

# Programmer's Reference Manual

PANOS





# Programmer's Reference Manual

PANOS



PART NO 0410, 012  
ISSUE NO 1  
JULY 1985

© Copyright Acorn Computers Limited 1985

Neither the whole or any part of the information contained in, or the product described in, this manual may be reproduced in any material form except with the prior written approval of Acorn Computers Limited (Acorn Computers).

The product described in this manual and products for use with it, are subject to continuous developments and improvement. All information of a technical nature and particulars of the product and its use (including the information in this manual) are given by Acorn Computers in good faith.

In case of difficulty please contact your supplier. Deficiencies in software and documentation should be notified in writing, using the Acorn Scientific Fault Report Form to the following address:

Sales Department  
Scientific Division  
Acorn Computers Ltd  
Fulbourn Road  
Cherry Hinton  
Cambridge  
CB1 4JN

All maintenance and service on the product must be carried out by Acorn Computers' authorised agents. Acorn Computers can accept no liability whatsoever for any loss or damage caused by service or maintenance by unauthorised personnel. This manual is intended only to assist the reader in the use of the product, and therefore Acorn Computers shall not be liable for any loss or damage whatsoever arising from the use of any information or particulars in, or any error or omission in, this manual, or any incorrect use of the product.

Published by Acorn Computers Limited, Fulbourn Road, Cherry Hinton, Cambridge CB1 4JN.

Within this publication the term BBC is used as an abbreviation for the British Broadcasting Corporation.

**NOTE:** A User Registration Card is supplied with the hardware. It is in your interest to complete and return the card. Please notify Acorn Scientific at the above address if this card is missing.

ISBN 0 907876 39 0 Acorn Scientific

# Contents

	Introduction	1
1	The user library	3
2	Errors	5
2.1	GetErrorMessage	9
2.2	SetErrorInformation	10
2.3	GetErrorInformation	11
3	Argument decoding	13
3.1	ArgumentInit	20
3.2	DecodeInit	23
3.3	GetStringArg	24
3.4	GetStateArg	25
3.5	GetBooleanArg	26
3.6	GetIntegerArg	27
3.7	GetCardinalArg	28
3.8	GetNumberOfValues	29
3.9	GetPresence	30
3.10	Substitute	31
3.11	DecodeEnd	32
4	Data conversion	33
4.1	StringToInteger	34
4.2	StringToCardinal	35
4.3	IntegerToString	36
4.4	CardinalToString	37
4.5	BooleanToString	38
4.6	StringToBoolean	39
5	Store allocation	41
5.1	Allocate	42
5.2	AllocateWithTag	43
5.3	SetStoretag	44
5.4	Deallocate	45
5.5	DeallocateGroup	46
5.6	GetNewTag	47
5.7	ReturnTag	48
5.8	DeallocateTop	49
5.9	DeallocateBottom	50
5.10	SetHeapEnd	51

5.11	ResetHeapEnd	52
5.12	CurrentHeapEnd	53
5.13	GetStoreInformation	54
6	I/O Library	55
6.1	FindInput	58
6.2	FindOutput	59
6.3	FindUpdate	60
6.4	CloseStream	61
6.5	SelectInput	62
6.6	SelectOutput	63
6.7	SelectUpdate	64
6.8	SetErrorStream	65
6.9	SetControlStream	66
6.10	InputStream	67
6.11	OutputStream	68
6.12	ErrorStream	69
6.13	ControlStream	70
6.14	WriteByte	71
6.15	ReadByte	72
6.16	CurrentByte	73
6.17	BlockRead	74
6.18	BlockWrite	75
6.19	SWriteByte	76
6.20	SReadByte	77
6.21	SCurrentByte	78
6.22	SBlockRead	79
6.23	SBlockWrite	80
6.24	GetFileOffset	81
6.25	SetFileOffset	82
6.26	BytesOutstanding	83
6.27	EndOfFile	84
6.28	FlushOutput	85
6.29	SFlushOutput	86
6.30	DeviceType	87
6.31	StreamType	89
6.32	SetTabs	90
6.33	GetTabs	91
7	File Support	93
7.1	GetDateStamp	94
7.2	SetDateStamp	95

7.3	Touch	96
7.4	RenameFile	97
7.5	DeleteFile	98
7.6	PhysicalFileName	99
7.7	SetWorkingDirectory	100
7.8	GetWorkingDirectory	101
7.9	LoadFile	102
7.10	SaveFile	103
7.11	PhysicalDirRead	104
7.12	InitDirRead	105
7.13	GetDirEntry	106
7.14	EndDirRead	107
7.15	IsWild	108
7.16	FileReplace	109
7.17	Expand	110
7.18	GetFileInformation	112
7.19	SetFileInformation	114
7.20	CreateFile	115
7.21	CreateDirectory	116
8	Loader	117
8.1	DeclareProc	118
8.2	DeclareData	119
9	Random numbers	121
9.1	Random	122
9.2	SetRandomSeed	123
10.	Time and date	125
10.1	BinaryTime	126
10.2	SetBinaryTime	127
10.3	BinaryTimeOfStandardTime	128
10.4	BinaryTimeOfTextualTime	129
10.5	StandardTimeOfBinaryTime	130
10.6	TextualTimeOfBinaryTime	131
10.7	Time	132
10.8	StandardTime	133
10.9	Date	134
10.10	TimeAndDate	135
11.	Condition Handlers	137
11.1	Initialise	148
11.2	Stop	149
11.3	Exception	150

11.4	Diagnose	152
11.5	DescribeFrame	153
11.6	DescribeModuleData	154
11.7	Unwind	155
11.8	Reserved	156
11.9	Signal	157
11.10	CallHandler	158
11.11	DeclareConditionHandler	159
12.	Asynchronous events	161
12.1	DeclareEventHandler	163
12.2	RemoveEventHandler	164
12.3	EventStatus	165
12.4	SetEventStatus	166
13.	Global String Variables	167
13.1	SetGlobalString	169
13.2	GetGlobalString	170
13.3	DeleteGlobalString	171
13.4	GetGlobalStringName	172
14.	Program Control	173
14.1	Call	174
14.2	Run	175
14.3	Obey	176
14.4	Invoke	177
14.5	CallRunOrObey	178
14.6	Name	179
14.7	FileName	180
14.8	Stop	181
14.9	SetKnownCommandsPath	182
14.10	Arguments	183
14.11	Verbosity	184
14.12	IdentifyRequired	185
14.13	HelpRequired	186
14.14	SwitchRequired	187
14.15	VerbosityRequired	189
15.	Command line interpreter	191
15.1	InterpretString	192
15.2	InterpretCommands	193
16.	Wild symbol expansion	195
16.1	Match	196
16.2	Replace	197

17.	BBC Library	199
17.1	OSByte	200
17.2	OSWord	201
17.3	OSFile	202
	Appendix A Panos-generated Errors	203
	Appendix B	209



# Introduction

This document describes the programmer's interface to Panos, the operating system for Acorn Cambridge Series computers. Panos rests on the low level machine support presented by Pandora, and provides a runtime system to support a range of high level languages.

The user gains access to the functionality provided by Panos via:

- A command line interpreter (CLI)
- A collection of utility programs
- The runtime library

The Panos command line interpreter, utility programs, and other user-interface related matters are described in the *Panos Guide to Operations*.

The majority of this document is taken up by a description of the Panos library. Each chapter deals with a particular module which contains one or more procedures of a given class, e.g. random numbers, command handling.

Appendix A of this manual lists the error codes associated with the user library procedures.

All procedures are described using a pseudo-language notation which lists the number and type of the parameters and results. Parameters and results are passed according to the rules described in the document *Panos Technical Reference Manual*. An informal introduction to this pseudo-language is given in chapter 1.

The following convention is observed:

Numbers not in decimal are prefixed by their base, for example 16\_1A is decimal 26; -2\_1010 is -10 in decimal.



# 1 The user library

All explicit communication with Panos from a user program is via the user library procedures. There are two ways by which Panos informs the user of errors; either the library procedure returns an error status (which should be checked explicitly by the caller) or an error exception is signalled. All library routines are provided in two versions, with the variant that generates an exception on error having its name prefixed by 'X'.

The library of Panos is divided up into several modules (see 'Acorn 32000 object format specification' in the *Panos Technical Reference Manual* for a description of the term 'module'). Each group of procedures described in the following chapters resides in a separate module. The module name is given at the bottom of each page for procedure descriptions.

Procedures are described in terms of a pseudo-programming language. The method of interfacing with real languages such as FORTRAN 77 and Pascal depends on the procedure calling system of each language. Most languages under Panos conform to the Acorn inter-language calling standard. In Acorn 32000 ISO Pascal, for example, a Panos library procedure can be accessed by the IMPORT directive.

For example, the method for importing the procedure SetKnownCommandsPath is:

```
type
    string=packed array[1..15] of char; { bound specified as required }
    ....
    ....
import function SetKnownCommandsPath(path:string;len:integer):integer
    ....
    status := SetKnowncommandsPath('$..Panoslib, @ ',13)
    ....
```

Full details of the inter-language calling standard and the Acorn Object Format produced by Acorn compilers are given in the *Panos Technical Reference Manual*. Explanation of how this maps into a particular language's calling system is given in that language's Reference Manual.

The procedures in this manual are described by the following syntax:

```

<procedure description> ::= <procedure name>(<parameter list>);
                               <result list>
<parameter list>           ::= | <parameter> <parameter list>
<result list>              ::= | <result> <result list>
<parameter>                ::= <parameter type>:<parameter name>
<result>                    ::= <result type>:<result name>
<parameter type>           ::= STRING | <base type> | <base type>REF
<result type>              ::= STRING | <base type>
<base type>                 ::= INTEGER | CARDINAL
                               | RECORD(<format name>)
                               | ADDRESS
                               | HIDDEN

```

These types are described in the inter-language calling standard section of the *Panos Technical Reference Manual* with **HIDDEN** being 32-bit raw binary.

## 2 Errors

This chapter describes the facilities provided for error handling; all of the procedures reside in moduleError . Many of the procedures in the Panos library return a 32-bit status code. On error the top bit of the status code is set, so the number is always negative. The other 31 bits are divided into fields which provide information about the error type, in which module it occurred and so on.

When an error occurs in a system module, it will call the procedure SetErrorInformation. This takes an error code and assigns an 'information string' to this error. The error number is then returned to the caller.

The calling program uses GetErrorMessage to convert the returned error number into three information strings: the message, the name of the system facility which detected the error and the name of the system facility which was initially called by the user.

Here is a typical sequence of events which might result in the error-handling procedures being called:

A program calls the procedure GetDateStamp to find the date at which a named file was created. The name is supplied as the string parameter 'DFS::0.\$prog1'. However, the named file does not exist on the filing system, so an error is generated. Suppose the basic error message is:

File % not found

The system will set the information for the error to the filename, i.e. 'DFS::0.\$prog1'. Thus when the user calls GetErrorMessage using the returned error code, the three strings returned will be:

```
BBC
File
File DFS::0.$prog1 not found
```

where BBC is the detecting facility (i.e. the module which discovered that the file did not exist), File is the interface facility (i.e. the module that the user called in the first place) and 'File DFS::0.\$prog1 not found' is the error message with the error information substituted.

All error codes generated by the Panos system and associated software are 32-bit values with the most significant bit set. They are divided into a number of fields:

```

3 3 2 2          2 1          1 1
1 0 8 7          0 9          2 1          9 8          0

```

1 Info	InterfaceFacility	DetectingFacility	Reserved(=111)	ErrorCode
--------	-------------------	-------------------	----------------	-----------

The structure of this error code has been designed such that a simple user program can return a small negative number (range -1 to -512) to denote an error condition.

### Info

This field describes whether any additional information is available for this error (see `GetErrorInformation`). The values in the field have the following meanings:

- 0        None; the error has already been reported.
- 1-5     Information available: use the value of the 32-bit errorcode as a handle to `GetErrorInformation` to obtain it.
- 6        Used when error is being signalled (see module 'Handler') to denote the passing of an associated error buffer.
- 7        None.

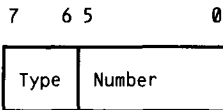
### InterfaceFacility

This gives the code of the facility called by the user. This may be converted to a string using `GetErrorMessage`. The table given below lists the facility names also.

### DetectingFacility

Gives the code of the facility which detected the error. This is also decoded by `GetErrorMessage`.

For both facility fields the code is structured as below:



The type field's two bits are:

Type 00 means that Number is a Panos facility as follows:

- |       |  |
|-------|--|
| 0     | 32000 Hardware exception                           |
| 1     | Data conversion                                    |
| 2     | Store  |
| 3     | IO   |
| 4     | Loader   |
| 5     | Random number generation                           |
| 6     | Time and date                                      |
| 7     | Condition handling.                                |
| 8     | Event handling                                     |
| 9     | Environment variables                              |
| 10    | Program control                                    |
| 11    | Pandora  |
| 12    | BBC  |
| 13    | Argument decoding (error in keystring format)      |
| 14    | Argument decoding (error in user parameter string) |
| 15    | File   |
| 16    | Reserved   |
| 17    | Command interpreter                                |
| 18    | Error handling                                     |
| 19    | Pattern matching                                   |
| 20-63 | Reserved   |

For Number 0 the ErrorCode (bits 0-8) is the 32000 Hardware Exception code (see the *Instruction Set Reference Manual* for details).

Type 2\_01 is reserved.

Type 2\_10 implies that Number is a Language Code (see 'Acorn 32000 Object Format specification' in the *Panos Technical Reference Manual*). The ErrorCode is a compiler error.

Type 2\_11 implies that Number is a Language Code. The ErrorCode is a run-time error.

## ErrorCode

See above.

*Note:* Facility 16\_FF is reserved for user programs.

The system procedure `GetErrorMessage` is provided to convert an error code into a textual message. The mapping between an error code and its corresponding text message is controlled by the system error file.

The error file is made up of a sequence of records separated by newline (LF) characters. Each record has the format

```
<ff> <ee> <skeleton error message>
```

where `<ff>` is a two digit facility number (base 16), e.g. 0F for File, and `<ee>` is the facility error code.

The `<skeleton error message>` is a printable message containing % characters where substitution of error information is required. The error information set for an error is made up of % separated fields.

When `GetErrorMessage` is building the message string it will substitute fields in the message skeleton from those in the corresponding position in the error information. If insufficient fields are provided in the error information then the value `<unknown>` will be used.

For example:

Skeleton	Error % on stream %
Information	87%12
Produces	Error 87 on stream 12
Skeleton	File % not found on FS %
Information	\$.file1
Produces	File \$.file1 not found on FS <unknown>

## 2.1 GetErrorMessage

```
GetErrorMessage(INTEGER:error);  
    INTEGER:Result  
    STRING:DetectingFacility  
    STRING:InterfaceFacility  
    STRING:Error message
```

```
XGetErrorMessage(INTEGER:error);  
    STRING:DetectingFacility  
    STRING:InterfaceFacility  
    STRING:Error message
```

### Action

Decodes the supplied error number and returns three strings describing:

- The System Facility which detected the Error,
- The System Facility which was called by the user and
- A text string describing the error.

Any additional information about the error is merged into the error message.

### Call

Error        The error number.

### Return

Result         $\geq 0$ , operation succeeded.  
               $< 0$ , operation failed (result = Error Code).

## 2.2 SetErrorInformation

```
SetErrorInformation(INTEGER:Error  
                  STRING:Information);  
                  INTEGER:Result
```

```
XSetErrorInformation(INTEGER:Error  
                   STRING:Information);  
                   INTEGER:Result
```

### Action

This caches the information string and sets the 'Info' field of the given error. The modified error can be used at a later stage as a parameter to GetErrorInformation which will endeavour to return the information string. For this to be successful only the Interface Facility in the resultant Error Code may be changed.

### Call

**Error**            An Error Code complete except for Interfacing Facility, which may be modified at a later stage.

**Information**    The information string to be associated with the error, e.g. 'DFS::0.\$fred%13'.

### Return

**Result**           Can be used both as an error code and as a handle to get back the information.

## 2.3 GetErrorInformation

GetErrorInformation(INTEGER:Error);  
    INTEGER: Result  
    STRING: Information

XGetErrorInformation(INTEGER:Error);  
    STRING Information

### Action

Endeavours to return any additional information associated with a system Error Code.

### Call

Error       The Error Code.

### Return

Result       > =0, Operation successful, information contains the  
              additional information.  
              <0, Operation failed (= Error Code).

А  
Б  
В  
Г  
Д  
Е  
Ж  
З  
И  
Й  
К  
Л  
М  
Н  
О  
П  
Р  
С  
Т  
У  
Ф  
Х  
Ц  
Ч  
Ш  
Щ  
Ъ  
Ы  
Ь  
Э  
Ю  
Я

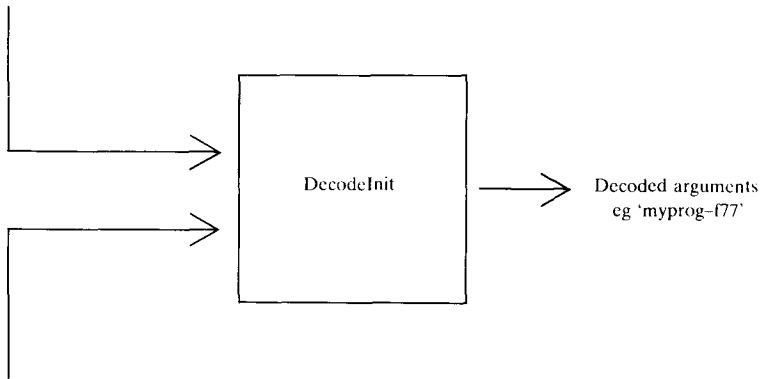
### 3 Argument decoding

When a program is called, its name may be followed by a list of arguments for use as parameters, e.g. source and object filenames for compilers, file specifications for filing system utilities etc.

This section describes the Panos procedures which can perform decoding of command line arguments. By utilising these procedures, all applications running under Panos can provide a uniform command interface to the user. The procedures all reside in module DecodeArg.

Decoding is performed by passing Panos a keystring which describes the format of arguments which the program expects. The procedure DecodeInit takes the keystring and processes the argument string accordingly. Decoded parameters may then be accessed by calling various other procedures, e.g. GetStringArg. This process is illustrated in figure 1.

keystring, eg 'source/a/c-f77 aol/k'



argument string, eg 'myprog'

Figure 1 Argument Decoding

A keystring is a sequence of keywords, which are qualified by control characters called option specifiers (e.g. /a and /e in the example above). These determine the type of keyword, and the number and type of arguments which may be associated with it.

When all of the arguments have been obtained by calls to the Get...Arg procedures, the application program should terminate the decoding process cleanly by calling DecodeEnd.

## The keystring

This section describes the format of the keystring in detail. As stated above, it is a list of keywords (separated by spaces or commas) which may be qualified by option specifiers.

Associated with each keyword may be a default argument list. This is used if the user does not supply any arguments on the command line for that keyword.

### Keyword name

A keyword name must begin with a letter and can contain letters or digits (Underscore, '\_', is treated as a lower case letter in keywords). The case of the keyword in the keystring is used to permit controlled abbreviation of the use of the keyword in the supplied argument string. (The actual case of the keyword in the supplied argument string is irrelevant.)

Abbreviation of a keyword when given in the argument string is possible if the keyword in the keystring ends in a sequence of lower case letters. Only lower case letters in the keystring keyword name may be truncated from the argument string keyword name. An abbreviation is not permitted if it is an otherwise legal truncation of more than one keyword.

An example is

'Name' in the keystring may be matched from the argument string by '-NAME' (or '-NAM', and '-NA' provided this is unambiguous). '-N', will not match. A '-NAME' will match in preference to abbreviations of any longer keyword names (i.e. is not ambiguous with keystring 'Name NAMEList').

Keyword names may be aliased by separating each alias with an equals sign, e.g. 'FROM = INput' will match '-FROM', '-INPUT', '-INPU', '-INP', and '-IN'.