than one character code.

### TABLE-CHECK Verb

This routine is also provided as a TCL-II verb. You must create the following verb definition item in the appropriate MD:

```
TABLE-CHECK
001 P
002 2
003 11AC note that the usual 'U' prefix is not used
004
005 U
```

The file name and the item name of the tables to be compared with those already loaded must be specified with the verb in the normal manner. For example:

```
TABLE-CHECK TEST A
```

If the tables match then no indication is given. If the tables do not match then the message MATRIX MISMATCH - CHAR *n* is output. The char number *n* refers to the attribute number of the first character code that does not match. Checking ceases at the first mis-match.

## ICL Utilities

### ICL Read/Write-1900 Verbs

Two TCL-II verbs, READ-1900 and WRITE-1900 enable you to read/write from/to 9-track magnetic tapes for communication with the ICL 1900 Series of computers.

These verbs are only intended to provide a method of overcoming the technical difficulties imposed by the operating system; they should not be viewed as a special version of the ICL 1900 housekeeping routines. Thus the applications programmer is still responsible for the formatting of data (including labels and sentinels) to be written to tape, and for the interpretation of data (including labels and sentinels) read from tape. It is anticipated that the verbs will be used in conjunction with user-written Procs, TCL commands and DataBasic programs for each application.

In multi-tape systems all the tape operations refer to the default channel assignments, thus `T-ATT` implies unit 1 whereas `ASSIGN =TAPE 3` gives a default unit of 3.

**Note:** Named channel variable assignments, such as `ASSIGN X=TAPE 2`, do not work.

### Use of Files

The WRITE-1900 verb reads data from a file, converts it from ASCII to 1900 code, and writes it to tape. The READ-1900 verb reads data from tape, converts it from 1900 code to ASCII, and writes it to a file.

The items in the files have a format which is dictated purely by the tape format employed, so it is most unlikely that these files can be used for any purpose other than tape input or output. The files should therefore be thought of as spooling files; an application program will format data blocks and write them to an output file for later transcription to tape using WRITE-1900, or read and reformat blocks from a tape input file using READ-1900.

### WRITE-1900 Verb

WRITE-1900 is a TCL-II verb. This allows the user to specify the name of the file and the items to be written to tape.

Items are converted to 1900 code and written to tape in the order specified by the user. Each attribute generates one block on the tape. Item-ids are not written to tape.

## READ-1900 Verb

READ-1900 is also a TCL-II verb, but requires no item-list. The verb allocates numeric item-ids starting at 1, and increasing by 1 for every block read. Each block is read from tape, converted to ASCII, and written to the specified file as a one-attribute item.

## Options

The action of either verb can be modified by the use of options. Each option is a single letter or a decimal number. Some of these options are only used during testing.

Any combination of options of the following can be specified. If both D and T are specified, D is ignored. On writing to tape, if S is specified with D and/or T, the tape mark(s) follow the trailer label. On reading from tape, the verb terminates when the first end-of-tape condition (specified by D and/or T and/or S and/or decimal number) is encountered.

- |               |  |
|---------------|--|
| <b>N</b>      | Null Attribute. Interprets each null attribute as a tape mark. On writing, a null attribute causes a tape mark to be written. On reading, a tape mark generates an item with one null attribute. If N is not specified, null attributes are ignored when writing, and tape marks are not filed when reading. |
| <b>T</b>      | Tape Mark. Terminates the tape with a tape mark. On writing, the verb writes a tape mark after the last attribute of the last item. On reading, the verb terminates after reading a tape mark; note that the tape mark is not filed (even if N is specified).  |
| <b>D</b>      | Double Tape Mark. Terminates the tape with a double tape mark. On writing, the verb writes two tape marks after the last attribute of the last item. On reading, the verb terminates after reading a double tape mark. Note that the two tape marks are not filed (even if N is specified).                  |
| <b>I</b>      | Item-id. Displays each item-id on a new line as the item is read from or written to the file.  |
| <b>C</b>      | Count. Display a count of the number of blocks read from or written to tape, immediately prior to termination. This count includes tape marks (if the N option is specified), but does not include terminators specified by the D or T options.  |
| <i>number</i> | Decimal Number. Specifies number of blocks to be read from tape (valid only with READ-1900). The number of blocks is defined in the  |

---

same way as the count in the C option. Can be used in conjunction with D or T options.

- S** Sentinels. Special option to ease handling of tapes with data blocks introduced by a start-of-data sentinel and terminated by a trailer label. Writes a start-of-data sentinel before the first block (first attribute of first item) and a trailer label after the last block; the trailer label contains a count of data blocks written; or (on reading) expects a start-of-data sentinel before the first data block (if absent, an error message is printed and the verb terminates); reading stops when the trailer label is encountered (it is not filed); if the block count in the trailer label is incorrect, prints an error message.

### Label and Sentinel Processing

Processing is eased considerably by sticking to the simplest possible tape layout, that is, header label, start-of-data sentinel, data blocks, trailer label. In addition, all data blocks should be transferred to or from a single file at one time. In this case the S option can be used; otherwise, the sentinels and trailer label must be processed explicitly by user-written DataBasic programs.

In the second of the following methods, processing is considerably more complex. If the block count in the trailer label is to be validated, this will involve counting the items in the TAPEINPUT file, either when the contents of TAPEINPUT are processed (it's a bit late by then) or by a separate program (time consuming). As with reading, writing is more complex when S is not used. In cases where S cannot be used it is even more complex.

### Reading With S Option

1. Attach the tape unit: **T-ATT**
2. Rewind the tape: **T-REW**
3. Read header block into label file: **READ--1900 TAPELABEL (1,N)**
4. Validate contents of label file using DataBasic or Proc: **RUN PROGS READLABEL**
5. Clear the data file: **CLEAR-FILE (DATA TAPEINPUT)**
6. Read data into file: **READ-1900 TAPEINPUT (S,C)**
7. Rewind the tape: **T-REW**
8. Detach the tape unit: **T-DET**

### Reading Without S Option

1. Attach the tape unit: `ASSIGN =TAPE 2`
2. Rewind the tape: `T-REW`
3. Read header and sentinel into label file: `READ-1900 TAPELABEL (3,N)`
4. Validate contents of label file, using DataBasic or Proc: `RUN PROGS READLABEL`
5. Clear data file: `CLEAR-FILE (DATA TAPEINPUT)`
6. Read data into file: `READ-1900 TAPEINPUT (T,C)`
7. Read trailer label qualifier block into label file: `READ-1900 TAPELABEL (1,N)`
8. Rewind the tape: `T-REW`
9. Detach the tape unit: `T-DET`

### Writing With S Option

1. Attach the tape unit: `ASSIGN =TAPE 4`
2. Rewind the tape: `T-REW`
3. Read header block into label file: `READ-1900 TAPELABEL (1,N)`
4. Rewind the tape: `T-REW`
5. Read, validate, and rewrite contents of label file using DataBasic:  
`RUN PROGS WRITELABEL`
6. Write header block to tape: `WRITE-1900 TAPELABEL 1 (N)`
7. Select and sort items to be written to tape: `SSELECT TAPEOUTPUT`
8. Write data blocks to tape followed by 2 tape marks:  
`WRITE-1900 TAPEOUTPUT (S,D,C)`
9. Rewind the tape: `T-REW`
10. Check the tape: `T-CHK`
11. Rewind the tape: `T-REW`

12. Detach the tape unit: T-DET
13. Clear the data file: CLEAR-FILE (DATA TAPEOUTPUT)

### **Writing Without S Option**

1. Attach the tape unit: **T-ATT**
2. Rewind: **T-REW**
3. Read header block into label file: READ-1900 TAPELABEL (1,N)
4. Rewind: T-REW
5. Read, validate, and rewrite contents of label file, using DataBasic, as follows -  
item 1: header sentinel  
item 2: trailer: RUN PROGS WRITELABELX
6. Write header and sentinel to tape: WRITE-1900 TAPELABEL 1 (N)
7. Select and sort items to be written to tape: SSELECT TAPEOUTPUT
8. Write data blocks to tape: WRITE-1900 TAPEOUTPUT (T,C)
9. Write trailer label to tape followed by 2 tape marks: WRITE-1900 TAPELABEL 2 (D)
10. Rewind the tape: T-REW
11. Check the tape: T-CHK
12. Rewind the tape: T-REW
13. Detach the tape unit: T-DET
14. Clear the data file: CLEAR-FILE (DATA TAPEOUTPUT)

### **Summary**

Systems should be designed so that the S option can be used for sentinel and trailer processing. Although manual processing is available, it should be avoided if at all possible; or rather, it should be regarded as a facility for use by programmers only.

## Guidelines for Handling Tape Files

### Tape Output File

The item-ids must be unique. This is accomplished in the same way as for other files. If the items are to appear on the tape in a sorted sequence, the sort key is best built in to the item-id.

In most cases, items will contain only one attribute: the block to be written to tape. Where several blocks are to be output in a certain order (for example, a header record followed by several detail records), these can be filed as one multi-attribute item as long as the item size doesn't become excessive. The individual blocks are normally formatted using DataBasic.

The following DataBasic features and routines are particularly useful.

U047F	padding out numeric fields to fixed length
U31A4	padding out alphanumeric fields to fixed length
U11A4	generating binary fields
+24836/U11A4	generating ICL binary fields
LEN()/U11A4	generating word counts
;,CAT	concatenating fields

### Tape Input File

Items can be retrieved in the order of the tape blocks without using SELECT or SSELECT, by incrementing a numeric variable in a DataBasic program to be used as the item-id for READ statements. The following DataBasic features and routines are particularly useful for reformatting:

U047F	stripping leading zeros from numeric fields
U31A4	stripping trailing spaces from alphanumeric fields
U11A4	converting binary numbers to decimal
U11A4/24836	converting ICL dates to System dates
[]	extracting individual fields

### Restrictions

The following restrictions apply to all 1900 9-track handling:

1. The WRITE-1900 tape block size is restricted to the maximum size specified in the T-ATT or ASSIGN command. Any excess characters are ignored.
2. The READ-1900 tape block size is restricted to the maximum size specified in the T-ATT or ASSIGN command.

3. ASCII characters outside the range X'20' to X'5F' cannot be read from tape, and will write to tape as garbage.
4. Multi-reel files are not supported.
5. Word-counts and multi-record blocks must be handled explicitly by the application program.

### **ICL Read and Write 1900 Routines**

The following set of four routines enable the user to read and write to/from 9-track magnetic tapes for communications with the ICL 1900 series of computers are described in the following sub-topics.

U0200 - Read/Write Data Block

U0201 - Read/Write Start-of-Data

U1201 - Read/Write End-of-Data

U2201 - Access/Update Current Tape Block

**Note:** In multi-tape systems all tape operations refer to the default channel assignments, thus `T-ATT` implies unit 1 whereas `ASSIGN =TAPE 2` gives a default channel of unit 2. Named channel variable assignments, such as `ASSIGN X=TAPE 2`, do not work.

#### **U0200 - Read/Write Data Block**

The U0200 routine enables the user to read or write a data block. It provides a method of overcoming the technical difficulties imposed by the operating system; it is not a version of the ICL 1900 housekeeping routines. Thus, the Applications Programmer is still responsible for the formatting of data (including labels and sentinels) read from tape. It is anticipated that the user exit will be used in conjunction with user-written DataBasic programs for each application.

#### **U200 Read**

The following shows how a data block is read from tape using the DataBasic intrinsic function ICONV and the U0200 routine.

```
<block> = ICONV(0,'U0200')
```

Legend: On entry: <block> is blank  
On exit : <block> is attribute 1 -

status (see later)  
attribute 2 - block

A block is read from tape, converted to ASCII and returned as the second attribute of variable <block>.

#### U200 Read Example

```
BLOCK = ICONV(0, 'U0200'); *READ NEXT BLOCK
IF BLOCK <1> THEN; *STATUS = 0?
    STATUS <1> = BLOCK <1>; *NO!
GOSUB 1000; *DISPLAY STATUS MSG.
STOP
END
```

#### U200 Write

The following shows how a data block is written to tape using the DataBasic intrinsic function OCONV and the U0200 routine.

<status> = OCONV (<block>, 'U0200')

Legend:        On entry: <status> is blank  
                 <block> is the block to write  
                 to tape  
                 On exit : <status> is attribute 1-  
                             status (see later)  
                             attribute 2-null

A block in variable <block> is passed to the U0200 routine; it is converted from ASCII to ICL 1900 code and written to tape.

#### U200 Write Example

```
STATUS = OCONV(BLOCK, 'U0200') *WRITE NEXT BLOCK
IF STATUS <1> THEN *STATUS = 0?
    GOSUB 1000; *NO, DISPLAY STATUS MSG.
STOP
END
```

#### Restrictions on ICL 1900 Tape Handling

The following restrictions apply to all ICL 1900 9-track tape handling applications:

1. When reading from or writing to tape, the block size is restricted to the maximum size specified in the T-ATT or ASSIGN verb or U2201 routine (see later).

2. ASCII characters outside the range X'20' to X'5F' cannot be read from tape, and will be written to tape as garbage.
3. Multi-reels are not supported.
4. Word counts and multi-record blocks must be handled explicitly by the application program.

### U0201/ U1201 - Read/Write Start-of-Data or End-of-Data Sentinel

The U0201 routine enables you to read or write a start-of-data (SOD) sentinel. U1201 enables you to read or write an end-of-data (EOD) sentinel.

The following shows how SOD and EOD sentinels are read or written using the DataBasic intrinsic functions ICONV and OCONV in conjunction with U0201 and U1201.

Read SOD sentinel: <status> = ICONV (0,'U0201')

Write SOD sentinel: <status> = OCONV (0,'U0201')

Read EOD sentinel: <status> = ICONV (0,'U1201')

Write EOD sentinel: <status> = OCONV (0,'U1201')

Legend: On exit: <status> is attribute 1-  
status (see later) attribute 2-null

### Example

```

STATUS = ICONV (0,'U0201')   *READ SOD SENTINEL
IF STATUS <1> THEN           *STATUS = 0?
  GOSUB 1000                 *NO, DISPLAY
                             STATUS MSG.
  STOP
END

```

### Status

The first attribute of the string returned after user exit calls to routines U0200, U0201 and U1201, is the status. The status is a single numeric character in the range 0 to 7. Below is an example of a DataBasic subroutine which displays a status message with respect to the status character. Status = 0 implies a successful operation.

```

*
*DISPLAY STATUS MESSAGE

```

```

*
1000 ON STATUS<1> GOTO
      1010,1020,1030,1040,1050,1060,1070
      RETURN
      1010 PRINT 'UNEXPECTED TAPE-MARK';
                                *STATUS = 1
      RETURN
1020 PRINT 'TAPE UNIT NOT ATTACHED';*STATUS = 2
      RETURN
1030 PRINT 'ATTEMPT TO WRITE NULL BLOCK';
                                *STATUS = 3
      RETURN
1040 PRINT 'INVALID BLOCK LENGTH'; *STATUS = 4
      RETURN
1050 PRINT 'SOD SENTINEL FORMAT ERROR';
                                *STATUS = 5
      RETURN
1060 PRINT 'EOD SENTINEL FORMAT ERROR';
                                *STATUS = 6
      RETURN
1070 PRINT 'BLOCK COUNT ERROR';   *STATUS = 7
      RETURN

```

### Guidelines for Label and Sentinel Processing

Processing is eased considerably by using the simplest possible tape layout, that is:

<data-blocks><eof> where <eof> is a tape-mark

Other tape layouts in order of increasing complexity are as follows:

<sod-sentinel><data-blocks><eof><eod-sentinel> <eof>

<header><eof><sod-sentinel><data-blocks><eof> <eod-sentinel><eof>

### Guidelines for Programs Handling Data

The following DataBasic features and user exits are particularly useful:

#### Input

U11A4	Converting binary numbers to decimal.
U11A4/-24836	Converting ICL dates to Reality dates.
U047F	Stripping leading zeros from numeric fields.
U31A4	Stripping trailing spaces from alphanumeric fields.
□	Extracting individual fields.

---

## Output

U11A4	Converting decimal numbers to binary.
+24836/U11A4	Converting Reality dates to ICL dates.
LEN()/U11A4	Generating word counts.
U047F	Padding out numeric fields to fixed length.
.;CAT	Concatenating fields.

**Note:** The constant, 24836, used above is the difference in the base dates of ICL systems and Reality.

## U2201 - Access/Update Current Tape Block Size

The U2201 routine enables the user to access or update the current tape block-size. This is achieved using the DataBasic intrinsic functions ICONV or OCONV and the U2201 routine.

### Access

The following shows how to access the current tape block size:

```
<block-size> = ICONV(0,'U2201')
```

Legend:	On entry:	<block-size> is blank.
	On exit:	<block-size> is the current block size.

### Example

```
BLOCKSIZE = ICONV(0,'U2201'); *GET CURRENT
                                BLOCK-SIZE
```

### Update

The following shows how to update the current tape block size:

```
<dummy> = OCONV (<block-size>,'U2201')
```

Legend:	On entry:	<dummy>	is blank.
		<block-size>	is new tape block size.
	On exit:	<dummy>	is the new block size.

Example

```
BLOCKSIZE = '1024'; *ASSIGN BLOCKSIZE VALUE
DUMMY = OCONV(BLOCKSIZE,'U2201'); *UPDATE CURRENT BLOCKSIZE
```

## ICL 1900 General Utilities

### U01A4 - ASCII <-> ICL 1900 7-Track Code

This routine converts ASCII code to ICL 1900, 7 track format and vice versa. This is a DataBasic (OCONV and ICONV) facility.

Example 1

Tape output statement using U01A4:

```
WRITET OCONV(DATA,"U01A4") ELSE
```

Example 2

Tape input statement using U01A4:

```
READT UCADATA THEN
CDATA = ICONV(UCADATA,"U01A4")
```

Note that this routine only recognises the 64 character code set. In ASCII, these are space (Hex 20) to underline (Hex 5F). Any other characters will generally convert to garbage.

### U11A4 - Decimal <-> ICL 1900 Binary

This routine eases the conversion of numbers from ICL 1900 binary format to the System decimal format and vice-versa.

To those unfamiliar with ICL 1900 binary format, the following brief explanation should prove useful:-

ICL 1900 computers have 24-bit words, which can be interpreted in several ways. Two common interpretations are:

1. Characters. These are held in six bits each, therefore there are four characters in a word. The 64 unique combinations of six bits represent letters, numbers, and special characters. The character set is equivalent to the ASCII characters in the range hex '20' to hex '5F' inclusive (but in a different order).

2. Numbers. These are held in a variety of binary formats, the simplest of which is an unsigned binary word. The value of the number can lie in the range 0 (all bits zero) to 16,777,215 (all bits one).

For example, the ICL bit pattern for the character '0' (zero) is 000000. The ICL bit pattern for the character '%' is 001101. Thus, an ICL word containing the bits:

```
0000000000000000000000001101
```

could be thought of as containing either the character string '000%' or the number 21. This routine when used in the ICONV function in DataBasic, will convert four characters into the 'equivalent' decimal number. The reverse occurs in the OCONV function. For example:

```
ICONV('000%', 'U11A4') evaluates to 21
```

```
OCONV(21, 'U11A4, 4') evaluates to 000%
```

In the OCONV case, the call-up code (U11A4) is followed by any non- hexadecimal character (a comma in the above example) and the length of the required result (four for a single ICL word).

Since this routine converts numbers to/from ASCII characters, it is still necessary to perform code conversion to/from ICL character format. Thus, U11A4 would be used in an ICONV statement on the binary words within a character string that had already been converted to ASCII using U01A4 or READ-1900. It would be used in an OCONV statement to generate character strings for subsequent conversion to ICL code by U01A4 or WRITE-1900. Note that this contrasts with the conversion of packed decimal numbers in IBM format, which must not go through the ASCII/EBCDIC conversion.

#### Restrictions

This routine will only convert numbers in the range 0 to 8,388,607. It will not convert negative numbers or handle character strings greater than four characters (one ICL word). Attempts to convert numbers outside this range will lead to unpredictable results.

In order to handle other ICL binary formats (for example, signed, or double-length, or mid-point) it will be necessary to use DataBasic to manipulate the data to conform to these limits.

For example, a signed binary number in ICL format is represented in two's complement form. It is necessary to do the complementing in DataBasic in order to handle negative numbers. Single-length (one-word) negative numbers could be handled thus:-

```

NUMBER = ICONV (STRING, 'U11A4')
IF NUMBER > 8388607 THEN NUMBER = NUMBER -16777216
...
IF NUMBER < 0 THEN NUMBER = 16777216 + NUMBER
STRING = OCONV (NUMBER, 'U11A4,4')

```

Double-length numbers must be split into two words and converted individually. For example, for unsigned numbers:-

```

LEFTNUM  = ICONV (STRING[1,4], 'U11A4')
RIGHTNUM = ICONV (STRING[5,4], 'U11A4')
NUMBER   = 8,388,607*LEFTNUM + RIGHTNUM
...
LEFTNUM  = INT (NUMBER/16777216)
RIGHTNUM = NUMBER - 16777216*LEFTNUM
STRING   = OCONV (LEFTNUM, 'U11A4,4') :
OCONV (RIGHTNUM, 'U11A4,4')

```

Double-length signed numbers are more complex: the suggested coding is as follows:-

```

LEFTNUM  = ICONV (STRING[1,4], 'U11A4')
RIGHTNUM = ICONV (STRING[5,4], 'U11A4')
IF LEFTNUM > 8388607 THEN
LEFTNUM = LEFTNUM - 16777215
IF RIGHTNUM THEN RIGHTNUM = RIGHTNUM - 8388608
END
NUMBER = 8388608*LEFTNUM + RIGHTNUM
...
LEFTNUM  = INT (NUMBER/8388608)
RIGHTNUM = NUMBER - 8388608*LEFTNUM
IF NUMBER < 0 THEN
LEFTNUM = 16777215 + LEFTNUM
IF RIGHTNUM THEN RIGHTNUM = 8388608 + RIGHTNUM ELSE
LEFTNUM = LEFTNUM+1
END
STRING = OCONV (LEFTNUM, 'U11A4,4') : OCONV (RIGHTNUM, 'U11A4,4')

```

## ICL READ/WRITE - 2900 Verbs

Two TCL-II verbs, READ-2900 and WRITE-2900, enable you to read or write 9-track magnetic tapes for communications with the ICL 2900 series of computers.

These verbs provide a method of overcoming the technical difficulties imposed by the operating system; they are not versions of the ICL 2900 housekeeping utilities. Thus, the Applications Programmer is still responsible for the formatting of data read from tape. It is anticipated that the verbs will be used in conjunction with user-written Procs and DataBasic programs for each application.

The default action of the verbs is to write/read fixed length records (80 characters long). Data is converted from ASCII to EBCDIC when writing and vice-versa when reading. Various options, listed later, are provided to modify the default action of these verbs.

**Note:** In multi-tape systems all tape operations refer to the default channel assignments, thus `T-ATT` implies unit 1 whereas `ASSIGN =TAPE 2` gives a default channel of unit 2. Named channel variable assignments, such as `ASSIGN X=TAPE 2`, do not work.

### Single File Structure

The data structures used conform to ICL 2900 standards. Each \* in the following represents a tape mark.

VOL1	HDR1	HDR2	*	datafile	*	EOF1	EOF2	*	*
------	------	------	---	----------	---	------	------	---	---

**Figure 2-1. Single File Structure**

A detailed description of the single file data structures is provided below.

<u>Columns</u>	<u>Field Description</u>	<u>Length</u>	<u>Contents</u>
<b>VOL1 header label.</b>			
1-4	Label identifier	4	VOL1
5-10	Volume identifier	6	AAAAAA
11-79	Non-mandatory fields	69	spaces
80	Label standard version	1	2
<b>HDR1 header label.</b>			
1-4	Label identifier	4	HDR1
5-21	File identifier	17	CMCCHARGES^^^^^^

<u>Columns</u>	<u>Field Description</u>	<u>Length</u>	<u>Contents</u>
			^
22-27	File set identifier	6	DATA00
28-31	File section number	4	0001
32-35	File sequence number	4	0001
36-39	Generation number	4	0001
40-41	Version number	2	01
42-47	Creation date	6	^00001
48-53	Expiration date	6	^00001
54	Accessibility	1	space
55-60	Block count	6	000000
61-73	System code	13	Spaces
74-80	Reserved	7	Spaces
<b>HDR2 header label.</b>			
1-4	Label identifier	4	HDR2
5	Record format	1	V, F or U
6-10	Block length	5	nnnnn
11-15	Record length	5	nnnnn
16-50	Reserved	35	Spaces
51-52	Buffer offset length	2	08
53-80	Reserved	28	Spaces
<b>EOF1 trailer label.</b>			
1-4	Label identifier	4	EOF1
5-54	As HDR1	50	

<u>Columns</u>	<u>Field Description</u>	<u>Length</u>	<u>Contents</u>
55-60	Block Count	6	nnnnn
61-80	As HDR1	20	
<b>EOF2 trailer label.</b>			
1-4	Label identifier	4	EOF2
5-80	As HDR22	76	

Where a '^' and 'n' represent a space and a digit.

### WRITE-2900 Verb

WRITE-2900 is a TCL-II verb. It allows you to specify the file (containing the data) to be written to tape. The verb reads data from the file, converts it from ASCII to EBCDIC and writes it to tape. Each attribute (of each item) generates one record on tape. Item-ids are not written to tape.

**Note:** In the examples that follow, system messages are omitted for clarity.

#### Example 1

Write fixed length records, default record size = 80.

```
:ASSIGN =TAPE 2
:T-REW
:SELECT OUTPUTFILE
>WRITE-2900 OUTPUTFILE
:T-REW
:T-DET
```

#### Example 2

Write fixed length records, block size = 2048, record size = 140 and header/trailer labels.

```
:T-ATT SIZE=2048
:T-REW
:SELECT OUTPUTFILE
>WRITE-2900 OUTPUTFILE (L,140,T)
:T-REW
:T-DET
```

**Example 3**

Write variable length records, block size = 512, record size = 504 and header/trailer labels.

```
:ASSIGN =TAPE 3 SIZE=512
:T-REW
:SELECT OUTPUTFILE
>WRITE-2900 OUTPUTFILE (L,V,T)
:T-REW
:T-DET
```

The options illustrated above are fully explained in a later section.

**READ-2900 Verb**

READ-2900 is also a TCL-II verb, but requires no item-list. It reads blocks from tape, converts data from EBCDIC to ASCII and writes it into items of a Reality file. The verb allocates numeric item-ids starting at 1, and increasing by one for every block read. Items may contain more than one attribute.

Segment-marks (X'FF') in records are converted to asterisks (X'2A'). Attribute marks (X'FE') in records will generate extra attributes in the item (Number of attribute-marks plus one).

**Example 1**

Read fixed length records, default record size = 80.

```
:ASSIGN =TAPE 4
:T-REW
:CLEAR-FILE (DATA INPUTFILE
:READ-2900 INPUTFILE
:T-REW
:T-DET
```

**Example 2**

Read fixed length records, block size = 2048, record size = 200 and header/trailer labels.

```
:T-ATT SIZE=2048
:T-REW
:CLEAR-FILE (DATA INPUTFILE
:READ-2900 INPUTFILE (L,200,T)
:T-REW
:T-DET
```

**Example 3**

Read variable length records, block size = 1024, record size = 1016 and header/trailer labels.

```
:ASSIGN =TAPE 2 SIZE=1024
:T-REW
:CLEAR-FILE (DATA INPUTFILE
:READ-2900 INPUTFILE (L,V,T)
:T-REW
:T-DET
```

**Example 4**

Write fixed length records, default record size = 80, 15 records per block, with unique header and trailer labels. Block size used =  $80 \times 15 + 8 = 1208$ .

```
:ASSIGN =TAPE 3 SIZE=1208
:T-REW
:SELECT OUTPUTFILE
>WRITE-2900 OUTPUTFILE (L,T,P)
VOLUME IDENTIFIER > VOL-ID
FILE IDENTIFIER > FILE-ID
:T-REW
:CLEAR-ASSIGN
```

The VOL1 header label will have the VOL-ID, space filled to six characters, inserted in columns 5-10.

The HDR1 header label will have the FILE-ID, space filled to 17 characters, inserted in columns 5-21.

The EOF1 trailer label will have the file identifier, space filled to seventeen characters, inserted in columns 5-21.

**Options**

The options illustrated above are fully explained in the following section:

**Note:** The default action is to write/read fixed length records (Record format "F" and records 80 characters long) with no header or trailer labels.

**C** Inhibit Conversion. Data conversion from ASCII to EBCDIC (write) or EBCDIC to ASCII (read) is inhibited.

- L** Header Labels. Three header labels and a tape-mark are written/read. These are:
- VOL1
  - HDR1
  - HDR2
  - tape-mark
- P** Unique Labels. Write unique labels (only valid in conjunction with option L). You are prompted for the volume and file identifiers required. Enter a volume identifier of up to six characters and a file identifier of up to 17 characters.
- The volume-id is written into columns 5 to 10 of the VOL1 header label (space filled if less than six characters).
- The file-id is written into columns 5-21 of both the HDR1 header label and the EOF1 trailer label (space filled if less than 17 characters).
- T** Trailer Labels. Two trailer labels and two tape-marks are written/read. These are:
- EOF1
  - EOF2
  - tape-mark
  - tape-mark
- V** Variable Length Records. Variable length records are written/read. The maximum record size is T-ATT block size minus 8. The record format character is changed to "V".
- U** Unblock option. Unblocks multiple logical records from tape into separate Reality items.
- n* Decimal Number Option. The decimal number is used as the maximum record size. This option may be used in conjunction with other options.

## Error Messages

In all, there are five labels and each label is validated. Any label with an invalid format (on reading) causes one of the following error messages to be output to the terminal.

```
*1200 LABEL 'VOL1' FORMAT ERROR.  
*1201 LABEL 'HDR1' FORMAT ERROR.  
*1202 LABEL 'HDR2' FORMAT ERROR.  
*1203 LABEL 'EOF1' FORMAT ERROR.  
*1204 LABEL 'EOF2' FORMAT ERROR.
```

When reading, if the block size (in the block header) does not match the size of the block read, the following error message is displayed:

```
*1205 BLOCK LENGTH ERROR.
```

There is one tape-mark after the header labels and two tape-marks following the trailer labels. If any one of these tape-marks are missing (when reading), the following error message is displayed:

```
*1206 TAPE MARK(S) MISSING AFTER HEADER/TRAILER LABELS.
```

When writing to tape, the length of the record (attribute) is matched against the record size specified (option or default). If there is a mismatch, the following error message is displayed:

```
*1207 RECORD LENGTH ERROR.
```

As each data block is read, a block count is incremented by one. At the end of the data, a block count is extracted from the first trailer label. If the block counts do not match the following error message is displayed:

```
*1208 BLOCK COUNT ERROR
```

Each block written to tape has a block header. The first 4 bytes of the block header are the Block Sequence Indicator (BSI). The first BSI is zero. Therefore as each block is read, the BSI is checked and if there is any discrepancy, the following error message is displayed:

```
*1209 BLOCK SEQUENCE INDICATOR ERROR
```

## Bull Utilities

### U219A - ASCII <-> Bull 7-Track Code

This routine converts ASCII code to Bull 7-track tape code (OCONV) or vice versa (ICONV).

Thus a tape output routine might look like this:

```
WRITET OCONV(DATA, 'U219A') ELSE
```

and a tape input routine might look like this:

```
READT DATA THEN  
DATA = ICONV(DATA, 'U219A')
```

Note that this utility only recognises a 64 character set; in ASCII these are space (hex '20') to underline (hex '5F'). Any other characters will generally convert to garbage.

### U319A - ASCII <-> Bull 7-Track Code

This routine is an extension of U219A, in that:

- |           |  |
|-----------|--|
| On input  | Trailing spaces are removed from the data block.                                   |
| On output | The block is adjusted to 80 bytes in length by truncation/space-fill at the right. |

---

## Chapter 3

# Terminal Printing Techniques

This chapter describes a number of printing techniques available for output to printers attached to terminals. The techniques described are: Print Screen, Spooled Printing, Direct Printing, Hardcopy Printing, Straight-through and Autoprint.

## Introduction

A number of different printing techniques are available for use with printers attached to terminals. The techniques available to you depend on the model of terminal you are using.

Print Screen and Spooled Printing are available with all types of terminal. Model 6748 requires the SP terminal executive to be loaded to support Spooled Printing.

Direct and Hardcopy printing are available with all terminals except Model 6748.

Straight-through is available on all terminals, except models 9745, 12080/9 and 12P3xxx, once the SP terminal executive has been loaded.

### **Notes:**

1. Model number 12P3xxx specifies a number of different models of the P30 terminal. The last three characters (xxx) comprising alphanumerics identify the different models.
2. When using Model 11310 or 11320 printer the SP terminal executive must be loaded in the terminal for all printing in order to effect character conversions between the standard ASCII and the Northgate character sets. It is also necessary to set the printer type in the terminal's local set-up menu to 12 or 13.
3. 12P3xxx models do not support a terminal executive.

## Terminal Executive

The SP terminal executive is downloaded from the host by executing the TERMLoad-SP command at the terminal into which it is to be loaded. Once loaded, the terminal model number is displayed to the right of the twenty-fifth line of the screen. While actually printing, the model number is replaced by a flashing 'P'. It is recommended that loading SP is executed in an initial set-up Proc.

## Printing Facilities

The following paragraphs provide a brief description of the printing facilities available via terminals. [Tables 3-1](#) on page 4 and [3-2](#) on page 5 summarise the facilities available with particular terminals and how to implement them.

### Print Screen

The Print Screen facility copies whatever data is currently displayed on the screen to the attached printer. It is a single process and does not continue as further data is displayed. The function may be controlled by either keyboard or program. Print screen may be used on all configurations.

A typical use of this facility is to obtain hard copies of parameter settings.

### Direct Printing

Direct Printing allows the printer to be used as the terminal's output device, in place of the screen. While the function is ON, both screen and most of the keyboard are disabled. It is available under program control only.

Direct Printing should be used for output generated by an application program.

### Straight-through Printing

Straight-through Printing allows the printer to be used as the terminal's output device in place of the screen. While ON, the screen and most of the keyboard are disabled.

Straight-through Printing can be used for the same applications as Direct Printing, but also supports configurations without built-in flow control. The SP terminal executive must be loaded in the terminal. A programmed 'protocol' must be included in the application, to control the flow of print data. Full details of this are given in [Straight-through Printing](#).

### Hardcopy Printing

Hardcopy Printing allows the production of a hardcopy 'log' of all screen activity. The screen and keyboard continue to operate normally. Hardcopy is switched on/off through the keyboard. Hardcopy is useful for recording and for learning purposes.

### Autoprint

The Autoprint facility is a simplified version of Straight-through Printing. It dispenses with the need for program controlled data flow when an automatically 'paged' output process (English or RUNOFF) is used to generate the data.

Autoprint requires the SP terminal executive to be loaded in the terminal. It can be controlled by either keyboard or program, and while ON the screen and most of the keyboard are disabled.

### Spooled Printing

Spooled Printing enables the contents of a spooler queue to be printed on the terminal printer.

**Table 3-1. Keyboard Controlled Printing Facilities**

\ Model	12P3xxx	12080/90 and 12140/42	9747	9745	6748
Facility \	(P30)	(P8/P9 & P12/14)	(P5)	(P4)	(P99)
Print Screen	PRINT SCREEN	CTRL+F16 or F16 *	CTRL+F1 or F1 *	CTRL+F1 or F1 *	F1 *
Hardcopy ON	Configured in Setup Menus	CTRL+F15 then F16	CTRL+F2	CTRL+F2	Not Available
OFF		CTRL+F15 then F16	CTRL+F3	CTRL+F3	
Straight- -through ON	Not Available	Not Available	F2*	Not Available	F2*
OFF		-	F3*		F3*
Autoprint ON	Not Available	CTRL+F13	F4*	Not Available	F4*
OFF		CTRL+F14	F5*		F5*
Spooled	PORT- DESPOOL (SX)	PORT-DESPOOL (SX)	PORT- DESPOOL (SX)	Not Availabl e	PORT- DESPOOL (SX) *

\* Asterisk indicates functionality when SP Terminal executive loaded

**Table 3-2. Program Controlled Printing**

\ Model Facility \	12P3xxx (P30)	12080/90 and 12140/42 (P8/P9 & P12/14)	9747 (P5)	9745 (P4)	6748 (P99)
Print Screen	ESC U	ESC U	Hex 13	Hex 13	Hex 11
Direct Print ON	ESC R ESC T	ESC R ESC T	ESC R ESC T	ESC R ESC T	Not Available
OFF					
Straight- ON -through	Not Available	Hex 16 *	Hex 16 *	Not Available	Hex 16 *
OFF		Hex 17 *	Hex 17 *		Hex 17 *
Autoprint ON	Not Available	Hex 1C *	Hex 1C *	Not Available	Hex 1C *
OFF		Hex 1D *	Hex 1D *		Hex 1D *

\* Asterisk indicates functionality when SP Terminal executive loaded

## Print Screen

Print Screen copies the whole of the currently displayed page to the printer. All lines, including blank lines, are printed.

Print Screen is effected from keyboard or program as follows:

**Table 3-3. Effecting Print Screen**

	<b>12P3xxx (P30)</b>	<b>12080/90 and 12140/42 (P8/P9 &amp; P12/14)</b>	<b>9747 (P5)</b>	<b>9745 (P4)</b>
Print Screen	PRINT SCREEN	CTRL+F16 or F16 *	CTRL+F1 or F1 *	CTRL+F1 or F1 *
Program	ESC U	ESC U	Hex 13	Hex 13

\* SP Terminal executive loaded functionality

The following examples show Print Screen operation from a DataBasic program and Proc written to drive a P8 terminal.

```

PS-TEST1
001 * Clear screen
002 EQUATE ESC TO CHAR(27),FF TO CHAR(12)
003 PRINT FF
004 * Start print loop
005 FOR X = 1 TO 23
006 PRINT "123":SPACE(74):"890":
007 NEXT X
008 * Issue Print Screen code
009 PRINT ESC:"U":
010 END
    
```

```

PS-TEST1
001 PQN
002 C Clear screen
003 T C,+
004 C Output data to screen
005 T "PRINT SCREEN DEMONSTRATION LINE"
006 P
    
```

007 C Issue Print Screen code  
008 T X1B,X55

**Note:** When the Print Screen code is issued, the printer starts printing from the current head position. An initialising routine to ensure that the print head is positioned before the Print Screen code is issued is therefore required.

## Direct Printing

Direct Printing outputs to the printer instead of the terminal's screen. While the function is ON both screen and keyboard are disabled except for the BREAK key. Direct printing is available under program control only.

Direct printing is turned ON and OFF as follows:

**Table 3-4. Codes to Control Direct Print**

	<b>12P3xxx (P30)</b>	<b>12080/90 and 12140/42 (P8/P9 &amp; P12/14)</b>	<b>9747 (P5)</b>	<b>9745 (P4)</b>
<b>ON</b>	ESC R	ESC R	ESC R	ESC R
<b>OFF</b>	ESC T	ESC T	ESC T	ESC T

The following is an example DataBasic program which controls Direct Printing.

```

DIR-TEST
001      *****
002      * This program demonstrates Direct Printing.
003      * It prints the Item Names and the data in the first
004      * attribute of all the items in the Master Dictionary
005      *****
006      * INITIALISE PROGRAM
007      * Set variables to null or 0 as
           appropriate
008      * Set up constants
009      *****
010      FILE='';ID='';ATTR1='';PDATA='';
           LINECOUNT=0
011      EQUATE DIRON TO CHAR(82),DIROFF TO
           CHAR(84),ESC
           TO CHAR(27),FF TO CHAR(12),CR TO
           CHAR(13)
012      *****
013      * OPEN FILE AND SELECT ITEMS
014      *****
015      OPEN '', 'MD' TO FILE ELSE PRINT "Cant open MD"; STOP
016      SELECT FILE
017      *****
018      * TURN DIRECT PRINT ON

```

```
019          *****
020          PRINT ESC:DIRON
021          PRINT FF:CR
022          *****
023          * GET AND PRINT DATA
024          *****
025 100      READNEXT ID ELSE GOTO 200
026          READV ATTR1 FROM FILE, ID, 1 ELSE ATTR1="Nothing
          there"
027          * Pad out ID with spaces to 40 characters -cosmetic
028          ID=ID:SPACE(40-LEN(ID))
029          * Construct print line
030          PDATA=ID:ATTR1
031          * Print line and check for end of page
032          PRINT PDATA:CR
033          LINECOUNT=LINECOUNT+1
034          IF LINECOUNT>64 THEN LINECOUNT=0;PRINT FF
035          GOTO 100
036          *****
037          * TURN DIRECT PRINT OFF AND FINISH
038          *****
039 200      PRINT FF:ESC:DIROFF
040          END
```

## Straight-through Printing

Straight-through Printing allows the printer to be used as the terminal's output device in place of the terminal screen. Whilst the Straight-through function is ON, the terminal effectively becomes a printing device - the terminal screen and most of the keyboard being disabled.

The only keys that remain operative are CTRL and BREAK, to allow abortion of the process.

Operation of the Straight-through Printing facility is very simple providing that either:

- the total print output is no longer than 1792 characters - including CR and LF characters but not including null (Hex 00) characters,
- or
- the line speed between the terminal and CPU is the same as or less than the print speed.

Under these circumstances, Straight-through Printing is accomplished by simply turning the facility ON from a program or Proc, performing the task and then turning the facility OFF again.

To perform a Straight-through Printing operation when neither of these two conditions can be met, it is necessary to impose some sort of line protocol on the line between the CPU and the terminal so that its speed is effectively reduced to that of the printer. There are several methods of accomplishing this but all are based on the principal of making a program running in the CPU send no more than 1500 characters at a time and then waiting for some sort of prompt before outputting the next 1500 characters.

The terminal buffer has room for 1792 characters but the 'buffer space available' code will be issued by the terminal when there are still 200 characters left in the buffer. 1500 characters is therefore the recommended maximum to be sent to the terminal in any one transmission. All of the programming examples given later deal with the implementation of Straight-through Printing using just such a protocol.

Straight-through Printing can only be used when the SP Terminal executive is loaded in the terminal. It is turned ON and OFF on the various terminals as follows:

**Table 3-5. Controlling Straight-through Printing**

	<b>12080/90 and 12140/42 (P8/P9 &amp; P12/14)</b>	<b>9747 (P5)</b>	<b>6748 (P99)</b>
Keyboard ON	F2	F2	
OFF	-	F3	F3
Program ON	Hex 16	Hex 16	Hex 16
OFF	Hex 17	Hex 17	Hex 17

To turn the function OFF from a program, the terminal must be prompted with a Decimal 23 (Hex 17) code and then the program made to loop until a Decimal 21 (Hex 15) response is received.

**Note:** Some printers are unable to interpret X,Y positional addressing (PRINT @(x,y) statements in DataBasic or T (x,y) statements in Proc), and some printers do not accept 'form feed' codes, in which case the vertical position of the printer must be controlled by using a line count and issuing 'line feed' codes only.

### Line Protocol Codes

The codes used in programming the line protocol for Straight-through Printing are as follows:

- Dec 22/Hex 16 Sent from program to terminal, turns Straight-through facility ON. Should always be followed by Decimal 05.
- Dec 05/Hex 05 Sent from program to terminal, enquires whether there is sufficient space in the buffer to receive a block of data (up to 1500 bytes). When the terminal's print buffer does have the required capacity, the terminal will automatically respond by sending Decimal 21. This interrogation routine should always be used, even before sending the first block of data. If the CPU receives any code other than Decimal 21 it should simply repeat the Decimal 05 enquiry code.
- Dec 21/Hex 15 Automatically generated by the terminal in response to a Decimal 05 prompt from the CPU when there is enough room in the 1792 byte

print buffer for the next 1500 bytes (i.e when the buffer contains less than 200 bytes). Also sent at the end of Straight-through Printing - see following.

Dec 23/Hex 17 Sent from program to terminal, turns Straight-through facility OFF at the terminal. This code does not have an immediate effect but is simply placed in the terminal's print buffer after the last print character. When the code is processed from the buffer, the terminal turns Straight-through Printing OFF and sends Decimal 21 to the CPU. This informs the CPU that the terminal has completed a Straight-through Printing task and has returned to normal operation.

**Note:** The controlling DataBasic program or Proc runs in the CPU and communicates with the Terminal Executive which resides in the terminal.

## Sample Programs

The following sample programs show the correct methods of applying the line protocol coding which is used to avoid buffer overflow problems.

Example DataBasic program employing user-coded protocol to control Straight-through Printing.

```
                STP-TEST
001  *****
002  * This program will demonstrate Straight-through
      Printing.
003  * It will print the Item Names and the data in the
      first
004  * attribute of all the items in the Master Dictionary.
005  *****
006  * INITIALISE PROGRAM.
007  * Set variables to null or 0 as appropriate. Set up
008  * constants and set PROMPT to Decimal 05 (protocol
009  * enquiry char) 009*
010  FILE='';ID='';ATTR1='';PDATA=''; DATACOUNT=0;
      LINECOUNT=0;
011  EQUATE ENQ TO CHAR(05), STPON TO CHAR(22), STPOFF TO
      CHAR(23), FF TO CHAR(12), CR TO CHAR(13)
012  PROMPT ENQ
013  *****
014  * OPEN FILE AND SELECT ITEMS.
015  *
016  OPEN '', 'MD' TO FILE ELSE PRINT "Cant open MD";STOP
017  SELECT FILE
```

```
018  *****
019  * CALL ROUTINE TO TURN STRAIGHT-THROUGH PRINTING ON.
020  *
021  GOSUB 700
022  *****
023  * GET-DATA LOOP.
024  * If there is no more data, go to finish-up routine.
025  *
026 100 READNEXT ID ELSE GOTO 200
027  READV ATTR1 FROM FILE, ID 1 ELSE ATTR1="Nothing there"
028  * Pad out ID with spaces to 40 characters -cosmetic.
029  ID=ID:SPACE(40-LEN(ID))
030  * Put ID and ATTR1 in PDATA.
031  PDATA=ID:ATTR1
032  * Call print routine.
033  GOSUB 800
034  *****
035  * TURN STRAIGHT-THROUGH OFF AND FINISH
036  * Set PROMPT character to Decimal 23 and wait until
037  * 'buffer empty' response is received from terminal.
038  *
039 200 PROMPT STPOFF
040  LOOP INPUT CODE UNTIL SEQ(CODE) = 21 DO REPEAT
041  STOP
042  *****
043  * SUBROUTINES
044  *****
045DR* INITIALISE PRINTER AND TURN STRAIGHT-THROUGH ON
      ROUTINE.
046  * Turn Straight-through Printing ON and check for
047  * buffer empty before sending 'form feed' and
      'carriage
048  * return. to put the print head at first position on
      new page.
049  *
050 700 PRINT STPON
051  LOOP INPUT CODE UNTIL SEQ(CODE) = 21 DO REPEAT
052  PRINT FF:CR:
053  RETURN
054  *****
055  * PRINT ROUTINE
056  *
057  * Increase DATACOUNT by number of data characters to
      be output
058  * plus 2 for the 'carriage return' and 'line feed'
      characters.
059 800 DATACOUNT=DATACOUNT+LEN(PDATA)+2
060  * Check to see if outputting current data will cause
```

```
061 * buffer overflow.If it will then use protocol loop,
062 * wait for 'buffer empty' from terminal before
      printing
063 * and reset DATACOUNT to the number of characters in
      the current output only.
064 IF DATACOUNT > 1500 THEN
065 LOOP INPUT CODE UNTIL SEQ(CODE) = 21 DO REPEAT
066 DATACOUNT=LEN(PDATA)+2
067 END
068 * Print the data.
069 PRINT PDATA
070 * Check the number of lines output so far and issue
071 * 'form feed' if necessary.
072 LINECOUNT-LINECOUNT+1
073 IF LINECOUNT > 64 THEN LINECOUNT=0;PRINT FF:
074 * Return to Get Data loop.
075 RETURN TO 100
076 *****
077 END
```

The following program is exactly the same as the [previous example](#), but with comments removed so as to show the coding more clearly.

```
STP-TEST
001 FILE='';ID='';ATTR1='';PDATA='';
      DATACOUNT=0;LINECOUNT=0
002 EQUATE ENQ TO CHAR(05), STPON TO CHAR(22), STPOFF TO
      CHAR(23), FF TO CHAR(12), CR TO CHAR(13)
003 PROMPT ENQ
004 OPEN '','MD' TO FILE ELSE PRINT "Cant open MD";STOP
005 SELECT FILE
006 GOSUB 700
007 100 READNEXT ID ELSE GOTO 200
008 READV ATTR1 FROM FILE,ID,1 ELSE ATTR1="Nothing
      there"
009 ID=ID:SPACE(40-LEN(ID))
010 PDATA=ID:ATTR1
011 GOSUB 800
012 200 PROMPT STPOFF
013 LOOP INPUT CODE UNTIL SEQ(CODE) = 21 DO REPEAT
014 STOP
015 700 PRINT STPON
016 LOOP INPUT CODE UNTIL SEQ(CODE) = 21 DO REPEAT
017 PRINT FF:CR:
018 RETURN
019 800 DATACOUNT=DATACOUNT+LEN(PDATA)+2
020 IF DATACOUNT > 1500 THEN
021 LOOP INPUT CODE UNTIL SEQ(CODE) = 21 DO REPEAT
```

```
022          DATACOUNT=LEN(PDATA)+2
023          END
024          PRINT PDATA
025          LINECOUN T=LINECOUNT+1
026          IF LINECOUNT > 64 THEN LINECOUNT=0;PRINT FF:
027          RETURN TO 100
028          END
```

The following is an example of a Proc using user-coded protocol to control Straight-through Printing.

```
          STP-TESTPROC
001          PQN
002          C***** INITIALISE PROGRAM
003          RI
004          RO
005          MV %7 "0"
006          MV %9 "0"
007          C***** SET TERMINAL CHARACTERISTICS
008          MV #1 "TERM 132,65,0,0,2,08"
009          P
010          C***** SELECT ITEM IDs AND OPEN FILE (MD)
011          HSSELECT MD NE ":DEL"
012          STON
013          HPQ-SELECT 1
014          PH
015          F-CLEAR 1
016          F-OPEN 1 MD
017          X Cant open MD
018          C***** CALL ROUTINE TO TURN STRAIGHT-THROUGH PRINTING
          ON
019          C AND INITIALISE PRINTER
020          GOSUB 700
021          100 C***** READ NEXT ITEM
022          MV %1 !1
023          IF # %1 GO 200
024          F-READ 1 %1
025          X Cant read item named in select list !
026          C***** GET ITEM ID AND CALCULATE LENGTH
027          MV %2 "1"
028          M
029          MV %3 &1.0
030          RO
031          A(,%2)
032          IF #1 = %3 GO F
033          F;%2;C1;+;%2
034          GO B
035          M
```

```
036 C***** GET ATTRIBUTE 1 AND CALCULATE LENGTH
037 MV %5 "1"
038 M
039 MV %6 &1.1
040 RO
041 A(,%5)
042 IF #1 = %6 GO F
043 F;%5;C1;+;?%5
044 GO B
045DRM
046 C***** CALCULATE NUMBER OF SPACES NEEDED TO INCREASE
047 C LENGTH OF ITEM ID TO 40 CHARS - Cosmetic
048 F;C40;%2;-;?%4
049 C***** CALL PRINT ROUTINE
050 GOSUB 800
051 GOTO 100
052 200 C***** TURN STRAIGHT-THROUGH OFF
053 M
054 C The prompt character (dot) on the next line is
generated
055 C in EDITOR by depressing the CTRL and W keys (Decimal
23).
056 IP.%10
057 C The dot on the next line is generated in EDITOR by
058 C depressing the CTRL and U keys (Decimal 21).
059 IF %10 . GO B
060 C***** RESET TERMINAL CHARACTERISTICS AND FINISH
061 MV #1 "TERM 79,23,0,0,0,21"
062 P
063 T C,+
064 X Finished
065 C*****          SUBROUTINES *****
066 C*****          *****
067 700 C***** TURN STRAIGHT-THROUGH ON AND
068 C          INITIALISE PRINTER
069 T I22,+
070 M
071 C The prompt character (dot) on the next line is
generated
072 C in EDITOR by depressing the CTRL and E keys (Decimal
05).
073 IP.%10
074 C The dot on the next line is generated in EDITOR by
075 C depressing the CTRL and U key (Decimal 21).
076 IF %10 # . GO B
077 T I12,I13,+
078 RSUB
079 800 C***** PRINT ROUTINE
```

```

080 C Increment total character count and check if outputting
    the
081 C current line will exceed the 1500 byte maximum.  If it
    does,
082 C call protocol loop routine and wait for 'buffer empty'.
083 F;%5;C42;+;?%5
084 F;%7;%5;+;?%7
085 IFN %7 > 1500 GOTO 850
086 810 C Output data to terminal's print buffer.
087 T &1.0,S%4,&1.1
088 C Increment line count and check if 65 lines have
089 C been output for the current page.      If so, go to the
090 C 'form feed' routine.
091 F;%9;C1;+;?%9
092 IFN %9 < 65 RSUB
093 C Form Feed Routine
094 T C,+
095 MV %9 "0"
096 RSUB
097 850 C Protocol Loop Routine - waits for 'buffer empty'.
098 M
099 CThe prompt character (dot) on the next line is generated
100 C in EDITOR by depressing the CTRL and E keys (Decimal
    05)
101 IP.%10
102 C The dot on the next line is generated in EDITOR by
103 C depressing the CTRL and U keys (Decimal 21)
104 IF %10 # . GO B
105 C Reset total character count to value of current data
    only.
106 MV %7 %5
107 GOTO 810

```

The following is an example of a Proc using User Exits to control Straight-through Printing of English output:

```

      STP-END-PROC
001 C***** SET TERMINAL CHARACTERISTICS
002 MV #1 "TERM 132,65,0,0,2,08"
003 P
004 C***** INVOKE ENGLISH COMMAND AND ASSEMBLER UTILITIES
005 MV #1 "SORT STOCK BY DESCRIPTION PART DESCRIPTION
      QUANTITY
      ID-SUPP (N) "
006 U119B
007 C3
008 P

```

```
009 C***** CALL U319B TO TURN STRAIGHT-THROUGH OFF
010 U319B
011 C***** RESET TERMINAL CHARACTERISTICS
012 MV #1 "TERM 79,23,0,0,0,21"
013 P
```

User Exits U119B, U219B and U319B are simply a pre-coded version of the protocol which has already been shown in the examples. It is invoked from a Proc by including the coding shown in the above example (on lines 006, 007 and 010). As well as the coding in the Proc, a call to the utility must also be made from at least one of the attribute definition items in the dictionary of the data file. Which of the definition items must contain the call depends on whether any of the output data is obtained from multivalued attributes.

The calling code used in the dictionary of the data file is U219B. This code should be placed in attribute 007 of the relevant attribute definition items - an explanation follows. If attribute 007 already contains a conversion code, just add a Value mark (CTRL+] - Decimal 29, Hex 1D) and then U219B to the end of the string.

If none of the data being output is obtained from a multivalued attribute then the calling code (U219B) is simply included in any one of the attribute definition items which is employed during the processing of the English command.

If any of the data being output is obtained from a multivalued attribute then the calling code (U219B) must be included in all of the affected attribute definition items. The U119B code (in the Proc) turns the Straight-through facility On. The English command is then processed and the U219B code (in the dictionary) makes the processor look at the line following the U119B code in the Proc. This line must contain a C (denoting a comment line - ignored by the Proc itself) and a number between 1 and 999. This number is used by the protocol loop to determine how many print lines it can pass to the terminal before the buffer will become full. To decide the correct value for this number:

- establish the length of an average single line of output (say 132).
- multiply by the largest expected number of multivalues in any one of the data attributes being printed (say 3)(3 x 132 = 396).
- divide the result into 1500 (1500/396 = 3.79).
- round the answer down to the next whole number (must be between 1 and 999)(3.79 gives 3).

During processing, the utility halts the output of the data as soon as it has sent an estimated buffer-full. When it receives the appropriate code from the terminal to indicate that the buffer has been emptied, it restarts the output. At the end of processing, the U319B code in the Proc turns the Straight-through Printing facility OFF and waits until it gets an appropriate acknowledgement from the terminal.

**Note:** The (N) option in the English command is used to avoid having to press RETURN at the end of each page.

## Hardcopy Printing

Hardcopy printing is controlled by keyboard only. The method of control on each terminal is as follows:

**Table 3-6. Controlling Hardcopy Printing**

	<b>12P3xxx (P30)</b>	<b>12080/90 and 12140/42 (P8/P9 &amp; P12/14)</b>	<b>9747 (P5)</b>	<b>9745/6 (P4)</b>
ON	Use Setup Menu	CTRL+F15 or F16	CTRL+F2	CTRL+F2
OFF	Use Setup Menu	CTRL+F15 then F16	CTRL+F3	CTRL+F3

## Autoprint

Autoprint is a simplified version of Straight-through Printing primarily intended to remove the necessity for specially encoded line protocol when outputting automatically 'paged' data to the Terminal Printer. 'Paged' data is automatically generated by both the English and RUNOFF processors (providing the N option is not used) and it is in printing output from these processors that Autoprint will be found most useful.

A 'page' of data is defined as being any string of data that is output from the CPU in one transmission containing no more than 8192 characters, including spaces and control characters (Carriage Return, Line Feed etc.) but not including pad characters (nulls). Null codes (Hex 00) sent by the CPU will be ignored and will not be read into the buffer.

During normal terminal operation using the terminal screen and the English or RUNOFF processor, a complete 'page' of data (the size being determined by the Terminal Characteristics) is sent to the terminal in one transmission. The process then waits for a Carriage Return code to be input before continuing. When ready the operator presses the RETURN key to restart the output and receive the next page. The Autoprint facility having a large buffer holding up to 8192 (8K) characters, takes full advantage of this sequence of events.

During Autoprint operation, the whole of the first output page is placed in the 8K print buffer and only when the printer has accepted all the data will the Terminal executive prompt the CPU to send the next page. Thus, the line protocol is automatically imposed by the output processors themselves and there is no need for any additional coding.

Note that although 'paged' data could also be generated via special coding in a user-written program or Proc, this would tend to defeat the object of the exercise as the main idea behind Autoprint is to do away with the necessity of including the special line protocol coding in the first place. The only difference under these conditions, between Autoprint and Straight-through Printing would be that a larger volume of data could be transmitted in one go. This is not usually a more efficient method of handling the data because:

1. It takes longer to transmit 8K of data than to transmit 1500 bytes
2. While a new page of data is being transmitted to the terminal, the printer will only continue printing whilst there is data left in its own internal buffer.

Therefore, unless the terminal to CPU interface is set at a high enough speed or the Terminal Printer is printing at a slow enough speed, it will normally be more efficient to

use Straight-through Printing with its small block size and quick turn-round, rather than Autoprint with its large block size and longer turn-round. For this reason, the implementation of Autoprint via specially written programs will not be pursued any further.

Autoprint can only be used when the SP Terminal executive is loaded in the terminal. It is turned ON and OFF on the various terminals as follows:

**Table 3-7. Controlling Autoprint**

	<b>12080/90 and 12140/42 (P8/P9 &amp; P12/14)</b>	<b>9747 (P5)</b>	<b>9745 (P4)</b>
Keyboard ON	CTRL+F13 CTRL+F14	F4 F5	F4 F5
OFF			
Program ON	Hex 1C Hex 1D	Hex 1C Hex 1D	Hex 1C Hex 1D
OFF			

When Autoprint is turned ON, the terminal screen is not be affected except that the 25th line displays a flashing 'P'. The only keys on the keyboard which remain operational are function key F14 (or F5), CTRL and BREAK.

The simplest method of operating Autoprint is to type the required English or RUNOFF command at the terminal keyboard and then, instead of pressing the RETURN key to issue the command, press CTRL and F13 (or just F4). All output to the terminal will then be routed to the Terminal Printer via the autoprint buffer. When the print job has been completed, press CTRL and F14 (or just F5) to return to normal terminal operation.

When the last data in the buffer has been sent to the printer, if Autoprint is left ON, the terminal prompts the CPU with a Backspace code. If, after five minutes, no further data has been received by the terminal, Autoprint is turned off.

---

The following examples show the correct method of implementing Autoprint from a Proc.

```
AP-TESTPROC1
001 PQN
002 C***** SET TERMINAL CHARACTERISTICS
003 MV #1 "TERM 132,65,0,0,2,08"
004 P
005 C***** TURN AUTOPRINT ON
006 T I28,+
007 C***** ISSUE ENGLISH COMMAND (Do not use N option)
008 MV #1 "SORT MD D/CODE"
009 P
010 C***** TURN AUTOPRINT OFF
011 C Send Decimal 29
012 T I29,+
013 C***** RESET TERMINAL CHARACTERISTICS
014 MV #1 "TERM 79,23,0,0,0,21"
015 P

AP-TESTPROC2
001 PQN
002 C***** SET TERMINAL CHARACTERISTICS
003 MV #1 "TERM 132,65,0,0,2,08"
004 P
005 C***** TURN AUTOPRINT ON
006 T X1C,+
009 C***** ISSUE ENGLISH COMMAND (Don not use N option)
010 HSORT MD BY D/CODE WITH D/CODE GT "A" D/CODE HEADING
    "'LL'MD'LL'"
011 P
012 C***** ISSUE TWO FORM FEEDS AND TURN AUTOPRINT OFF
013 T C,C,+
014 C Send Hex 1D
015 T X1D,+
016 C***** RESET TERMINAL CHARACTERISTICS
017 MV #1 "TERM 79,23,0,0,0,21"
018 P
```

## Spooled Printing

Spooled printing is available on all terminals except Model 9745/6. The SP terminal executive must be loaded in the terminal. when using the L option.

To direct spooler output to an attached terminal printer, assign a form-queue to the port associated with the terminal.

Now use PORT-DESPOOL (or its synonym PORTOUT), with option S or L, to direct the current jobs on that form-queue to the printer attached to the terminal on which the command is executed.

Usually option X is used as well so that when the form-queue is empty the terminal reverts to normal operation. For example,

```
PORT-DESPOOL (SX)
```

Alternatively, use the LOGON command to logon another port to an account which executes a LOGON Proc including that command.

The spooler then outputs all current jobs in the form-queue to the printer and returns control to the terminal when the form-queue is empty. While the printer is under spooler control, the terminal itself is entirely inoperable.

**Note:** Execution of PORT-DESPOOL without the 'X' option leaves the printer under spooler control, and the terminal inoperable, until you execute CTRL+BREAK then type **END**.

The option selected determines the protocol used. The S option uses the Direct Print protocol. The L option uses the Straight-through protocol which is supported with the SP terminal executive loaded. You are recommended to use the S option where possible.

---

# Chapter 4

## Printer Programming

This chapter provides programming information for some printers. Full details are given of all escape and control code sequences that are supported.

For all other printer models, refer to the manual supplied with the printer.

For printers that have not been recently supplied and are not covered in this manual, refer to the RealityX 3.1 Reference Manual or Reality System Manual General Utilities and Printing, or the appropriate printer manual.

## Introduction

Control codes and escape sequences are printer instructions. For example, you can use control codes and escape sequences to set print modes or move the print cursor in either the horizontal or vertical direction.

### Sending Control Codes to a Printer

To send a control code to the printer, use the DataBasic *PRINT statement* together with CHAR function to send the decimal value that follows to the printer as character. (Enclose the decimal value of the control code in parentheses.) End the statement with a colon(:) if required. The colon at the end of the PRINT statement stops the carriage return (CR) and line feed (LF) from going to the printer.

### Sending Escape Sequences to a Printer

Use the 'escape' code (ESC, decimal 27) to form 'escape' sequences. The ESC code tells the printer to interpret the characters that follow as a printer command. Upon receiving the last character in the sequence, the printer executes the command and leaves escape mode.

Send the ESC code to the printer in DataBasic using CHAR(27), which is the decimal representation. The sequence following ESC is represented either using the ASCII characters enclosed in quotes or CHAR functions with the decimal value (or a combination of these).

### Variables in Escape Sequences

All numbers in parentheses refer to locations in the ASCII code set. For example, if n in the code sequence ESC Cn is a variable that determines the number of lines in a form and you want a form length of 66 lines, then n = 66. The 66th location in the ASCII code set is the character "B". You can send this sequence to the printer in various ways. For example:

```
PRINT CHAR(27) : "C" : "B" :
```

or:

```
PRINT CHAR(27) : "C" : CHAR(66) :
```

## Model 11115/11115-X and 11130 150 and 300 LPM Line Printers

### Control Codes

The following control codes have the functions shown when received by these printers.

**Table 4-1. Control Codes**

Decimal Code	Function
06	Sets vertical pitch to 8 lines per inch on current line only.
10	Line feed
12	Form feed
13	Carriage Return
128	Skip to Channel 1 (Top of Form)
129	Skip to Channel 2
130	Skip to Channel 3
131	Skip to Channel 4
132	Skip to Channel 5
133	Skip to Channel 6
134	Skip to Channel 7
135	Skip to Channel 8
136	Skip to Channel 9
137	Skip to Channel 10
138	Skip to Channel 11
139	Skip to Channel 12
140	Not used except as Dummy Channel
238	Start EVFU load sequence
239	End EVFU load sequence

### Cataloged Page Length/EVFU Program

A cataloged DataBasic program is provided to enable the user to define the page (form) length and to define and load the EVFU. Page length is a function of the EVFU in as much as the EVFU must contain one channel definition (real or dummy) for each line on the page. The program by entering the command PRINTRONIX in response to the TCL prompt (:). Once the command has been entered, operation will proceed as follows :

```
ENTER FORM LENGTH IN INCHES? 11
```

can be a decimal value (i.e. 11.5) but must give a whole number when multiplied by 'number of lines per inch'.

```
IS THIS 6 OR 8 LINES PER INCH? 6
```

could be 8. This program does NOT set the number of lines per inch.

```
VFU CHANNEL SKIPS? Y
```

press RETURN only if not required.

```
CHANNEL 13 USED AS DUMMY CHANNEL  
CHANNEL,LINE#? 2,3
```

channel 1 (top of form) automatically defined by program.

```
CHANNEL,LINE#? 3,6  
CHANNEL,LINE#? 10,36  
CHANNEL,LINE#?
```

press RETURN only when finished.

### To Load the EVFU from Program

The EVFU can be loaded from any DataBasic program by including the following coding (for example):

```
* Initialise variable which will contain EVFU set-up string  
EVFU = ''  
* Assign 'Start-load' and 'End-load' codes to variables  
SLOAD = CHAR(238)  
ELOAD = CHAR(239)  
* Dimension on array (x) with as many elements as there are  
* lines per page (66 in this example).  
DIM X(66)  
* Assign dummy channel code (decimal 140) to all elements of  
array  
MAT X = 140  
* Assign Channel 1 (decimal 128) to line 1 of form. Channel 1  
* must always correspond to top of form  
X(1) = 128  
* Assign Channels as required by putting the relevant Channel  
* code in the element of the array which corresponds to the  
* required line number. Examples only are shown  
X(5) = 129  
X(10) = 130  
X(15) = 131
```

```

X(20) = 132
X(25) = 133
X(30) = 134
X(35) = 135
X(40) = 136
X(45) = 137
X(50) = 138
X(55) = 139
* Use loop to insert relevant coding and element values into
* EVFU set-up string
FOR LUP = 1 TO 66
    EVFU = EVFU:CHAR(X(LUP))
NEXT LUP
* Insert 'Start-Load' and 'End-Load' codes as beginning and end
* (respectively) of EVFU set-up string
EVFU = SLOAD:EVFU:ELOAD
* Turn printer on
PRINTER ON
* Output EVFU set-up string
PRINT EVFU
* Turn printer off
PRINTER OFF
END

```

The preceding routine can be embedded in a larger program (less the END statement) or used on its own. Alternatively, the user can design a different routine, the only requirement being that the printer must receive the EVFU set-up string in the correct format. The correct sequence is as follows:

START CODE (decimal 238)

CHANNEL 1 code (decimal 128)

Up to 131 further CHANNEL codes - depending on the number of lines per form. Any line not defined by a real Channel code must be assigned the Dummy Channel code of decimal 140. The first Channel code will correspond to line 2, the second will correspond to line 3 and so on.

END CODE (decimal 239)

### **To Utilise the EVFU from Program**

Once loaded, the EVFU may be utilised from any program simply by outputting the appropriate Channel code immediately prior to the data it is to affect. When received by the printer, the code will cause, in effect, a 'Skip to Channel x' and subsequent data will be printed on that line until either a carriage return or another Channel code is received.

## Changing the Number of Lines Per Inch

8 LPI pushbutton selected to 8 lines per inch (lamp lit): all lines will be printed at 8 lines per inch.

8 LPI pushbutton selected to 6 lines per inch (lamp not lit): the inter-line spacing following any line which contains a control code 06 will be reduced to give an effective vertical pitch of 8 lines per inch. The 06 code, selecting 8 lines per inch, is reset by carriage return, linefeed codes (CR,LF) and must therefore be included in every print line which it is to affect.

## Printable Characters

Models 11115 and 11130 print the following Northgate UK2 character set.

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2		!	"	#	£	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	

### Notes:

1. Hex 20 is a space.
2. The table does not show the actual typeface.

Model 11115-X prints the UK1 (ISO646 compatible) character set. UK1 is similar to the UK2 set shown above with the following differences:

Hex	3	4
2	£	\$

## Model 11161/11161-X 600 LPM Line Printers

### Control Codes

The following decimal control codes have the functions shown when received by the printer.

**Table 4-2. Control Codes**

Decimal Code	Function
10	Line feed
12	Form feed
13	Carriage return
108	EVFU Start Code - 6 lpi
109	EVFU Start Code - 8 lpi
110	EVFU Start Code - Panel setting lpi
111	EVFU End Code
128	Skip to Channel 1
to	to
139	Skip to Channel 12
144	Slew 0 lines
to	to
159	Slew 15 lines

### Cataloged Page Length/EVFU Program

A cataloged DataBasic program is provided to enable the user to define the page (form) length and to define and load the EVFU. Page length is a function of the EVFU in as much as the EVFU must contain one channel definition (real or dummy) for each line on the page. The program may be accessed from any account and is called by entering the command EVFU-SETUP in response to the TCL prompt (:). Once the command has been entered, operation will proceed as follows (example operator responses are emboldened):

```
ENTER FORM LENGTH IN INCHES? 11
```

can be a decimal value (i.e. 11.5). The integer part of the result, when multiplied by 'number of lines per inch', will be taken.

```
IS THIS 6 OR 8 LINES PER INCH? 6
```

could be 8. This program does NOT set the number of lines per inch.

```
VFU CHANNEL SKIPS ? Y
```

press RETURN only if not required.

```
CHANNEL,LINE#? 2,3  
CHANNEL,LINE#? 3,6  
CHANNEL,LINE#? 10,36
```

Channel 1 (Top of Form) automatically defined by program.

CHANNEL,LINE#?

press RETURN only when finished

The page length/EVFU information will now be loaded into the printer and terminal operation will return to TCL.

**Note:** Multiple channels can be assigned to the same line and a single channel may be assigned to several lines, but the same channel/line combination will result if duplicate assignments are attempted.

### To Load the EVFU from Program

Any program written to set up the EVFU must start by sending a 'start code' and finish with an 'end code'. The start code is selectable according to the line spacing setting required:

---

Start Code		
Dec	Hex	Line Space
108	X'6C'	6 l.p.i The LPI switch on the
109	X'6D'	8 l.p.i operator panel is ignored
110	X'6E'	Specified by the LPI switch on the operator panel.

---

The end code is X'6F' (CHAR (111))

Each print line, which may have multiple channels, is specified in 2 bytes as follows:-

byte 1 channels 1-6  
byte 2 channels 7-12

The format of the 2 byte code is:

Bit	7	6	5	4	3	2	1	0
First Byte	1	0	C6	C5	C4	C3	C2	C1
Second Byte	1	0	C12	C11	C10	C9	C8	C7

where  $C_n$  is set to one to select channel  $n$ .

So there will be a CHAR( $n$ ) value for each byte where

$n = 1$  = for channels 1 or 7  
 $= 2$  = for channels 2 or 8  
 $= 4$  = for channels 3 or 9  
 $= 8$  = for channels 4 or 10  
 $= 16$  = for channels 5 or 11  
 $= 32$  = for channels 6 or 12

To set multiple channels on one line, add the values. For example:

Byte 1 for channels 2 and 4  $n = 2+8=10$   
 Byte 2 for channels 7 and 9  $n = 1+4=5$

There is a maximum form depth of 143 lines.

The EVFU may be loaded from any DataBasic program by including the following coding (for example):

```

* MODEL 11161 EVFU Loading program
*
* Dimension an array (LINE) with as many pairs of elements as
* there are lines per page
DIM LINE(66,2) ; * 66 Lines for this example
*
* Assign dummy channel codes for all lines
MAT LINE = 0
*
* Set line 1 to channel 1 (Top of Form)
LINE (1,1) = 1
*
*
* Assign Channels as required by putting the relevant channel
* codes in the elements of the array which correspond to the
* required line number.
*
* For Example:
LINE(5,1) = 2; * Set line 5 to channel 2

```

```
LINE(10,1) = 4; * Set line 10 to channel 3
LINE(20,1) = 8; * Set line 20 to channel 4
LINE(30,1) = 20; * Set line 30 to channels 3 and 5
LINE(30,2) = 5; * and channels 7 and 9
LINE(50,2) = 32; * Set line 50 to channel 12
*
*
* Print each line individually since large pages could exceed
* the 140 character limit of a PRINT statement.
*
PRINTER ON
PRINT CHAR(108); * EVFU start code, 6 lines per inch
*
FOR LUP = 1 TO 66
  PRINT CHAR(LINE(LUP,1)): CHAR(LINE(LUP,2)):
NEXT LUP
*
PRINT CHAR(111); * EVFU end code.
PRINTER OFF
END
```

The preceding routine may be embedded in a larger program (less the END statement) or used on its own. Alternatively, the user may design a different routine, the only requirement being that the printer must receive the EVFU set-up string in the correct format as indicated at the beginning of this section.

Any line not defined by a real Channel code must be assigned the Dummy Channel code of decimal 0. Set line 1 to channel 1 (Top of Form).

### **To Utilise the EVFU from Program**

Once loaded, the EVFU may be utilised from any program simply by outputting the appropriate Channel code immediately prior to the data it is to affect. When received by the printer, the code will cause, in effect, a 'Skip to Channel x' and subsequent data will be printed on that line until either a carriage return or another Channel code is received.

## Printable Characters

Models 11161 prints the following Northgate UK2 character set

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2		!	"	#	£	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	

### Notes:

1. Hex 20 is a space.
2. The table does not show the actual typeface.

Model 11161-X prints the UK1 (ISO646 compatible) character set. UK1 is similar to the UK2 set shown above with the following differences:

Hex	3	4
2	£	\$

## Model 11180/11180-X/11199/11199-X High Volume Line Printers

### Control Codes

The following control codes have the functions shown when received by these printers.

**Table 4-3. Control Codes**

Decimal Code	Function
10	Line feed
12	Form feed
13	Carriage Return
128	Skip to Channel 1 (Top of Form)
to	to
139	Skip to Channel 12
144	Slew 0 lines
to	to
159	Slew 15 lines
238	Start EVFU load sequence
249	End EVFU load sequence

### Cataloged Page Length/EVFU Program

A cataloged DataBasic program is provided to enable the user to define the page (form) length and to define and load the EVFU. Page length is a function of the EVFU in as much as the EVFU must contain one channel definition (real or dummy) for each line on the page. The program may be accessed from any account and is called by entering the command EVFU-SETUP in response to the TCL prompt (:). Once the command has been entered, operation will proceed as follows (example operator responses are emboldened):

```
ENTER FORM LENGTH IN INCHES? 11
```

can be a decimal value (i.e. 11.5). The integer part of the result, when multiplied by 'number of lines per inch', will be taken.

```
IS THIS 6 OR 8 LINES PER INCH? 6
```

could be 8. This program does NOT set the number of lines per inch.

VFU CHANNEL SKIPS? Y

press RETURN only if not required.

CHANNEL,LINE#? 2,3  
 CHANNEL,LINE#? 3,6  
 CHANNEL,LINE#? 10,36

channel 1 (Top of Form) automatically defined by program.

CHANNEL,LINE#?

press RETURN only when finished.

The page length/EVPU information will now be loaded into the printer and terminal operation will return to TCL.

**Note:** Multiple channels can be assigned to the same line and a single channel may be assigned to several lines, but the same channel/line combination will result if duplicate assignments are attempted.

### To Load the EVFU from Program

Any program written to set up the EVFU must commence by sending a start code, CHAR(238), and finish with an end code, CHAR(239). Each print line, which may have multiple channels, is specified in 2 bytes as follows:-

Byte 1 channels 1-6  
 Byte 2 channels 7-12

The format of the 2 byte code is:

Bit	7	6	5	4	3	2	1	0
First byte	1	0	C6	C5	C4	C3	C2	C1
Second byte	1	0	C12	C11	C10	C9	C8	C7

where  $C_n$  is set to one to select channel n.

So there will be a CHAR(n) value for each byte where:

$n = 1$  = for channels 1 or 7  
= 2 = for channels 2 or 8  
= 4 = for channels 3 or 9  
= 8 = for channels 4 or 10  
= 16 = for channels 5 or 11  
= 32 = for channels 6 or 12

To set multiple channels on one line, add the values.

For example:

Byte 1 for channels 2 and 4  $n = 2+8=10$

Byte 2 for channels 7 and 9  $n = 1+4= 5$

There is a maximum form depth of 143 lines.

The EVFU may be loaded from any DataBasic program by including the following coding (for example):

```
* MODEL 11180/11199 EVFU Loading program
*
* Dimension an array (LINE) with as many pairs of elements as
* there are lines per page
DIM LINE(66,2) ; * 66 Lines for this example
*
* Assign dummy channel codes for all lines
MAT LINE = 0
*
* Set line 1 to channel 1 (Top of Form)
LINE (1,1) = 1
*
*
* Assign Channels as required by putting the relevant channel
* codes in the elements of the array which correspond to the
* required line number.
*
* For Example:
LINE(5,1) = 2; * Set line 5 to channel 2
LINE(10,1) = 4; * Set line 10 to channel 3
LINE(20,1) = 8; * Set line 20 to channel 4
LINE(30,1) = 20; * Set line 30 to channels 3 and 5
LINE(30,2) = 5; * and channels 7 and 9
LINE(50,2) = 32; * Set line 50 to channel 12
*
*
```

```

* Print each line individually since large pages could exceed
* the 140 character limit of a PRINT statement.
*
PRINTER ON
PRINT CHAR(238): ; * EVFU start code
*
FOR LUP = 1 TO 66
    PRINT CHAR(LINE(LUP,1)): CHAR(LINE(LUP,2)):
NEXT LUP
*
PRINT CHAR(239); * EVFU end code.
PRINTER OFF
END
    
```

The preceding routine may be embedded in a larger program (less the END statement) or used on its own. Alternatively, the user may design a different routine, the only requirement being that the printer must receive the EVFU set-up string in the correct format as indicated at the beginning of this section.

Any line not defined by a real Channel code must be assigned the Dummy Channel code of decimal 0. Set line 1 to channel 1 (Top of Form).

### To Utilise the EVFU from Program

Once loaded, the EVFU may be utilised from any program simply by outputting the appropriate Channel code immediately prior to the data it is to affect. When received by the printer, the code will cause, in effect, a 'Skip to Channel x' and subsequent data will be printed on that line until either a carriage return or another Channel code is received.

### Printable Characters

Models 11180 and 11199 print the following Northgate UK2 character set

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2		!	"	#	£	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	

**Notes:**

1. Hex 20 is a space.
2. The table does not show the actual typeface.

Model 11180-X and 11199-X print the UK1 (ISO646 compatible) character set. UK1 is similar to the UK2 set shown above with the following differences:

Hex	3	4
2	£	\$

## Printable Characters

### Character Set 1

L/H	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	SP	0	@	P	`	P	NUL	DLE	°	À	Ð	a	°	
1	SOL	DC1	!	1	A	Q	a	q	SOH	DC1	í	±	Á	Ñ	b	±
2	STX	DC2	"	2	B	R	b	r	STX	DC2	ó	²	Â	Ò	Ƴ	³
3	ETX	DC3	#	3	C	S	c	s	ETX	DC3	ú	³	Ã	Ó	p	≤
4	EOT	DC4	\$	4	D	T	d	t	EOT	DC4	ñ	´	Ä	Ô	s	[
5	ENQ	NAK	%	5	E	U	e	u	ENQ	NAK	Ñ	µ	Å	Õ	s	]
6	ACK	SYN	&	6	F	V	f	v	ACK	SYN	ª	¶	Æ	Ö	m	,
7	BEL	ETB	'	7	G	W	g	w	BEL	ETB	º	·	Ç	x	t	»
8	BS	CAN	(	8	H	X	h	x	BS	CAN	¿	,	È	Ø	F	•
9	HT	EM	)	9	I	Y	i	y	HT	EM	Ú	´	É	Ù	Q	·
A	LF	SUB	*	:	J	Z	j	z	LF	SUB	¿	°	Ê	Ú	w	.
B	VT	ESC	+	;	K	[	k	{	VT	ESC	1/2	»	Ë	Û	d	Ö
C	FF	FS	,	<	L	\	l		FF	FS	1/4	¼	Ì	Ü	∞	n
D	CR	GS	-	=	M	]	m	}	CR	GS	i	½	Í	Ý	Æ	2
E	SO	RS	.	>	N	^	n	~	SO	RS	«	¾	Î	Þ	î	þ
F	SI	US	/	?	O	_	o	DEL	SI	US	»	¿	Ï	ß	ç	SP

**Character Set 2**

L/H	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	SP	0	@	P	`	P	ç	É		°	À	Ð	a	°
1	SOL	DC1	!	1	A	Q	a	q	ü	æ	í	±	Á	Ñ	b	±
2	STX	DC2	"	2	B	R	b	r	é	Æ	ó	²	Â	Ò	Ƴ	³
3	♥	DC3	#	3	C	S	c	s	â	ô	ú	³	Ã	Ó	p	≤
4	♦	DC4	\$	4	D	T	d	t	ä	ö	ñ	´	Ä	Ô	s	{
5	♣	§	%	5	E	U	e	u	à	ò	Ñ	μ	Å	Õ	s	}
6	♠	SYN	&	6	F	V	f	v	ã	û	ª	¶	Æ	Ö	m	,
7	BEL	ETB	'	7	G	W	g	w	ç	ù	°	·	Ç	x	t	»
8	BS	CAN	(	8	H	X	h	x	ê	ÿ	¿	,	È	Ø	F	•
9	HT	EM	)	9	I	Y	i	y	è	ö	Ú	¹	É	Ù	Θ	•
A	LF	SUB	*	:	J	Z	j	z	è	ú		°	Ê	Ú	Ω	.
B	VT	ESC	+	;	K	[	k	{	ı	ç	1/2	»	Ë	Û	δ	√
C	FF	FS	,	<	L	\	l		î	£	1/4	¼	Ì	Ü	∞	n
D	CR	GS	-	=	M	]	m	}	ì	¥	ı	½	Í	Ý	∅	2
E	SO	RS	.	>	N	^	n	~	Ä	R	«	¾	Î	Ë	€	■
F	SI	US	/	?	O	_	o	DEL	À	f	»	¿	Ó	ß	∩	SP

---

# Index

## A

Autoprint 3-3, 3-21

## B

Bull utilities 2-60

## C

Conventions 1-5

Conversion utilities 2-29

## D

DataBasic, utilities 2-25

Direct printing 3-3, 3-8

## E

English utilities 2-23

Escape sequences 4-2

## H

Hardcopy printing 3-3, 3-20

## I

ICL 1900 utilities 2-39

ICL 2900 utilities 2-52

## L

LOGON command 3-24

## O

Obsolescent utilities 2-7

Overpunched numeric fields 2-26

## P

PORT-DESPOOL command 3-24

PORTOUT command 3-24

Print screen 3-3, 3-6

Printer, programming 4-1 to 4-18

Printers

11115-X/11115 4-3

11130 4-3

11161/11161-X 4-7

11180/11180-X 4-12

11199/11199-X 4-12

Printing via terminals 3-2

Proc utilities 2-12

## R

READ-1900 verb 2-40

READ-2900 verb 2-52, 2-56

Relocated utilities 2-4

**S**

Screen dump 3-6  
Server exits  
    summary of 2-9  
Spooled printing 3-4, 3-24  
Straight-through printing 2-16, 3-3, 3-10

**T**

TABLE-CHECK verb 2-35, 2-38  
TABLE-INPUT program 2-36  
TABLE-LOAD verb 2-35, 2-38  
Terminal printing techniques 3-2  
Terminal, executive 3-2

**U**

U0190 Polish string 2-14  
U019F stop<-->comma<-->space 2-29  
U01A4 ASCII <-> ICL 7-track 2-50  
U01A7 overpunch.to/from num 2-26  
U0200 read/write tape block 2-45  
U0201 read/write SOD sentinel 2-47  
U047F strip/pad lead zeros 2-28, 2-44,  
    2-48  
U10DD get system serial # 2-14  
U119B straight-through on 2-17  
U119F stop <-> comma 2-30  
U11A4 ICL 1900 bin<->dec 2-44, 2-48,  
    2-50  
U11A7 overpunch to/from num 2-26  
U1201 read/write EOD sentinel 2-47  
U147F get system time/date 2-19  
U219A ASCII<->Bull 7 track 2-60  
U219B buffer printer output 2-17  
U21A7 overpunch to/from num 2-27  
U21AC translate using table 2-35  
U2201 tape block-size 2-49  
U247F get line no. + acc. name 2-20  
U315E number of ports 2-14  
U319A ASCII<->Bull 7 track 2-60  
U319B straight-through off 2-17  
U319F total multi/sub values 2-23

U31A4 strip/pad trail. spaces 2-25, 2-44,  
    2-48  
U31AC reduce rptd chars to one 2-32  
U347F error handling routine 2-20  
U40E0 toggle msg print 2-25  
U419F total attr + values 2-23  
U41A4 check tape attached 2-18  
U41A9 <> to () 2-32  
U41AC input and reduce chars 2-27  
U447F disable break key 2-21  
U44AF number of tape units 2-22  
U50BB get process no./acct 2-14  
U519F <> to/from () 2-31  
U51AA reversing a string for output 2-33  
U547F enable break key 2-21  
U647F clear R flag 2-21  
U747F set R flag 2-21  
U847F get number of items selected 2-22  
U919F dec/hex -> binary 2-31  
UA070 referencing a specified multi-value  
    2-24  
User exits 2-2  
    obsolescent 2-7  
    relocated 2-4  
    summary of 2-9  
    summary of current 2-9  
    withdrawn 2-5

**V**

Verb utilities 2-35  
    summary of 2-9

**W**

Withdrawn utilities 2-5  
WRITE-1900 verb 2-39  
WRITE-2900 verb 2-52, 2-55