The Professional Adventure Writing System

Technical Guide

CPM 2.2
or
CPM+/3

A Technical guide to:

The Professional Adventure Writer

An adventure writing system for CP/M computers.

Program: T.J.Gilberts, G.Yeandle and P.Wade
Graphics: A.Williams
Manuals: T.J.Gilberts and G.Yeandle

Acknowledgement

Thanks to Jill for putting up with the many hours I spent hammering away at the keyboard.

Contents

Overview

PAW can be divided into three main functional areas thus:

1/ The Source

The source (.SCE files) is a collection of tables, which contain all the information to define an adventure game.

2/ The Compiler

The Compiler checks the source file for errors and converts it into a PAW database (.PDB file) which the Interpreter can understand.

3/ The Interpreter

This is the real heart of PAW. The next chapter deals with the operation of the interpreter in detail, but it essentially fetches commands from the player and uses the information contained in the database to decode and respond to those commands.

The Interpreter

The following description of the interpreter should be read in conjunction with the flowcharts provided overleaf.

Initialise

The screen isn't cleared as this **always** occurs upon describing location zero. The flags are all zeroed except for; flag 37, the number of objects conveyable, which is set to 4; flag 52, the maximum weight of objects conveyable, which is set to 10; flags 46 & 47, the current pronoun ("IT" usually), which are set to 255 (no pronoun) and flag 1 which is set to be the number of objects carried but not worn. Note that clearing the flags has the effect that the game always starts at location zero. This is because flag 38, the current location of the player, is now zero.

Describe Current Location

If flag 2 is non zero it is decremented (reduced by 1). If it is dark (Flag 0 is non zero) and flag 3 is non zero then flag 3 is decremented. If it is dark, flag 4 is non zero and object 0 (the source of light) is absent, then flag 4 is decremented.

The screen is cleared if the current screen mode (contents of flag 40) is even.

If it is dark, and Object 0 is absent, then System message 0 (referred to as SM0 - "It's too dark to see anything") is displayed. Otherwise the location description is displayed **without** a NEWLINE.

Search Process Table 1

Flowchart 2 and the next section describe the scanning of a process table. Process table 1 is used mainly to contain the entries to add extra information to the current location description. E.g. details of open doors, objects present etc.

We now enter the main loop of the interpreter which deals with each time frame (i.e. each phrase extracted or each timeout on input which occurs) and the response to players commands.

Search Process Table 2

This contains the main control for PAW's turn at the game. It is used to implement the movements and actions of PSI's, the uncontrolled events such as bridges collapsing, and so on.

Get Phrase

If flags 5 to 8 are non zero they are decremented. If it is dark

(Flag 0 is non zero) and flag 9 is non zero then it is decremented. If it is dark and flag 10 is non zero it is decremented if object 0 is absent.

The parser is called to extract a phrase and convert it into a logical sentence - LS. If the input buffer is empty, a new input line is obtained from the player by printing a prompt and making a call to the input routine. The prompt will be the system message whose number is contained in flag 42 - a value of 0 will select one of system messages 2,3,4 or 5 in the ratio 30:30:30:10 respectively.

If the timeout option is selected, by making flag 48 contain a value greater than zero, then the input routine might time out. In this case SM35 ("Time passes...") is displayed and a return made to scan process 2 again.

A phrase is extracted and converted into the logical sentence; by converting any words present, that are in the vocabulary, into their word value and placing them in the required flags.

If no valid phrase can be found then SM6 ("I couldn't understand any of that") is displayed, and a return made to scan process 2.

Search Response Table

Turns (flags 31 and 32), which is the number of valid phrases extracted by the parser, is increased by one. Two flags are used to allow 256 lots of 256 turns (i.e. 65536).

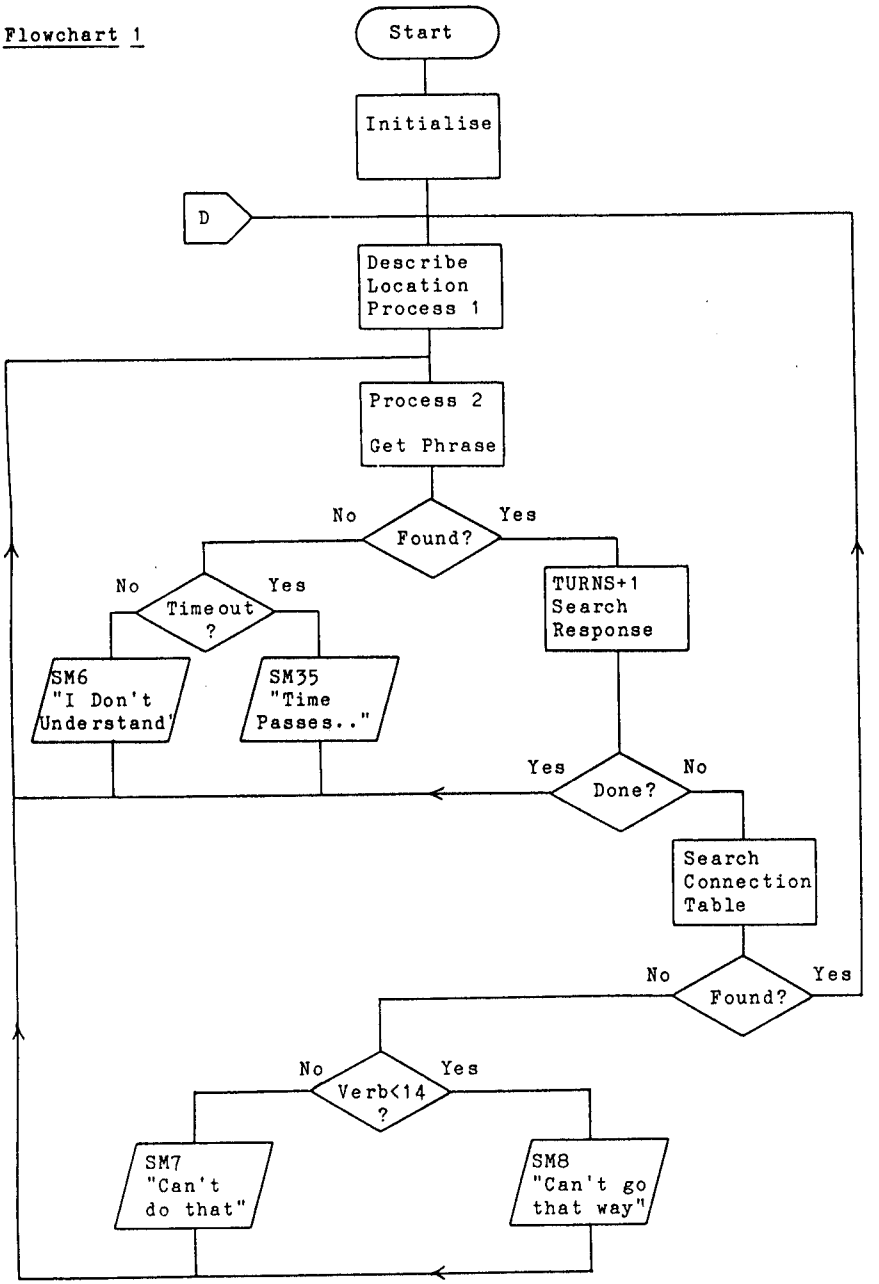
The Response table ie Process 0 is then scanned, for an entry which matches the Verb and Noun1 of the current LS, using the process table routine detailed below.

If it is successful in carrying out an action (i.e. If PAW executes at least one action other than NOTDONE) then a return is made to scan process table 2.

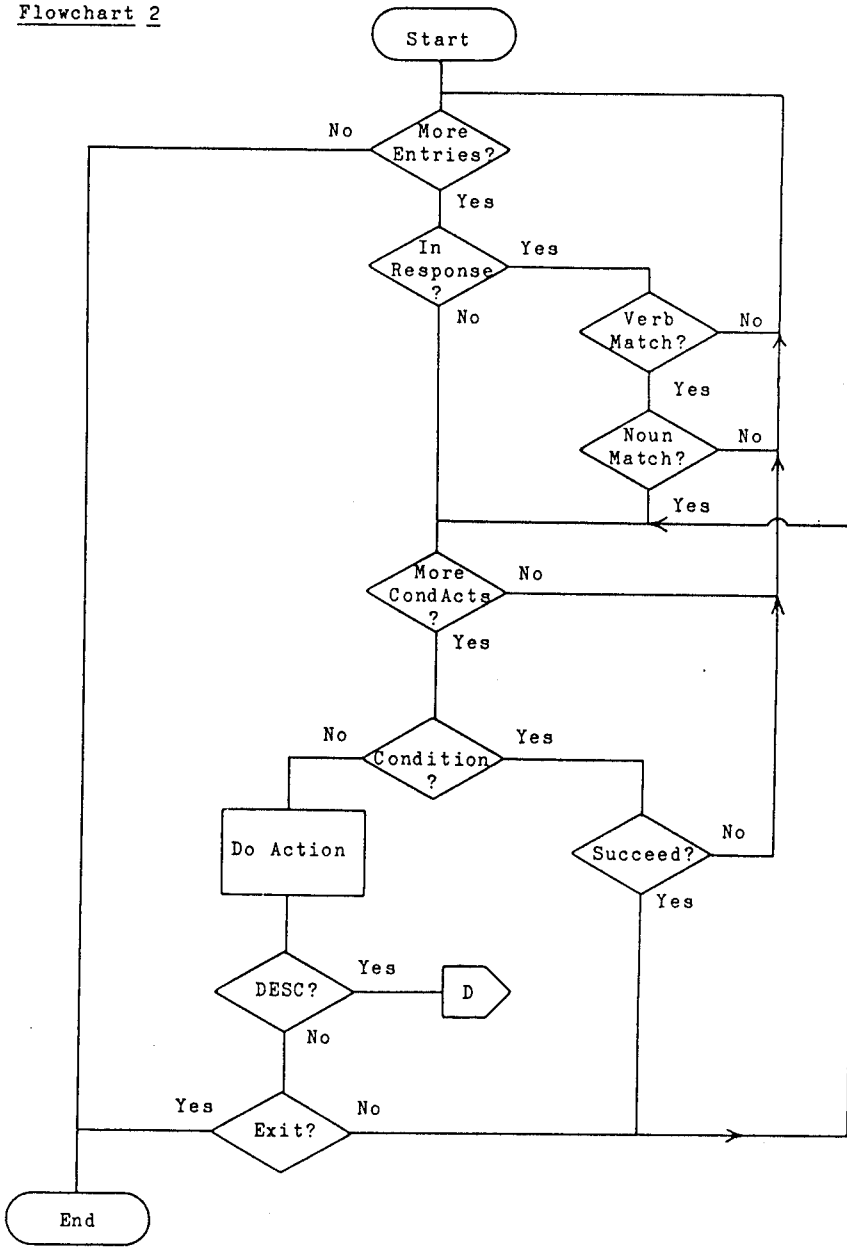
Search the Connections Table

A search is made of the connections table entry for the current location, for a word which matches the current LS Verb. If one is found then the players current location (flag 38) is set to be the number associated with the word. Then a return is made to describe the current location. Otherwise PAW prints SM8 ("I can't go in that direction") if the current LS Verb has a word value less than 14 or SM7 ("I can't do that") if it does not. In both cases a return is made to scan process table 2.

Flowchart 1


Start
Initialise
D
Describe Location Process 1
Process 2
Get Phrase
No
Found?
Yes
No
Time out ?
Yes
SM6 "I Don't Understand"
SM35 "Time Passes.."
TURNS+1 Search Response
Yes
Done?
No
Search Connection Table
No
Found?
Yes
No
Verb<14 ?
Yes
SM7 "Can't do that"
SM8 "Can't go that way"

Flowchart 2


Start
More Entries?
No
Yes
In Response ?
Yes
No
Verb Match?
No
Yes
Noun Match?
No
Yes
More CondActs ?
No
Yes
Condition ?
No
Yes
Do Action
Succeed?
No
Yes
DESC?
Yes
D
No
Exit?
Yes
No
End

The Interpreter

Scan a Process Table

Response is process table 0, so it will also be scanned by this section of PAW.

Essentially PAW will look at each entry in the table until it is exhausted - the table of entries, not PAW! Assuming there is an entry, it will ensure the Verb and Noun match those of the LS. The use of the word "_", as either the Verb or the Noun, will cause a match with any word in that part of the logical sentence. Thus an entry in Response of "_ _", will cause a trigger of the entry no matter what the LS.

PAW will then look at each CondAct in turn; if it is a condition, which succeeds, then PAW will look at the next condact. Otherwise it will drop out of the current condact list and look at the next entry, if present, in the table - an exception to this is the CondAct QUIT which, if it fails, will drop out of the table completely - this is not shown in flowchart 2 for clarity. If it is an action it will be carried out. Actions can be divided into five main groups:

1/ Desc; which will completely exit the execution of all tables (i.e. even if in a 10th level sub-process) and jump to describe the current location.

2/ END; (a group on its own) which will completely exit the execution of all tables and jump to initialise a new game.

3/ Exit; any action which will stop processing of the current table and exit to the calling table (or back to the main loop if in Response, Process 1 or 2). e.g. INVEN,DONE etc.

4/ Conditional Exit; any action which will stop processing of the current table and exit to the calling table (or back to the main loop if in Response, Process 1 or 2) if it fails to do its required function. e.g. GET, PUTIN etc. Otherwise it will continue with the next CondAct.

5/ Normal; any action which carries out its function, and allows PAW to continue looking at the next CondAct in the current entry. e.g. COPYFF, PLUS etc

It may be seen that QUIT acts like a type 4 action, but is still a condition, so it's a CondAct! The summary of CondActs at the end of this book shows which type each Action is.

Using the Interpreter

A>PAWINT CAVE	would run adventure CAVE.PDB from the default drive
A>PAWINT d:CAVE	would run adventure d:CAVE.PDB
A>PAWINT CAVE C	would load adventure CAVE.PDB from the default drive & give you the opportunity to Create a standalone copy of the adventure (eg CAVE.COM)

Diagnostics

Diagnostics are not available in a standalone adventure or following the use of action RAMSAVE. If diagnostics are available they can be switched on and off whenever the input routine is called. A null input (just press RETURN) will switch you into diagnostics and you will be able to look at and change the flag values and object locations. When diagnostics are first switched on you are in Flag mode and the value of flag 38 is displayed. The commands available are:-

RETURN	switch diagnostics off
O	switch to Object mode
F	switch to Flag mode
n	display the flag or object loc'n specified (n=0-255)
=n	alter the item displayed to the specified value

NB Flag 38 is protected from being set higher than the number of locations.
No object can be put at location 255.
Location 252 means not created.
Location 253 means worn.
Location 254 means carried.
You can only look at objects which have been defined.
Putting an object at a location which has not been defined is the same as making it not created.

The CondActs

There now follows a detailed description of each CondAct. They are divided into groups according to the subject they deal with in PAW; such as flags, objects etc and give some hints as to a possible use.

Several abbreviations are used in the descriptions as follows;

locno. is a valid location number.

locno+ is a valid location number or; 252 or "_" (meaning not-created), 253 or "WORN", 254 or "CARRIED" and 255 or "HERE" (which is converted into the current location of the player).

mesno. is a valid message.

sysno. is a valid system message.

flagno. is any flag (0 to 255).

procno. is a valid sub-process number.

word is word of the required type, which is present in the vocabulary, or "_" which ensures no-word - not an anymatch as normal.

value is a value from 0 to 255.

Conditions

There are four conditions which deal with testing the location of the player as follows;

AT locno.

Succeeds if the current location is the same as locno.

NOTAT locno.

Succeeds if the current location is different to locno.

ATGT locno.

Succeeds if the current location is greater than locno.

ATLT locno.

Succeeds if the current location is less than locno.

There are eight conditions which deal with the current location of an object;

PRESENT objno.

Succeeds if Object objno. is carried, worn or at the current location.

ABSENT objno.

Succeeds if Object objno. is not carried, not worn and not at the current location.

WORN objno.

Succeeds if object objno. is worn

NOTWORN objno.

Succeeds if Object objno. is not worn.

CARRIED objno.

Succeeds if Object objno. is carried.

NOTCARR objno.

Succeeds if Object objno. is not carried.

ISAT objno. locno+

Succeeds if Object objno. is at Location locno.

ISNOTAT objno. locno+

Succeeds if Object objno. is not at Location locno.

There are eight conditions which deal with the value and comparison of flags;

ZERO flagno.

Succeeds if Flag flagno. is set to zero.

NOTZERO flagno.

Succeeds if Flag flagno. is not set to zero.

EQ flagno. value

Succeeds if Flag flagno. is equal to value.

NOTEQ flagno. value

Succeeds if Flag flagno. is not equal to value.

GT flagno. value

Succeeds if Flag flagno. is greater than value.

LT flagno. value

Succeeds if Flag flagno. is set to less than value.

SAME flagno$_1$ flagno$_2$

Succeeds if Flag flagno$_1$ has the same value as Flag flagno$_2$.

NOTSAME flagno$_1$ flagno$_2$

Succeeds if Flag flagno$_1$ does not have the same value as Flag flagno$_2$.

There are five conditions to check for an extended logical sentence. It is best to use these conditions only if the specific word (or absence of word using "_") is needed to differentiate a situation. This allows greater flexibility in the commands understood by the entry.

ADJECT1 word

Succeeds if the first noun's adjective in the current LS is word.

ADVERB word

Succeeds if the adverb in the current LS is word.

PREP word

Succeeds if the preposition in the current LS is word.

NOUN2 word

Succeeds if the second noun in the current LS is word.

ADJECT2 word

Succeeds if the second noun's adjective in the current LS is word.

The following condition is for random occurrences. You could use it to provide a chance of a tree falling on the player during a lightning strike or a bridge collapsing etc. Do not abuse this facility, always allow a player a way of preventing the problem; such as rubber boots for the lightning, or similar.

CHANCE percent

Succeeds if percent is less than or equal to a random number in the range 1-100 (inclusive). Thus a CHANCE 50 condition would allow PAW to look at the next CondAct only if the random number generated was between 1 and 50, a 50% chance of success.

A single condition to test for a timeout;

TIMEOUT

Will succeed if the last request for input from the player was allowed to timeout. For example an entry in Process table 2 of;

```
_       _       TIMEOUT
                MESSAGE    0
```

(where message 0 read "Come on sleepy") could be created, perhaps with a CHANCE or a count of time outs to make it less monotonous!

The true CondAct;

QUIT

SM12 ("Are you sure?") is printed and the input routine called. Will succeed if the player replies with a word which starts with the first letter of SM30 ("Y") to the prompt. If not then Actions NEWTEXT & DONE are performed.

Actions

There are nineteen actions to deal with the manipulation of object positions;

GET objno.

If Object objno. is worn or carried, SM25 ("I already have the _.") is printed and actions NEWTEXT & DONE are performed.

If Object objno. is not at the current location, SM26 ("There isn't one of those here.") is printed and actions NEWTEXT & DONE are performed.

If the total weight of the objects carried and worn by the player plus Object objno. would exceed the maximum conveyable weight (Flag 52) then SM43 ("The _ weighs too much for me.") is printed and actions NEWTEXT & DONE are performed.

If the maximum number of objects is being carried (Flag 1 is greater than, or the same as, Flag 37), SM27 ("I can't carry any more things.") is printed and actions NEWTEXT & DONE are performed. In addition any current DOALL loop is cancelled.

Otherwise the position of Object objno. is changed to carried, Flag 1 is incremented and SM36 ("I now have the _.") is printed.

DROP objno.

If Object objno. is worn then SM24 ("I can't. I'm wearing the _.") is printed and actions NEWTEXT & DONE are performed.

If Object objno. is at the current location (but neither worn nor carried), SM49 ("I don't have the _.") is printed and actions NEWTEXT & DONE are performed.

If Object objno. is not at the current location then SM28 ("I don't have one of those.") is printed and actions NEWTEXT & DONE are performed.

Otherwise the position of Object objno. is changed to the current location, Flag 1 is decremented and SM39 ("I've dropped the _.") is printed.

WEAR objno.

If Object objno. is at the current location (but not carried or worn) SM49 ("I don't have the _.") is printed and actions NEWTEXT & DONE are performed.

If Object objno. is worn, SM29 ("I'm already wearing the _.") is printed and actions NEWTEXT & DONE are performed.

If Object objno. is not carried, SM28 ("I don't have one of those.") is printed and actions NEWTEXT & DONE are performed.

If Object objno. is not wearable (as specified in the object definition section) then SM40 ("I can't wear the _.") is printed and actions NEWTEXT & DONE are performed.

Otherwise the position of Object objno. is changed to worn, Flag 1 is decremented and SM37 ("I'm now wearing the _.") is printed.

REMOVE objno.

If Object objno. is carried or at the current location (but not worn) then SM50 ("I'm not wearing the _.") is printed and actions NEWTEXT & DONE are performed.

If Object objno. is not at the current location, SM23 ("I'm not wearing one of those.") is printed and actions NEWTEXT & DONE are performed.

If Object objno. is not wearable (and thus removable) then SM41 ("I can't remove the _.") is printed and actions NEWTEXT & DONE are performed.

If the maximum number of objects is being carried (Flag 1 is greater than, or the same as, Flag 37), SM42 ("I can't remove the _. My hands are full.") is printed and actions NEWTEXT & DONE are performed.

Otherwise the position of Object objno. is changed to carried. Flag 1 is incremented and SM38 ("I've removed the _.") printed.

CREATE objno.

The position of Object objno. is changed to the current location and Flag 1 is decremented if the object was carried.

DESTROY objno.

The position of Object objno. is changed to not-created and Flag 1 is decremented if the object was carried.

SWAP $objno_1$ $objno_2$

The positions of the two objects are exchanged. Flag 1 is not adjusted. The currently referenced object is set to be Object $objno_2$.

PLACE objno. locno+

The position of Object objno. is changed to Location locno. Flag 1 is decremented if the object was carried. It is incremented if the object is placed at location 254 (carried).

PUTO locno+

The position of the currently referenced object (i.e. that object whose number is given in flag 51), is changed to be Location locno. Flag 54 remains its old location. Flag 1 is decremented if the object was carried. It is incremented if the object is placed at location 254 (carried).

PUTIN objno. locno.

If Object objno. is worn then SM24 ("I can't. I'm wearing the _.") is printed and actions NEWTEXT & DONE are performed.

If Object objno. is at the current location (but neither worn nor carried), SM49 ("I don't have the _.") is printed and actions NEWTEXT & DONE are performed.

If Object objno. is not at the current location, but not carried, then SM28 ("I don't have one of those.") is printed and actions NEWTEXT & DONE are performed.

Otherwise the position of Object objno. is changed to Location locno. Flag 1 is decremented and SM44 ("The _ is in the"), a description of Object locno. and SM51 (".") is printed.

TAKEOUT objno. locno.

If Object objno. is worn or carried, SM25 ("I already have the _.") is printed and actions NEWTEXT & DONE are performed.

If Object objno. is at the current location, SM45 ("The _ isn't in the"), a description of Object locno. and SM51 (".") is printed and actions NEWTEXT & DONE are performed.

If Object objno. is not at the current location and not at Location locno. then SM52 ("There isn't one of those in the"), a description of Object locno. and SM51 (".") is printed and actions NEWTEXT & DONE are performed.

If Object locno. is not carried or worn, and the total weight of the objects carried and worn by the player plus Object objno. would exceed the maximum conveyable weight (Flag 52) then SM43 ("The _ weighs too much for me.") is printed and actions NEWTEXT & DONE are performed.

If the maximum number of objects is being carried (Flag 1 is greater than, or the same as, Flag 37), SM27 ("I can't carry any more things.") is printed and actions NEWTEXT & DONE are performed. In addition any current DOALL loop is cancelled.

Otherwise the position of Object objno. is changed to carried, Flag 1 is incremented and SM36 ("I now have the _.") is printed.

Note: No check is made, by either PUTIN or TAKEOUT, that Object locno. is actually present. This must be carried out by you if required.

DROPALL

All objects which are carried or worn are created at the current location (i.e. all objects are dropped) and Flag 1 is set to 0. This is included for compatibility with older writing systems. Note that a DOALL 254 will carry out a true DROP ALL, taking care of any special actions included.

The next six actions are automatic versions of GET, DROP, WEAR, REMOVE, PUTIN and TAKEOUT. They are automatic in that instead of needing to specify the object number, they each convert Noun(Adjective)1 into the currently referenced object - by searching the the object definition section. The search is for an object which is at one of several locations in descending order of priority - see individual descriptions. This search against priority allows PAW to 'know' which object is implied if more than one object with the same Noun description (when the player has not specified an adjective) exists; at the current location, carried or worn - and in the container in the case of TAKEOUT.

AUTOG

A search for the object number represented by Noun(Adjective)1 is made in the object definition section in order of location priority; here, carried, worn. i.e. The player is more likely to be trying to GET an object that is at the current location than one that is carried or worn. If an object is found its number is passed to the GET action. Otherwise if there is an object in existence anywhere in the game or if Noun1 was not in the vocabulary then SM26 ("There isn't one of those here.") is printed. Else SM8 ("I can't do that.") is printed (i.e. It is not a valid object but does exist in the game). Either way actions NEWTEXT & DONE are performed

AUTOD

A search for the object number represented by Noun(Adjective)1 is made in the object definition section in order of location priority; carried, worn, here. i.e. The player is more likely to be trying to DROP a carried object than one that is worn or here. If an object is found its number is passed to the DROP action. Otherwise if there is an object in existence anywhere in the game or if Noun1 was not in the vocabulary then SM28 ("I don't have one of those.") is printed. Else SM8 ("I can't do that.") is printed (i.e. It is not a valid object but does exist in the game). Either way actions NEWTEXT & DONE are performed

AUTOW

A search for the object number represented by Noun(Adjective)1 is made in the object definition section in order of location priority; carried, worn, here. i.e. The player is more likely to be trying to WEAR a carried object than one that is worn or here. If an object is found its number is passed to the WEAR action. Otherwise if there is an object in existence anywhere in the game or if Noun1 was not in the vocabulary then SM28 ("I don't have one of those.") is printed. Else SM8 ("I can't do that.") is printed (i.e. It is not a valid object but does exist in the game). Either way actions NEWTEXT & DONE are performed

AUTOR

A search for the object number represented by Noun(Adjective)1 is made in the object definition section in order of location priority; worn, carried, here. i.e. The player is more likely to be trying to REMOVE a worn object than one that is carried or here. If an object is found its number is passed to the REMOVE action. Otherwise if there is an object in existence anywhere in the game or if Noun1 was not in the vocabulary then SM23 ("I'm not wearing one of those.") is printed. Else SM8 ("I can't do that.") is printed (i.e. It is not a valid object but does exist in the game). Either way actions NEWTEXT & DONE are performed

AUTOP locno.

A search for the object number represented by Noun(Adjective)1 is made in the object definition section in order of location priority; carried, worn, here. i.e. The player is more likely to be trying to PUT a carried object inside another than one that is worn or here. If an object is found its number is passed to the PUTIN action. Otherwise if there is an object in existence anywhere in the game or if Noun1 was not in the vocabulary then SM28 ("I don't have one of those.") is printed. Else SM8 ("I can't do that.") is printed (i.e. It is not a valid object but does exist in the game). Either way actions NEWTEXT & DONE are performed

AUTOT locno.

A search for the object number represented by Noun(Adjective)1 is made in the object definition section in order of location priority; in container, carried, worn, here. i.e. The player is more likely to be trying to get an object out of a container which is actually in there than one that is carried, worn or here. If an object is found its number is passed to the TAKEOUT action. Otherwise if there is an object in existence anywhere in the game or if Noun1 was not in the vocabulary then SM52 ("There isn't one of those in the"), a description of Object locno. and SM51 (".") is printed. Else SM8 ("I can't do that.") is printed (i.e. It is not a valid object but does exist in the game).

Either way actions NEWTEXT & DONE are performed

Note: No check is made, by either AUTOP or AUTOT, that Object locno. is actually present. This must be carried out by you - if required.

COPYOO objno$_1$ objno$_2$

The position of Object objno$_2$ is set to be the same as the position of Object Objno$_1$. The currently referenced object is set to be Object objno$_2$.

There are four actions which allow various parameters of objects to be; placed in flags, set from flags - for comparison or manipulation.

COPYOF objno. flagno.

The position of Object objno. is copied into Flag flagno. This could be used to examine the location of an object in a comparison with another flag value. e.g.

```
COPYOF      1       11
SAME        11      38
```

could be used to check that object 1 was at the same location as the player - although ISAT 1 255 would be better!

COPYFO flagno. objno.

The position of Object objno. is set to be the contents of Flag flagno. An attempt to copy from a flag containing 255 will result in a run time error. Setting an object to an invalid location will still be accepted as it presents no danger to the operation of PAW.

WHATO

A search for the object number represented by Noun(Adjective)1 is made in the object definition section in order of location priority; carried, worn, here. This is because it is assumed any use of WHATO will be related to carried objects rather than any that are worn or here. If an object is found its number is placed in flag 51, along with the standard current object parameters in flags 54-57. This allows you to create other auto actions (the tutorial gives an example of this for dropping objects in the tree).

WEIGH objno. flagno.

The true weight of Object objno. is calculated (i.e. if it is a container, any objects inside have their weight added - don't forget that nested containers stop adding their contents after ten levels) and the value is placed in Flag flagno. This will have a maximum value of 255 which will not be exceeded. If Object objno. is a container of zero weight, Flag flagno. will be cleared as objects in zero weight containers, also weigh zero!

Now ten actions to manipulate the flags;

SET flagno.

Flag flagno. is set to 255.

CLEAR flagno.

Flag flagno. is cleared to 0.

LET flagno. value

Flag flagno. is set to value.

PLUS flagno. value

Flag flagno. is increased by value. If the result exceeds 255 the flag is set to 255.

MINUS flagno. value

Flag flagno. is decreased by value. If the result is negative the flag is set to 0.

ADD flagno$_1$ flagno$_2$

Flag flagno$_2$ has the contents of Flag flagno$_1$ added to it. If the result exceeds 255 the flag is set to 255.

SUB flagno$_1$ flagno$_2$

Flag flagno$_2$ has the contents of Flag flagno$_1$ subtracted from it. If the result is negative the flag is set to 0.

COPYFF flagno$_1$ flagno$_2$

The contents of Flag flagno$_1$ is copied to Flag flagno$_2$.

RANDOM flagno.

Flag flagno. is set to a number from the Pseudo-random sequence from 1 to 100. This could be useful to allow random decisions to be made in a more flexible way than with the CHANCE condition.

MOVE flagno.

This is a very powerful action designed to manipulate PSI's. It allows the current LS Verb to be used to scan the connections section for the location given in Flag flagno. If the Verb is found then Flag flagno. is changed to be the location number associated with it, and the next condact is considered. If the verb is not found, or the original location number was invalid, then PAW considers the next entry in the table - if present. Thus you could consider that PAW carries out the following imaginary entries on exit from Response if no action has been done;

```
_    _    MOVE     38          ;Attempt to move player
          DESC                 ;Describe his new loc.

_    _    LT       33  14      ;Movement word?
          SYSMESS   7          ;"Can't go that way.."
          DONE

_    _    SYSMESS  8           ;"I can't do that"
```

This feature could be used to provide characters with Random movement in valid directions; by setting the LS Verb to a random movement word and allowing MOVE to decide if the character can go that way. Note that any special movements which are dealt with in Response for the player, must be dealt with separately for a PSI as well.

Three actions to manipulate the flags dealing with the player;

GOTO locno.

Changes the current location to locno. This effectively sets flag 38 to the value locno.

WEIGHT flagno.

Calculates the true weight of all objects carried and worn by the player (i.e. any containers will have the weight of their contents added up to a maximum of 255), this value is then placed in Flag flagno. This would be useful to ensure the player was not carrying too much weight to cross a bridge without it collapsing etc.

ABILITY value$_1$ value$_2$

This sets Flag 37, the maximum number of objects conveyable, to value$_1$ and Flag 52, the maximum weight of objects the player may carry and wear at any one time (or their strength), to be value$_2$. No checks are made to ensure that the player is not already carrying more than the maximum. GET and so on, which check the values, will still work correctly and prevent the player carrying any more objects, even if you set the value lower than that which is already carried!

There are three actions which deal with the manipulation of the flags for screen mode, format etc;

MODE mode option

There are four screen modes each controlled by Flag 40 and set using the MODE action thus:-

Mode 0

The screen is cleared before a location is described. SM32 ("More...") appears when the screen area fills.

Mode 1

The screen is not cleared before a location is described. SM32 ("More...") appears when the screen area fills.

Mode 2

The screen is cleared before a location is described. SM32 ("More...") does not appear when the screen area fills.

Mode 3

The screen is not cleared before a location is described. SM32 ("More...") does not appear when the screen area fills.

PROMPT sysno.

Causes System message sysno. to be displayed whenever PAW obtains a command line from the player. A value of 0 (default) will cause PAW to select one of SM2, SM3, SM4 or SM5 in the ratio 30:30:30:10 respectively. Note this does not affect the prompts displayed by the END or QUIT condacts.

TIME duration option

Allows input to be set to 'timeout' after a specific duration in 1 second intervals, i.e. the Process 2 table will be called again if the player types nothing for the specified period. This

action alters flags 48 & 49. 'option' allows this to also occur on ANYKEY and the "More..." prompt. In order to calculate the number to use for the option just add the numbers shown next to each item to achieve the required combination;

1 - While waiting for first character of Input only.
2 - While waiting for the key on the "More..." prompt.
4 - While waiting for the key on the ANYKEY action.

e.g. TIME 5 6 (option = 2+4) will allow 5 seconds of inactivity on behalf of the player on input, ANYKEY or "More..." and between each key press. Whereas TIME 5 3 (option = 1+2) allows it only on the first character of input and on "More...".

TIME 0 0 will stop timeouts (default).

Three actions to deal with the printing of flag values on the screen;

PRINT flagno.

The decimal contents of Flag flagno. are displayed without leading or trailing spaces. This is a very useful action. Say flag 100 contained the number of coins carried by the player, then an entry in a process table of:-

```
MES      10          ;"You have "
PRINT   100
MESSAGE  11          ;" gold coins."
```

could be used to display this to the player.

TURNS

SM17-20 "You have taken x turn(s)." is printed where x is Flag 31 + 256 * Flag 32.

SCORE

SM21-22 "You have scored x%" is printed where x is Flag 30.

Five actions to deal with screen output and control;

CLS

Clears the screen.

NEWLINE

Prints a carriage return/line feed.

MES mesno.

Prints Message mesno.

MESSAGE mesno.

Prints Message mesno., then carries out a NEWLINE action.

SYSMESS sysno.

Prints System Message sysno.

Three actions dealing with listing objects on the screen. The first two are controlled by/set the value of flag 53 as described in the chapter on objects.

LISTOBJ

If any objects are present then SM1 ("I can also see:") is printed, followed by a list of all objects present at the current location. If there are no objects then nothing (as in null, not the word!) is printed.

LISTAT locno+

If any objects are present then they are listed. Otherwise SM53 ("nothing.") is printed - note that you will usually have to precede this action with a message along the lines of "In the bag is;" etc. It would be possible to create an alternative to the INVEN action described next by using WORN & CARRIED as parameters for LISTAT.

INVEN

This action is not affected by the continuous object list flag for compatibility with older writing systems.

SM9 ("I have with me:-") is printed. If no objects are carried or worn SM11 ("Nothing at all.") is printed. Otherwise the object text for each object that is carried or worn is printed on a separate line. If an object is worn its object text is followed by SM10 (" (worn)"). Action DONE is then performed.

The two actions which completely exit Response/Process execution;

DESC

Will cancel any DOALL loop, any sub-process calls and make a jump to describe the current location.

END

SM13 ("Would you like to play again?") is printed and the input routine called. Any DOALL loop and sub-process calls are cancelled. If the reply does not start with the first character of SM31 a jump is made to Initialise. Otherwise the player is returned to CP/M.

Three exit table actions;

DONE

This action jumps to the end of the process table and flags to PAW that an action has been carried out. i.e. no more condacts or entries are considered. A return will thus be made to the previous calling process table, or to the start point of any active DOALL loop.

NOTDONE

This action jumps to the end of the process table and flags PAW that no action has been carried out. i.e. no more condacts or entries are considered. A return will thus be made to the previous calling process table or to the start point of any active DOALL loop. This will cause PAW to print one of the "I can't" messages if needed. i.e. if no other action is carried out and no entry is present in the connections section for the current Verb.

OK

SM15 ("OK") is printed and action DONE is performed.

Four actions to allow the current state of the game to be saved and restored;

SAVE

This action saves the current game position (to a .PGP file) on disc. In detail, SM60 ("Type in name of file.") is printed and the input routine is called to get the filename from the player in the form [d:]filename where d: is an optional drive. If the supplied filename is not acceptable SM59 ("File name error.") is printed and actions ANYKEY & DESC are performed. An attempt is made at making the disc read/write (in case the disc has been changed). If filename.$$$ exists it is deleted. If the directory is full SM57 ("Directory full.") is printed and actions ANYKEY & DESC are performed. filename.$$$ is created and the current game position is written to it. If the disc is found to be full SM58 ("Disc full.") is printed and actions ANYKEY & DESC are performed. filename.$$$ is closed. If filename.PGP exists it is deleted. filename.$$$ is renamed to filename.PGP and

action DESC is performed. If the close or rename of filename.$$$ fails SM56 ("I/O ERROR! FILE NOT SAVED!") is printed and actions ANYKEY & DESC are performed (however we could not reproduce this error during testing).

LOAD

This action loads a game position (from a .PGP file) from disc. In detail, a filename is obtained in the same way as for SAVE. If filename.PGP does not exist SM54 ("File not found.") is printed and actions ANYKEY & DESC are performed. The current game position is overwritten with the contents of filename.PGP and is checked to be compatible with the current game. If it is action DESC is performed otherwise SM55 ("File corrupt.") is printed, action ANYKEY is performed and the game is restarted.

RAMSAVE

In a similar way to SAVE this action saves all the information relevant to the game in progress not onto disc but into a memory buffer. This buffer is of course volatile and will be destroyed when the machine is turned off which should be made clear to the player.

RAMLOAD flagno.

This action is the counterpart of RAMSAVE and allows the saved buffer to be restored. The parameter specifies the last flag to be reloaded which can be used to preserve values over a restore, for example an entry of:

```
RAMLO _     COPYFF  30  255
            RAMLOAD 254
            COPYFF  255  30
            DESC
```

could be used to maintain the current score, so that the player can not use RAMSAVE/LOAD as an easy option for achieving 100%!

Note: unlike SAVE and LOAD the RAM actions allow the next Condact to be carried out. They should normally always be followed by a DESC in order that the game state is restored to an identical position.

The actions could be used to implement an OOPS command that is common on other systems to take back the previous move; by creating an entry in Process 2 (or Response) which does an automatic RAMSAVE every time the player enters a move.

Two actions to allow the game to be paused for a time or until a key is pressed;

ANYKEY

SM16 ("Press any key to continue") is printed and the keyboard is scanned until a key is pressed or until the timeout duration has elapsed if enabled.

PAUSE value

Pauses for value/50 secs. However, if value is zero then the pause is for 256/50 secs.

Two actions to deal with control of the parser;

PARSE

This action was designed to deal with speech to PSIs. Any string (i.e. a further phrase enclosed in quotes [""]) that was present in the players current phrase is converted into a LS - overwriting the existing LS formed originally for that phrase. If no phrase is present, or it is invalid, then PAW will look at the next condact. Otherwise the next entry is considered with the new LS of the speech made to the PSI. Because it overwrites the current LS it must be used only in a sub-process table, the table will have the form of:

```
_     _     PARSE                ;Always do this entry
            MESSAGE   x          ;"They don't understand"
            DONE

word  word  CondAct list         ;Any phrases PSI understands

_     _     MESSAGE   x          ;as above or different message
```

there will be two or more calling entries which will be similar to:

```
SAY   name  SAME    pos  38      ;Are they here?
            PROCESS   y          ;Decode speech..
            DONE                 ;LS destroyed so always DONE.

SAY   name  MESSAGE   z          ;"They are not here!"
            DONE
```

NEWTEXT

Forces the loss of any remaining phrases on the current input line. You would use this to prevent the player continuing without a fresh input should something go badly for his situation. e.g. the GET action carries out a NEWTEXT if it fails to get the required object for any reason, to prevent disaster with a sentence such as:

GET SWORD AND KILL ORC WITH IT

as attacking the ORC without the sword may be dangerous!

One action to deal with sound

BELL

Rings the keyboard bell ie CTRL G.

Several actions which are more difficult to classify;

PROCESS procno.

This powerful action transfers the attention of PAW to the specified Process table number. Note that it is a true subroutine call and any exit from the new table (e.g. DONE, OK etc) will return control to the condact which follows the calling PROCESS action. A sub-process can call (nest) further process' to a depth of 10 at which point a run time error will be generated.

DOALL locno+

Another powerful action which allows the implementation of an 'ALL' type command.

1 - An attempt is made to find an object at Location locno. If this is unsuccessful the DOALL is cancelled and action DONE is performed.

2 - The object number is converted into the LS Noun1 (and Adjective1 if present) by reference to the object definition section. If Noun(Adjective)1 matches Noun(Adjective)2 then a return is made to step 1. This implements the "Verb ALL EXCEPT object" facility of the parser.

3 - The next condact and/or entry in the table is then considered. This effectively converts a phrase of "Verb All" into "Verb object" which is then processed by the table as if the player had typed it in.

4 - When an attempt is made to exit the current table, if the DOALL is still active (i.e. has not been cancelled by an action) then the attention of PAW is returned to the DOALL as from step 1; with the object search continuing from the next highest object number to that just considered.

The main ramification of the search method through the object

definition section is; objects which have the Same Noun(Adjective) description (where the game works out which object is referred to by its presence) must be checked for in ascending order of object number, or one of them may be missed.

Use of DOALL to implement things like OPEN ALL must account for the fact that doors are often flags only and would have to be made into objects if they were to be included in a DOALL.

RESET locno+

This action is designed to allow the implementation of multi-part games where the objects which are not carried forward are reset to their starting location.

All objects which can be carried between parts must be present (with the same description) in each part. Any others may be reused within each part at will.

Any objects which are present at the current location are moved to Location locno. and the current location is set to be locno. Any other objects are set to their start locations as specified in the object definition section. No effect on flags. Action DESC is performed when complete.

The suggested method of its use is given in the chapter on multi-part games.

EXTERN value

Calls external routine with parameter value. There is a special chapter dealing with this subject.

Detailed Description of the Source file

The source file consists of a number of inter-related sections describing the adventure. The sections are:-

The Control (CTL) section

This section tells the compiler which drive to put the database on and what character has been chosen as a null word. Throughout this manual the null word is assumed to be an underline _.

The Vocabulary (VOC) section

Each entry in this section contains a word (or the first five characters), a word value and a word type. Words with the same word value and type are called synonyms.

The System Messages (STX) section

This section contains the messages used by the Interpreter which are numbered from 0 upwards. The description of the Interpreter shows when these messages are used. In addition extra messages can be inserted by the writer to provide messages for the game if so required.

The Message Text (MTX) section

This section contains the text of any messages which are needed for the adventure. The messages are numbered from 0 upwards.

The Object Text (OTX) section

This section, which has an entry for each object, contains the text which is printed when an object is described. An object is anything in the adventure which may be manipulated and objects are numbered from 0 upwards. Object 0 is assumed by the Interpreter to be a source of light.

The Location Text (LTX) section

This section, which has an entry for each location, contains the text which is printed when a location is described. The entries are numbered from 0 upwards and location 0 is the location at which the adventure starts.

The Connections (CON) section

This section has an entry for each location and each entry may either be empty (null) or contain a number of movements. A movement consists of a Verb (or conversion Noun) from the vocabulary followed by a location number. This means that any Verb (or conversion Noun) with that word value causes movement to that location. A typical entry could be:-

```
SOUTH    6
EAST     7
LEAVE    6
NORTH    5
```

which means that SOUTH or LEAVE or their synonyms cause movement to location 6, EAST or it's synonyms to location 7 and NORTH or it's synonyms to location 5.

Note 1. When the adventure is being played it is only the LS Verb which will cause movement.

Note 2. If a movement is performed by an entry in the Response table using the GOTO action, then it may not be needed in the Connections table, unless that entry is required for a PSI who can move unconditionally.

The Object Definition (OBJ) section

This section has an entry for each object which specifies:-

a) The location at which the object is situated at the beginning of the adventure.
b) The objects weight.
c) Whether the object is a container ie can contain other objects.
d) Whether the object can be worn/removed.
e) The noun and adjective associated with the object.

The Process tables (PRO) section

This section forms the heart of the source file providing the main game control.

The Response Table ie Process 0

Each entry contains the Verb and Noun for the LS the entry is to deal with followed by any number of condacts. When the adventure is played, if there is an entry in the table which matches the Verb and Noun1 of the LS entered then the condacts are performed. The condacts that may be present and the effect that they have is fully specified in the description of the Interpreter.

Process 1

Is scanned by PAW after a location is described, to allow any additional information which forms a part of the location description to be displayed.

Process 2

Is scanned by PAW every time frame. That is after every phrase

extracted from the player's input, or after every timeout on input.

Process 3 (and upwards)

These are optional and define sub-processes that can be referenced using the PROCESS action.

Syntax of The Source File

The source file consists of a number of sections which must be present in the correct order. A good example of a source file is START.SCE. Each line of the source file should be shorter than 256 characters. If not you may get strange results. Any line with a semicolon in column 1 is regarded as a comment. Blank lines are only allowed in the process tables

In the definitions of each section of the source file which follow:-

a) w-s means white space ie spaces or TAB characters.
b) Items in [ ] are optional.
c) EOL means End of line.

CTL

The Control section consists of 3 lines as follows:-

/CTL [comment] EOL
DBDRIVE [comment] EOL
NULLWORD [comment] EOL

Note a) DBDRIVE is the disk drive the database is to be written to. 1 character A-P.
b) NULLWORD is the character to be used as a null word. 1 character not a-Z or 0-9.

VOC

The Vocabulary section starts with the line:-

/VOC [comment] EOL

and can be followed by any number of lines as follows:-

[w-s] WORD w-s VALUE w-s WORDTYPE [w-s[;comment]] EOL

Note a) only the 1st 5 chars of WORD are significant,
b) they are converted to upper case & only A-Z 0-9 allowed,

c) duplicate words are not allowed,
d) VALUE is in range 1-254 (On the Spectrum version the range is 2-254),
e) WORDTYPE must be one of the following:- VERB
NOUN
ADJECTIVE
ADVERB
PREPOSITION
PRONOUN
CONJUGATION
f) In the interpreter Nouns < 20 can be used as verbs if no verb is entered. Verbs < 14 are used as movement words.

STX, MTX, OTX & LTX

The System Message Text section starts with the line:-

/STX [comment] EOL

The Message Text section starts with the line:-

/MTX [comment] EOL

The Object Text section starts with the line:-

/OTX [comment] EOL

The Location Text section starts with the line:-

/LTX [comment] EOL

Within each section each entry consists of:-

/n [w-s [comment]] EOL

then any number of text lines not starting with /

Note a) n must be consecutive & start at 0
b) STX limit is 60-255
c) MTX limit is 1-255
d) OTX limit is 1-255
e) LTX limit is 1-252

NB The text lines should only contain normal ASCII printable characters. The Compiler will give you a warning if it finds any control characters in the text. Any character with a value less than decimal 32 is considered a control code including TAB's. Note that the Compiler only gives a warning about control codes - If you leave them in it may confuse the Interpreter and some control codes may also confuse the Compiler.

The Compiler joins consecutive non-null lines with a space eg

```
I                       | will be printed by the Interpreter as
am big.                 |
So are you.             | I am big. So are you.
```

The Compiler converts null lines into carriage returns eg

```
I am                    | will be printed by the Interpreter as
big.                    |
                        | I am big.
So                      | So are you.
are you.
```

Similarly

```
                        | will be printed by the Interpreter as
I am big.               |
                        | I am big.
                        |
So are                  | So are you.
you.                    |
```

CON

The Connections section starts with the line:-

/CON [comment] EOL

Each entry consists of:-

/n [w-s [comment]] EOL

then any number of lines

[w-s] WORD w-s LOCNO [w-s[;comment]] EOL

Note a) There must be the same no of entries as in LTX.
b) n must start at 0 & be consecutive.
c) WORD must be in the vocabulary as a Verb (or Noun < 20).
d) LOCNO must be specified in LTX.

OBJ

The Object Definition section starts with the line:-

/OBJ [comment] EOL

Each line consists of:-

/n w-s LN w-s WT w-s CONT w-s WR w-s NOUN w-s ADJ [w-s[;com]] EOL

Note a) n must start at 0 & be consecutive.
b) There must be the same number of entries as OTX.
c) LN is the objects position at the start of the adventure and must be specified in LTX, or be 252-254, WORN, CARRIED or the null word character.
252 and the null word character mean not created
253 and WORN mean worn
254 and CARRIED mean carried
d) WT is the objects weight in the range 0-63.
e) CONT specifies if the object is a container. It can be Y or the null word.
f) WR specifies if the object can be worn/removed. It can be Y or the null word.
g) NOUN is a noun in the vocabulary which refers to the object or the null word.
h) ADJ is an adjective in the vocabulary which refers to the object or the null word.
i) If an object starts worn then WR must be Y.
j) You cannot specify an Adjective without a noun.

PRO

The Process section consists of a number of tables starting with the line:-

/PRO [w-s] n [w-s[comment]] EOL

Note a) n is the number of the Process table and must start at 0 and be consecutive.
b) You must always specify a minimum of 3 process tables ie Nos. 0-2.
c) Blank lines between entries or lines starting w-s; are ignored.

Each entry starts with:-

V w-s N w-s KEYWORD [w-s PARAM1 [w-s PARAM2]] [w-s[;comment]] EOL

then any number of:-

w-s KEYWORD [w-s PARAM1 [w-s PARAM2]] [w-s[;comment]] EOL

Note a) V must be a verb in the vocabulary, a noun < 20 in the vocabulary or the null word character.
b) N must be a noun in the vocabulary or the null word character.
c) KEYWORD can be a CONDITION or ACTION.

d) Anything starting in col 1 is assumed to be the start of the next entry.

The CONDITIONS & ACTIONS and their parameters are:-

Conditions

AT	locno	
NOTAT	locno	
ATGT	locno	
ATLT	locno	
PRESENT	objno	
ABSENT	objno	
WORN	objno	
NOTWORN	objno	
CARRIED	objno	
NOTCARR	objno	
CHANCE	percent	
ZERO	flagno	
NOTZERO	flagno	
EQ	flagno	value
GT	flagno	value
LT	flagno	value
ADJECT1	adjective	
ADVERB	adverb	
TIMEOUT		
ISAT	objno	locno+
PREP	preposition	
NOUN2	noun	
ADJECT2	adjective	
SAME	flagno	flagno
NOTEQ	flagno	value
NOTSAME	flagno	flagno
ISNOTAT	objno	locno+

Actions

INVEN		
DESC		
QUIT		
END		
DONE		
OK		
ANYKEY		
SAVE		
LOAD		
TURNS		
SCORE		
CLS		
DROPALL		
AUTOG		
AUTOD		
AUTOW		
AUTOR		
PAUSE	value	
GOTO	locno	
MESSAGE	mesno	
REMOVE	objno	
GET	objno	
DROP	objno	
WEAR	objno	
DESTROY	objno	
CREATE	objno	
SWAP	objno	objno
PLACE	objno	locno+
SET	flagno	
CLEAR	flagno	
PLUS	flagno	value
MINUS	flagno	value
LET	flagno	value
NEWLINE		
PRINT	flagno	
SYSMESS	smesno	
COPYOF	objno	flagno
COPYOO	objno	objno
COPYFO	flagno	objno
COPYFF	flagno	flagno
LISTOBJ		
EXTERN	value	
RAMSAVE		
RAMLOAD	flagno	
BELL		
ADD	flagno	flagno

SUB	flagno	flagno
PARSE		
LISTAT	locno+	
PROCESS	prono	
MES	mesno	
MODE	value	
TIME	value	value
DOALL	locno+	
PROMPT	smesno	
WEIGH	objno	flagno
PUTIN	objno	locno
TAKEOUT	objno	flagno
NEWTEXT		
ABILITY	value	value
WEIGHT	flagno	
RANDOM	flagno	
WHATO		
RESET	locno	
PUTO	locno+	
NOTDONE		
AUTOP		
AUTOT		
MOVE		

Note a) flagno & value are in the range 0-255.
b) percent is in the range 1-99.
c) objno must be defined in OTX.
d) mesno must be defined in MTX.
e) smesno must be defined in STX.
f) prono must be the number of a process table.
g) locno must be defined in LTX.
h) locno+ must be defined in LTX or be 252-255 or the null word character 'WORN', 'CARRIED' or 'HERE'.
i) adjective must be an adjective in the vocabulary or the null word character.
j) adverb must be an adverb in the vocabulary or the null word character.
k) preposition must be a preposition in the vocabulary or the null word character.
l) noun must be a noun in the vocabulary or the null word character.

LNK

A line starting with /LNK may appear anywhere in the source file and it forces the Compiler to switch to a different source file (but on the same drive). Note that everything following the /LNK statement in the current file is ignored. Its syntax is:-

/LNK [w-s] filename

eg /LNK srcfile2 will switch to SRCFILE2.SRC

Detailed Description of the Compiler

The Compiler has been specially designed so that it does not stop as soon as it finds an error. The processing is as follows:-

```
Process the command line.
If any errors found stop.
Open the print file .PRN (if required)
Process CTL
if no errors found open the database file .$$$ (if required)
if no errors found process VOC, STX, MTX, OTX & LTX
if no errors found process CON, OBJ & PRO
if no errors found process the end routines.
Print Compilation ends OK
   or Compilation ends with n WARNING(S)
   or Compilation ends with n ERROR(S)
   or Compilation ends with n ERROR(S) and n WARNING(S)
```

Using the Compiler

A>PAWCOMP CAVE	would compile CAVE.SCE on the default drive and create a file called CAVE.PDB.
A>PAWCOMP d:CAVE	would compile d:CAVE.SCE and create a file called CAVE.PDB.
A>PAWCOMP CAVE P	would compile CAVE.SCE on the default drive creating a file called CAVE.PDB & redirect compiler print to CAVE.PRN on the default drive.
A>PAWCOMP CAVE Pr	would compile CAVE.SCE on the default drive creating a file called CAVE.PDB & redirect compiler print to r:CAVE.PRN.
A>PAWCOMP CAVE N	would compile CAVE.SCE on the default drive without writing a PDB file (ie it just checks syntax).
A>PAWCOMP CAVE C	would compile CAVE.SCE on the default drive and compress the text so that the database takes less room.

A>PAWCOMP d:CAVE N C Pr gives a combination of the previous examples.

Remember P for a Print file
N for No database
and C to Compress the text

Compressing the text

If this option is selected a dictionary of 128 predefined common letter groupings is included in the database. During the compilation all texts (except object texts) are scanned for each of these 128 common letter groupings and if found they are replaced with a 1 byte token in the range 128-255. This method of compression can reduce the space occupied by text by over 40%.

Note a) Compressing text will take longer than not compressing text.

b) The 128 predefined common letter groupings are mainly in lower case. If your texts are mainly in upper case you may get a negative saving ie the database may occupy more space rather than less.

Errors and Warnings

Compiler WARNINGS

There are 3 types of warnings that can be issued:-

1) Database too big for current memory by n bytes

   The Interpreter will be unable to run the adventure with the current memory.

2) Control character present in text - value(n)

   The Compiler has found a character with value n (0-32 decimal). This character may confuse the Interpreter.

3) Character value > 127 present in text - value(n)

   The Compiler has found a character with value n (128-255 decimal). These values are used for compress tokens so cannot be used in the text. You will get an error instead of a warning if you try to compress the text with these characters present.

Compiler ERRORS

Whenever an error is found the line number of the source file and the contents of the line are printed when appropriate then the error number and error reason. The errors that can be found are:-

0) Max texts already processed

   The maximum No. of texts allowed in this section have already been defined.

1) Valid word not found xxxx

   Either a parameter was missing or an invalid character was detected. xxxx may not be present in the message.

2) No. (1-254) not found

   A number in the range 1-254 was expected but not found.

3) Too many parameters

   Too many parameters have been specified. Possibly the ; has been left out at the beginning of a comment.

4) Vocab limit exceeded - VOCAB too big!

   There is a limit on the size of the vocabulary which has now been reached. The 'nn words processed' message which follows, shows by how many words your vocabulary is too big.

5) xxxx is not in Vocabulary

   Word xxxx has not been defined in the vocabulary.

6) Connections for all locations already processed

   Movements for all locations specified in LTX have been processed but the end of the CON section has not been reached.

7) Location No. not found

   A location No. was expected but not found.

8) Location No. too big

   The location No. specified has not been defined in the LTX section.

9) Entries for all objects already processed

   Entries for all objects specified in the OTX section have been processed but the end of the OBJ section has not been reached.

10) Start of entry expected

    The compiler expected this line to be the start of an entry.

11) System message No. not found

    A System message No. was expected but not found.

12) Percentage not found

    A percentage was expected but not found.

13) Percentage out of range

    The percentage specified is outside the range 1-99.

14) xxxx is not a condition or action

    xxxx is not a recognised condition or action.

15) Object No. not found

An object No. was expected but not found.

16) Object No. too big

The object No. specified has not been defined in the OTX section.

17) Message No. not found

A message No. was expected but not found.

18) Message No. too big

The message No. specified has not been defined in the MTX section.

19) Flag No. not found or too big

A flag No. was expected but was not found or was > 255.

20) System message No. too big

The System message No. specified has not been defined in the STX section.

21) Value not found or too big

A value was expected but was not found or was > 255.

22) Table limit exceeded

The compiler has an area of memory which it uses for 2 things. Firstly it has to store all of the vocabulary in it (7 bytes for each word). Then whatever remains is used as a work area for processing this table (4 bytes for each entry). This area of memory is now full! The 'nn entries processed successfully' message which follows, shows by how many entries this section is too big.

23) /nn expected

The next entry expected should start /nn.

24) Invalid File Name

The file name in the command line or /LNK statement contained some invalid characters.

25) Drive A-P not found

A drive identifier in the range A-P was expected but not found.

26) Object starts worn but is unwearable

If an object starts the game worn then it must be wearable.

27) Invalid null word character

An invalid character has been specified as the null word character.

28) Adventure Name not given

An adventure name could not be found in the command line.

29) Invalid drive x

An invalid drive (not A-P) was specified on the command line.

30) Parameters missing

A parameter was expected but has not been supplied.

31) Insufficient System messages

System messages 0-60 must be specified.

32) Connections for remaining locations missing

You must specify connections for every location specified in LTX.

33) Entries for remaining objects missing

You must specify a definition for every object specified in OTX.

34) Unable to open xxxx

The database file xxxx could not be opened. eg Disk directory full.

35) Failure to rename xxxx

The database file .$$$ could not be renamed to .PDB

36) Duplicate word xxxx

The word xxxx has been defined twice in VOC. Remember that only the first 5 characters are significant.

37) xxxx not found

The source file xxxx could not be found.

38) Disk full

There is insufficient disk space for the database.

39) Illegal use of Flag 38

You have specified an illegal use of Flag 38 eg SET 38, LET 38 x - where x is greater than the No. of locations.

40) A container needs a corresponding location

For an object to be a container there must be a location with the same No. as the object.

41) xxxx expected

The next section expected should start with xxxx.

42) Valid word type not found xxxx

xxxx is not a valid wordtype.

43) Object weight not found

An Object weight was expected but not found.

44) xxxx is not a(n) yyyy

A particular wordtype (yyyy) is expected here but xxxx is not that type.

45) Adjective specified without noun

To specify an adjective you must also specify a noun.

46) /PRO nnn expected

The next section expected should start with /PRO nnn.

47) Process table No. not found or too big

A Process table No. was expected but was not found or was > 255.

48) Process table No. out of range

In the PROCESS action only Process table Nos from 2 to 254 are allowed.

49) You can only PUTIN to or TAKEOUT of a container

You are trying to put an object into something which is not a container or you are trying to take an object out of something which is not a container.

50) Wearable indicator not found or invalid x

A 'Y' or the nullword character was expected but not found. x may not be present.

51) Object weight too big

Object weights must be in the range 0-63.

52) Container indicator not found or invalid x

A 'Y' or the nullword character was expected but not found. x may not be present.

53) Character value > 127 present in text - value(n)

The Compiler has found a character with value n (128-255 decimal). These values are used for compress tokens so cannot be used in the text. You cannot compress text which has these values in it.

54) Database much too big

The database is so big that it has wrapped around from 65535 to 0!

nn) Compiler error nn

There is an error within the compiler. More specifically one of the internal checks within the compiler has failed. Firstly try re-compiling your adventure. If it fails a second time, try again using your backup copy of the compiler in case your normal copy has become corrupt. If the problem persists contact GILSOFT - we will be pleased to help you.

Interpreter Initialisation Errors

The following messages may be issued during Interpreter initialisation:-

1) File name error

The filename must be unambiguous. eg PAWINT TUT* is invalid.

2) Second parameter can only be 'C'

The only valid second parameter is C. eg PAWINT TUTORIAL B is invalid.

3) Database not found

Either you have not specified a database name or the database cannot be found.

4) Database too big by approx nnn

Your database is too big for the current memory size. There are four things you might be able to do:-

a) Get more memory! or a bigger computer!
b) Squash the database by using the compress option when compiling.
c) Make the database smaller.
d) Split the database into two or more smaller adventures.

5) Database incompatible with this version

Either the database is corrupt or it was written using a compiler which is not compatible with your Interpreter.

Interpreter Runtime Errors

Although we do as much checking as we can in the Compiler there are a few errors that cannot be detected until runtime.

When a RUNTIME error occurs you will be presented with the following information provided that diagnostics are available:-

a) The error No.
b) The Process Table No.
c) The word values of the entry concerned
d) The Condition/Action No.

a) The error Nos. have the following meanings:-

1) An attempt to set flag 38 (current location) too high
2) An attempt to put an object at location 255
3) An attempt to stack PROCESS calls too deep
4) An attempt to stack DOALL's
5) An invalid Condition/Action No. (Corrupt Database?!)

b) The Process Table No. should need no further explanation.

c) The word values can be looked up in the Vocabulary. NB A word value of 255 means the nullword character.

d) The valid Condition/Action Nos have the following meanings:-

0	AT
1	NOTAT
2	ATGT
3	ATLT
4	PRESENT
5	ABSENT
6	WORN
7	NOTWORN
8	CARRIED
9	NOTCARR
10	CHANCE
11	ZERO
12	NOTZERO
13	EQ
14	GT
15	LT
16	ADJECT1
17	ADVERB
18	INVEN
19	DESC
20	QUIT
21	END
22	DONE
23	OK
24	ANYKEY
25	SAVE
26	LOAD
27	TURNS
28	SCORE
29	CLS
30	DROPALL
31	AUTOG
32	AUTOD
33	AUTOW
34	AUTOR
35	PAUSE
36	TIMEOUT
37	GOTO
38	MESSAGE
39	REMOVE
40	GET
41	DROP
42	WEAR
43	DESTROY
44	CREATE
45	SWAP
46	PLACE
47	SET
48	CLEAR
49	PLUS
50	MINUS

51	LET
52	NEWLINE
53	PRINT
54	SYSMESS
55	ISAT
56	COPYOF
57	COPYOO
58	COPYFO
59	COPYFF
60	LISTOBJ
61	EXTERN
62	RAMSAVE
63	RAMLOAD
64	BELL
68	PREP
69	NOUN2
70	ADJECT2
71	ADD
72	SUB
73	PARSE
74	LISTAT
75	PROCESS
76	SAME
77	MES
79	NOTEQ
80	NOTSAME
81	MODE
83	TIME
85	DOALL
86	PROMPT
88	ISNOTAT
89	WEIGH
90	PUTIN
91	TAKEOUT
92	NEWTEXT
93	ABILITY
94	WEIGHT
95	RANDOM
100	WHATO
101	RESET
102	PUTO
103	NOTDONE
104	AUTOP
105	AUTOT
106	MOVE

NB You must test your adventure to ensure you do not get any of these runtime errors.

The Essays

There now follows a discussion of several topics which rely on several features of PAW in combination and are thus not so easy to describe under any one heading:

The parser gives a little English lesson.
Objects will be more flexible after reading this.
Multi-part games for when memory just isn't big enough.
Light & Dark throws some light on the matter?
EXTERN especially for any internees.
PSIs add a bit of character to your games!

Enough of the frivolity there are pages to fill...

The Parser

The parser works by scanning an input line (up to 125 characters) for words which are in the vocabulary, extracting 'Phrases' which it can turn into Logical sentences.

When a phrase has been extracted, the Response and Connections tables are scanned to see if the Logical Sentence is recognised. If not then system message 8 ("I can't do that") or system message 7 ("I can't go in that direction") will be displayed depending on the Verb value (i.e. if less than 14 then system message 7 will be used) and a new text input is requested. A new text input will also be requested if an action fails in some way (e.g. an object too heavy) or if the writer forces it with a NEWTEXT action. The results might otherwise be catastrophic for the player. e.g. GET AXE AND ATTACK TROLL, if you don't have the axe you wouldn't really want to tackle the Troll!

If the LS is successfully executed then another phrase is extracted or new text requested if there is no more text in the buffer.

Phrases are separated by conjugations ("AND" & "THEN" usually) and by any punctuation.

A Pronoun ("IT" usually) can be used to refer to the Noun/Adjective used in the previous Phrase - even if this was a separate input. Nouns with word values less than 50 are Proper Nouns and will not affect the Pronoun.

The Logical Sentence format is as follows:-

(Adverb)Verb(Adjective1(Noun1))(preposition)(Adjective2(Noun2))

where bracketed types are optional. i.e. the minimum phrase is a Verb (or a Conversion Noun - which is a Noun with a word value

if no Verb is found in a phrase will be converted into a Verb e.g. NORTH). If the Verb is omitted then the LS will assume the previously used Verb is required. i.e. GET SWORD AND SHIELD will work correctly! The current 'IT' (pronoun) will become the first Noun in a list like this. Ie 'IT' would be replaced with SWORD in the example. It (if you will excuse the pun) will not change until a different Verb (or conversion Noun) is used.

Note that the phrase does not strictly have to be typed in by the player in this format. As an example:

```
GET THE SMALL SWORD QUICKLY
QUICKLY GET THE SMALL SWORD
QUICKLY THE SMALL SWORD GET
```

are all equivalent phrases producing the same LS. Although the third version is rather dubious English.

A true sentence could be:-

```
GET ALL. OPEN THE DOOR AND GO SOUTH THEN GET THE BUCKET AND
LOOK IN IT.
```

which will become five LS's:-

```
GET ALL
OPEN DOOR    (because THE is not in the vocabulary)
SOUTH        (because GO is not in the vocabulary)
GET BUCKET
LOOK BUCKET (from IT) IN (preposition)
```

Note that DOALL will not generate the object described by Noun(Adjective)2 of the Logical sentence. This provides a simple method of implementing EXCEPT. e.g. GET ALL EXCEPT THE FISH, it has the side effect of not allowing PUT ALL EXCEPT THE FISH IN THE BUCKET, as this has three nouns!

Objects

Underlines in text will be converted during gameplay into a description of the last object referenced by GET,DROP,DESTROY etc. This is mainly to deal with the fact that GET,DROP etc report their success (or failure!) but can be used usefully for examining objects and other automatic reports. Flag 53 is used to control the way objects are displayed when the LISTOBJ and LISTAT actions are used. If the flag is set to 64 (i.e. Bit 6 is set) then objects will be listed without newlines between them, forming a valid English sentence - compound listing.

The formats are as follows:

```
SM53 ("nothing.")  - can only occur with LISTAT
object SM48(".")
object SM47(" and ") object SM48(".")
object SM46(", ") object SM47(" and ") object SM48(".")
```

In addition, Bit 7 of flag 53 will be set (i.e. flag will be greater than 127) if any objects were printed. This allows you to determine whether or not a NEWLINE is required.

A LISTAT action will usually be preceded by a message.

The description of **object** is constructed from the full description given in the object text section. The preferred format for an object description is:

indefinite.article (adjective) noun . extra text

where; the indefinite article is "A" or "An" or "Some". The Adjective and the Noun should have a lower case letter e.g. 'A small key', 'Some sand' or 'An orange. Rather mouldy'. PAW extracts a description of the object in two ways:

1/ For GET, DROP etc (i.e. "_") the indefinite article is skipped and the description printed up to (but not including) the first full stop. e.g. "I now have the **small key.**".

2/ For a compound list of objects the indefinite article is forced to start with a lower case letter and the description printed up to (but not including) the first full stop. e.g. "In the bag is **a small key.**".

Obviously if you don't use underline or compound listings, then you are free to describe objects any way you like.

Important: If an object is to be a container; there must be an unused location with the same number for PAW to use as the 'inside'! i.e. Object objno. 1 would need Location locno. 1 - Not forgetting to mark it as a container in the object definition section.

Creating Multi-Part Adventures

In order to create a larger (and thus more interesting) play area in an adventure, without sacrificing the quality of the description, you can split the game into smaller sections. It is best to do this with a game that lends itself to having several areas, with only one join between each, this is called a bottleneck. e.g. a game where setting sail on a boat is the final task in the first part.

To allow the score, turns taken and other information to be carried forward into the next game you must use the LOAD/SAVE game position actions. In order to load a game position into a different game to that which it was saved from, you need the same number of locations and objects in each part. In addition, all objects which may possibly be carried forward by the player, must have the same description in all parts.

Let's take a game with 120 locations, that is to be split in half, thus requiring 60 locations in each part. Actually location 60 will exist in both games as the transition location (where the player starts and finishes) and a spare flag (say 26) will be used to indicate which part of the game a position is from. So when the player completes part 1 they are moved to location 60 and flag 26 is set to 1 to show it.

The setup for part 1 would be:

Location 60
End of Part 1 - Prepare a disc to save your position.
(You may save more than one copy if you like).
Please LOAD part 2 and follow the onscreen prompts.

Process 1

```
_    _      AT      60        ;End of game?
            LET     26   1    ;Valid position from part 1.
            SAVE
```

And in part 2:

Location 0
Part 2 - Prepare to load disc with saved position.

Location 60
Any introduction wanted for Part 2.

Process 1

```
_    _      AT      0         ;Just starting?
            LOAD              ;Will then be at another location.
```

```
_    _    NOTEQ   26   1     ;Not a valid position from part 1.
          GOTO    0          ;So request another load.
          DESC

_    _    AT      60         ;Just loaded a valid position
          ANYKEY             ;Wait until introduction read
          RESET   1          ;Start game properly at location 1
```

The RESET action does a DESC of the new start location automatically, after setting all objects that aren't carried, worn or at location 60 to their starting position. Note that you should insert any CLEAR actions for flags between the ANYKEY and the RESET as the flags are not affected by the RESET.

Light and Dark

Darkness is becoming something of a cliche in adventures these days, but used correctly it can add to the sense of realism considerably.

Within PAW, darkness is created by setting flag 0 to a value other than 0. This must be done whenever the player moves into and out of darkness. i.e. the move must be done with a GOTO in the Response table, to allow the SET or CLEAR action to occur.

If the player is being provided with a source of light then object 0 is the easiest way of implementing it. A source of light does not have to be a torch or candle, with a little imagination it can be infra-red glasses or a wide beam laser!

Take for example the creation of a night and day cycle, over 24 time frames which we will assume are equivalent to 1 hour.

The entries required in Process 2 are:

```
_    _    EQ      5   0      ;End of cycle
          LET     5   24     ;Start the counter again

_    _    EQ      5   18     ;Nightfall
          SET     0
          MESSAGE x

_    _    EQ      5   6      ;Daybreak
          CLEAR   0
          MESSAGE y
```

Importantly if part of the game is underground, or inside a building, don't forget to determine if the player can actually see nightfall and daybreak from where they are, before printing the messages.

EXTERN

The EXTERN command can be used to call your own machine language program. This feature can be utilized only in a final game as the Interpreter needs to be patched to point at your bit of machine code. See file PAWINST.TXT for more information on the EXTERN action and how your piece of machine code must interface with the Interpreter.

Pseudo-Intelligences

The main thing to remember is that a character (or PSI) is a word in the Vocabulary (usually a Noun with a value less than 50 so as to be a Proper Noun). Some flags, a series of messages and some entries in one or more Process tables. One flag shows where they are, the messages provide information about their actions and the process table entries tie it all together.

So imagine a character called Sanec who can walk around independently. He is described in the vocabulary as SANEC (word value 25, Noun). Flag number 20 is used to give his location. Process table 3 will deal with speech to him. While Process table 4 will deal with his movements and actions. The following entries allow him to move around when you ask him too. After a short time he will get 'bored' and vanish in a puff of smoke!

Message 1
Sanec did not seem to understand what you said.

Message 2
No one of that name here!

Message 3
Sanec replies "hello" in a gruff voice.

Message 4
Sanec wanders that way as he has nothing better to do.

Message 5
Sanec the wizard is here.

Message 6
Sanec 'politely' ignores what you say.

Message 7
Sanec turns to face you and in his gruff voice announces;
"I'm bored with all this, I'm off to a bigger game"
and promptly vanishes in a puff of green smoke!

First; Sanecs' presence at a location must be announced. So in Process 1 (which is called after every describe of a location) we check if he is here i.e. flag 20 (his location) is the same as flag 38 (our location). Note that we ensure we are not at location 0 as this is always an introduction screen.

```
_    _      SAME      20  38  ;Ensure Sanec is here
            NOTAT      0      ;Player is not in location 0.
            MESSAGE    5      ;Say Sanec is here.
```

To deal with speech to Sanec, we need two entries in Response as follows:-

```
SAY   SANEC PREP      TO        ;this could be omitted to allow
;                                short Verb Noun sentences to be
;                                understood
            SAME      20  38    ;Make sure Sanec is here
            PROCESS    3        ;Deal with any speech
            DONE                ;Prevent drop through with new LS.

SAY   _     PREP      TO        ;again optional
            MESSAGE    2        ;no one of that name here!
            DONE
```

The following entries in Process 3:-

```
_     _     PARSE               ;This entry always carried out to
;                                convert the input string to a LS.
            MESSAGE    1        ;PARSE comes here if it fails to
;                                find a valid phrase
            DONE                ;Note that the LS is corrupt and no
;                                further table entries must be
;                                executed

HELLO _     MESSAGE    3        ;Assuming HELLO is a verb in vocab
            DONE                ;So that SAY TO SANEC "HELLO" works

_     _     LT        33  14    ;A movement word said to Sanec?
            MOVE      20        ;See if a connection for that way
            MESSAGE    4        ;Come here and tell player if so
            DONE

_     _     MESSAGE    6        ;He ignores you (i.e. nothing else)
```

Obviously many more entries would be required to give Sanec an appearance of understanding speech, but with a few clever entries he can give a wide variety of responses.

Finally; to give Sanec a chance of disappearing when bored, we need an entry in Process 4 of:

```
_     _     EQ        20   2    ;SANEC at location two?
            CHANCE    10        ;10% chance
            SET       20        ;Location 255 does not exist
            AT         2        ;are we where he was?
            MESSAGE    7        ;POOFF! - tell player he disappeared
```

And an entry in Process 2 to call table 4 regularly:

```
_     _     PROCESS 4           ;Process SANEC
```

In this way a very convincing character can be built up. They add a great deal to the sense of realism in games. Especially if interaction with them is required as part of the solution.

The system messages

SM0 - is used instead of the location description when it is dark.
SM1 - is printed by LISTOBJ if at least one object is present.
SM2 to SM5 - are the four input prompts which are selected randomly unless flag 42 is set to be a valid message number.
SM6 - is produced by the parser when no further phrase can be understood.
SM7 - is produced if no action was carried out (or NOTDONE was) in Response when the Verb is < 14
SM8 - is produced if no action was carried out (or NOTDONE was) in Response when the Verb is > 13
SM9 to SM11 - are printed by action INVEN.
SM12 - printed by QUIT
SM13 and 14 - are printed by the END action.
SM15 - the OK action message.
SM16 - the ANYKEY action message.
SM17 to SM20 - are the TURNS action messages.
SM21 and SM22 - are the SCORE action messages.
SM23 to SM29 - are the first of many messages produced by the object manipulating actions.
SM30 - the positive response expected by END and QUIT.
SM31 - the negative response expected by END and QUIT.
SM32 - produced when a screen full of text has appeared.
SM33 - the input marker.
SM34 - spare. (Cursor on Spectrum PAW)
SM35 - displayed when a timeout occurs
SM36 to SM45 - are more messages produced by the object manipulating actions.
SM46 - the link between objects when listing continuously
SM47 - the final link between the last two objects when listing
SM48 - the termination of a list of objects (printed by both LISTOBJ and LISTAT, so take care.)
SM49 and SM50 - yet more object messages
SM51 - the termination for a compound sentence on PUTIN/TAKEOUT (and AUTOP/AUTOT)
SM52 - a final object message.
SM53 - message for LISTAT action if no objects found.
SM54 to SM60 are messages used by the SAVE and LOAD actions.

SM61 onwards are free to be used for your own use. PAW on other machines may use more messages, so bear this in mind if you intend transferring the adventure to another version.

Summary

Function of P.A.W. Flags

The normal flags are free for use in any way in games. The auto decrement flags (2 to 10) are also free for use, but be sure you know in which situations they are reduced before using them. Other flags should mostly only be set using the appropriate action, but useful tests can be carried out on their contents.

Flag 0 When non zero indicates game is dark (see also object 0)
Flag 1 Holds quantity of objects player is carrying (but not wearing)

The following 9 flags are decremented if non zero by PAW;
Flag 2 When a location is described
Flag 3 When a location is described and it's dark (Flag 0 not 0)
Flag 4 When a location is described, it's dark and object 0 is absent
Flags 5 to 8 Every time frame (i.e. every phrase/timeout)
Flag 9 Every time frame that it's dark
Flag 10 Every time frame that it's dark and object 0 is absent

Flags 11 to 28 are free for use in your own games

Flag 29 holds the Picture Control flags in the Spectrum version of PAW.

Flag 30 Score flag
Flag 31/32 (LSB/MSB) holds number of turns player has taken (actually this is the number of phrases extracted from the players input).

Flag 33 holds the Verb for the current logical sentence
Flag 34 holds the first Noun in the current logical sentence
Flag 35 holds the Adjective for first Noun
Flag 36 holds the Adverb for the current logical sentence

Flag 37 holds maximum number of objects conveyable (initially 4)
Set using ABILITY action.
Flag 38 holds current location of player

Flag 39 holds current top line of screen in the Spectrum version of PAW.

Flag 40 holds screen mode (range 0 to 3) set with MODE action.
Bit 1. suppresses the More... message
Bit 0. stops the screen being cleared before a DESC
Flag 41 holds line number for split in the Spectrum version of PAW.

Flag 42 holds prompt to use (a system message number - 0 selects one of four randomly
Set by the PROMPT action.

Flag 43 holds the Preposition in the current logical sentence
Flag 44 holds the second Noun in the current logical sentence
Flag 45 holds the Adjective for the second Noun
Flag 46 holds the current pronoun ("IT" usually) Noun
Flag 47 holds the current pronoun ("IT" usually) Adjective

Flag 48 holds Timeout duration required
Flag 49 holds Timeout Control flags
Bit 7 - Set if timeout occurred last frame
Bit 6 - Set if data available for recall (not of use to writer)
Bit 5 - Set this to cause auto recall of input buffer on timeout
Bit 2 - Set this so timeout can occur on ANYKEY
Bit 1 - Set this so timeout can occur on "More..."
Bit 0 - Set this so timeout can occur at start of input only
Set using TIME (as is flag 48), TIMEOUT tests Bit 7 of this flag.

Flag 50 holds Objno. for DOALL loop. i.e. value following DOALL

Flag 51 holds last object referenced by GET/DROP/WEAR/WHATO etc. This is the number of the currently referenced object as printed in place of any underlines in text.

Flag 52 holds players strength (maximum weight of objects carried and worn - initially 10)
Set with ABILITY action.

Flag 53 holds object print flags
Bit 7 - Set if any object printed as part of LISTOBJ or LISTAT
Bit 6 - Set this to cause continuous object listing i.e. LET 53 64 will make PAW list objects on the same line forming a valid sentence.

Flag 54 holds the present location of the currently referenced object
Flag 55 holds the weight of the currently referenced object
Flag 56 is 128 if the currently referenced object is a container.
Flag 57 is 128 if the currently referenced object is wearable

Flags 58 & 59 should be avoided as they will be used for any expansion

Flag 60 to 255 are available for your own use.

Summary

The CondActs

Conditions:

```
AT          locno              ;Ensure player at specific location
NOTAT       locno
ATGT        locno              ;higher location than specified
ATLT        locno              ;lower...

PRESENT     objno
ABSENT      objno
WORN        objno
NOTWORN     objno
CARRIED     objno
NOTCARR     objno
ISAT        objno    locno+
ISNOTAT     objno    locno+

ZERO        flagno
NOTZERO     flagno
EQ          flagno   0-255
NOTEQ       flagno   0-255
GT          flagno   0-255
LT          flagno   0-255
SAME        flagno   flagno
NOTSAME     flagno   flagno

ADJECT1     word
ADVERB      word
PREP        word
NOUN2       word
ADJECT2     word

CHANCE      0-99             ;Random possibility of success
TIMEOUT                      ;Players last input timed out

QUIT
```

Actions (Those marked § are type 4, † are type 3, ¶ are type 1)

```
GET       §  objno
DROP      §  objno
WEAR      §  objno
REMOVE    §  objno
CREATE       objno
DESTROY      objno
SWAP         objno    objno
PLACE        objno    locno+
PUTO         locno+
PUTIN     §  objno    locno
TAKEOUT   §  objno    locno
DROPALL
AUTOG     §
```

```
AUTOD   §
AUTOW   §
AUTOR   §
AUTOP   §  locno
AUTOT   §  locno
COPYOO     objno    objno

COPYOF     objno    flagno    ;Copy position of object to flag
COPYFO     flagno   objno
WHATO                         ;Convert Noun1(Adjective1) to
                               current object
WEIGH      objno    flagno    ;Weight of object is put in flag

SET        flagno
CLEAR      flagno
PLUS       flagno   0-255     ;Add value to flag
MINUS      flagno   0-255
LET        flagno   0-255
ADD        flagno1  flagno2   ;contents of flag1 added to flag2
SUB        flagno1  flagno2
COPYFF     flagno1  flagno2
RANDOM     flagno             ;Set to random number from 0 to 99
MOVE       flagno             ;Adjust contents of flag according
                               to the LS Verb and the Connection
                               section entry for location, that
                               the contents specify. (allows
                               movement in PSIs)

GOTO       locno
WEIGHT     flagno             ;Weight of objects carried & worn
                               are put in flag
ABILITY    0-255    0-255     ;Set conveyable objects and strength

MODE       0-255
PROMPT     sysno              ;Prompt on input. 0 is random
TIME       0-255    0-255

PRINT      flagno             ;display contents of flag on screen
TURNS
SCORE
CLS
NEWLINE
MES        mesno              ;message without a newline
MESSAGE    mesno              ;message with a newline
SYSMESS    sysno              ;system message without newline

LISTOBJ                       ;List objects at current location
LISTAT     locno+             ;List objects at specified location
INVEN   †

DESC    ¶

END                           ;Type 2,Exits table to restart game
```

Summary

```
DONE     †
NOTDONE  †
OK       †

SAVE     ¶
LOAD     ¶
RAMSAVE
RAMLOAD      flagno

ANYKEY
PAUSE        0-255              ;Delay program for n/50 of a second

PARSE                           ;Convert input string to valid LS
NEWTEXT                         ;Force the loss of remaining phrases

BELL

PROCESS      procno             ;Execute sub-process

DOALL        locno+             ;Generate Noun(Adjective)1 for each
                                 object at Location locno.

RESET    ¶   locno              ;Move player and present objects,
                                 reset others to start position -
                                 used to chain games with LOAD
EXTERN       0-255              ;Call external program
```

Where:
locno. is a valid location number defined in LTX.
locno+ also allows the use of;-

" "	or 252 (not-created),
"WORN"	or 253 (worn),
"CARRIED"	or 254 (carried) and
"HERE"	or 255 (current location of player)

mesno. is a valid message number defined in MTX.
sysno. is a valid system message number defined in STX.
flagno. is any flag (0 to 255).
procno. is a valid sub-process number.
word; is a word of the required type, which is present in the vocabulary, or "_" which ensures no-word - not an anymatch as normal).

Published by Gilsoft International Ltd.,
2 Park Crescent, Barry, South Glamogan CF6 8HD
Telephone Barry (0446) 732765