A Technical guide to:

DAAD Adventure Writer - Version 2 Release 1

A multi-machine adventure writing system.

Revised in November '91.

Contents

Section 1 - Overview

The basic principle behind DAAD is to provide a system where a game needs to be written only once and will then work on all machines. This of course is not truly possible without sacrificing facilities. So a compromise has been arrived at where an amount of extra work may need to be done to allow each machine to be used to best advantage. The core of the game design remains independent reducing development and debugging time considerably.

DAAD is a programming language designed especially for writing Adventure Games. It is loosly based on the QUILL and PAW systems of Gilsoft, and a familiarity with their operation is useful. The main differences from these utilities are as follows:

* DAAD Uses the a compiler system similar to the CPC PAW.

* The DAAD database can be compiled for a variety of computers. Thus the same game can be run on all supported computers with little or no change.

* The DAAD Process language is far more advanced with many features of traditional languages such as looping and indirection, although it lacks several of the machine specific commands for PAW.

* It costs a lot of dough to buy a copy!

There are several terms used in this text which should be defined:

Development Machine: The computer you write the game on. There are development versions of DAAD for IBM and Atari ST (although only the IBM has a full set of utilities at the moment).

Object machine: The computer the game will finally run on. DAAD supports the following machines:
IBM (with support for MDA,CGA and EGA)
Sinclair Spectrum 48k/128k
Amstrad CPC range
Commodore 64 (and 128 in 64 mode)
MSX computers with at least 64K of RAM
Atari ST range
Commodore Amiga range
Amstrad PCW 8000/9000 Series

DAAD can be divided into five main functional areas thus:

1/ The Source
The source (.SCE files) are a collection of tables, which contain all the information to define an adventure game. They are in the format of an ASCII Text file.

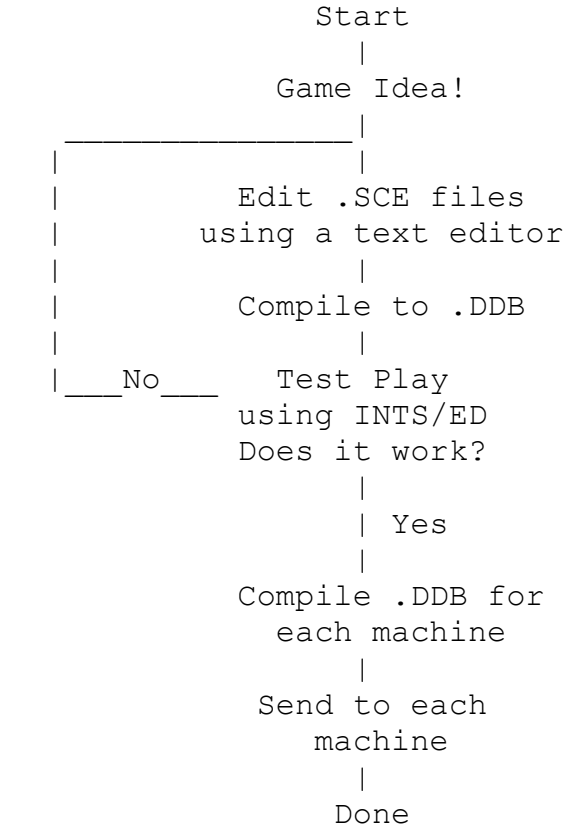
2/ The Compiler
This program is provided for the development machine only.
The Compiler checks the source file for errors and converts it into a DAAD database for the specified machine (.DDB file) which the Interpreters can understand.

3/ The Interpreters
These are the real heart of DAAD. There is an interpreter for each computer supported on the DAAD system. It runs the data in the database using its collection of routines to carry out the tasks needed by adventure games. It provides the machine independent aspect of DAAD.

4/ The Graphics Editors
There are two types of graphic editors:
  a) Drawstring
     These use a form of drawing known as PLOT,LINE and FILL. Which allow very complex graphics to be drawn using very little memory. This is used for 8 bit computers. Note that there are graphics editors for each machine except MSX, as MSX can use the Spectrum graphics almost unchanged.
  b) Image
     These use graphics drawn with an art package (possible digitised), providing the necessary functions to assemble pictures into a graphics database for the Interpreters. This system is used on 16 bit computers. These can be produced with any art package required, but DEGAS PI1 on ST is the standard file format accepted by the programs.

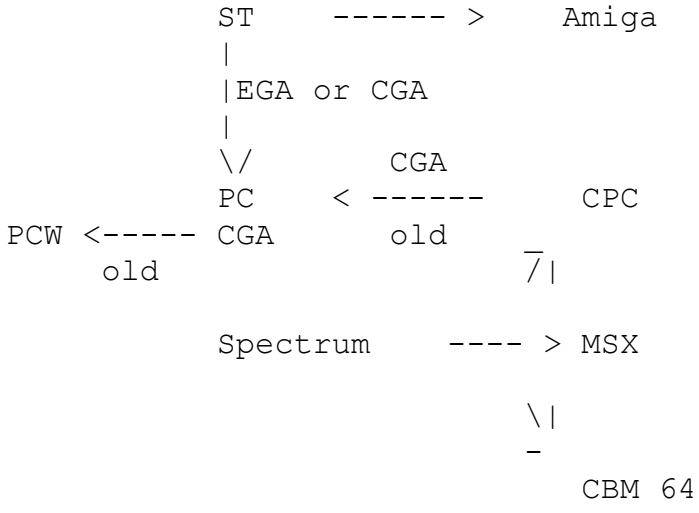
5/ The Utilities & Communications
These are a collection of programs provided, usually on the development machine and where needed on the target, to allow the conversion and sending of data, between machines

The main cycle of developing a text game with DAAD can be visualised as follows:



The graphics follow a slightly different process. They are split into the two groups as mentioned above:

The routes for graphics, following the arrows from source machine to object, are normally:-



The two halves of a graphics database and a text database are combined on the object machine. Although in the case of the 16Bit machines where the disk is used to hold the graphics the files remain seperate.

Section 2 - Disk Contents

The disks supplied are as follows and of course you should make copies and store the originals in a safe place. You will also find it helpful to copy some files to working disks (or install the entire system on a hard disk) for each game, this is discussed at the end of this section.

PC (360K, 5.25 inch)

Disk 1 - Developer

This contains the major files to allow development work to be carried out on the PC.

DC	EXE	The Compiler
DMG	EXE	PC EGA/CGA graphics manager - note that this has now been superceeded by the ST DMG. This remains to allow PCW support. It will be dropped in later releases.
OBJ	(dir)	Contains the object files to make the various types of DAAD interpreter.
BLANK	SCE	The standard empty database
BLANK	DDB	And a compiled version
SYMBOLS	SCE	A #include file to define common system symbols
SYMBOLS	ASM	A file to define symbols for Assembler use
SYMBOLS	Z80	Extra symbols specific to the Z80
SYMBOLS	652	And those for the 6502 processor
MACROS	Z80	Definition of MACROS for an assembler EXTERN
EXTERN	ASM	A MASM format source file, the mimimum 8086 ext.
CHARS	ASM	A MASM file containing NOOFPARTS, last in link

Disk 2 - Utilities

DAAD is supplied with a large range of utilities to carry out a range of tasks. The sub directories on this disk reflect that purpose.

GRAPHICS	(dir)	Utilities related to graphics work.
DRAW	(dir)	Those that relate to 'drawstring' machines.
DSC	COM	Spectrum -> CBM graphics converter
DSM	COM	Spectrum -> MSX graphics converter
DSA	COM	Spectrum -> CPC graphics converter
DCA	COM	Compress Amstrad Graphics (for old databases)
DCS	COM	Compress Spectum Graphics (old databases)
n.b. DCA & DSA are now incorporated in the DG programs. They are included to allow any pictures or databases drawn with old versions to be compressed.		
PIXELS	(dir)	For the image machines (PC/ST/AMIGA/PCW)
CACGA	COM	Convert CPC Mode 2 to CGA screen image
CSTEGA	EXE	Convert/Display ST .PI1 to/on EGA mode 13
CSTVGA	EXE	Convert/Display ST .PI1 to/on VGA mode 13
CSTA	COM	Convert ST DEGAS (PI1) to CPC Mode 2

```
 CSTAM   EXE    Convert ST DEGAS (PI1) to Amiga bitmap
 CSTS    EXE    Convert ST DEGAS (PI1) to Spectrum
 CST3PCW EXE    Convert ST DEGAS HiRes (PI3) to PCW
 CST1PCW EXE    Convert ST DEGAS LoRes (PI1) to PCW
 n.b. A CSTCGA is in preparation, but CSTA and CACGA together
 can be used to convert an ST .PI1 to CGA screen!

COMMS    (dir)  Programs to send and receive files.
 XMP     EXE    XModem+ comms using RTS/CTS (Spectrum mainly)
 GOCBM   COM    Send files to CBM (needs special receiver)
 PPM     COM    Send and play games using PDS on MSX
 AAH     COM    Add Amstrad Header to a file
 ASH     COM    Add Spectrum +3 Header to a file
 RAAH    COM    Remove Amstrad Header from a file
 RASH    COM    Remove Spectrum +3 Header from a file
For communication with Atari, Amiga and CPC you use commercial or
PD comms packages

SQUEEZE  (dir)  Compression related programs
 FINDTOK EXE    Analise a FINDTOK file for common tokens

RECOVER  (dir)  Help when you trash your source file
 LOOK   COM     Allow text part of a .DDB file to be examined.

The remaining directories contain utilities which have no  direct
use but may prove useful sometime.

PRINT    (dir)  Dealing with the printer
 SPP     COM    Sets the printer to IBM mode by software
WP       (dir)  For wordprocessing use
 STRIP   COM    Removes any spaces from the end of a line
 ASC1ST  COM    Converts the major part of 1st word text to ASCII
ASM      (dir)
 MHEX    COM    Make a HEX file from data
 GENTASM COM    Convert HISOFT GENS format to 2500AD.
```

SPECTRUM (DISCiPLE 800K, 3.5 inch/Spectrum +3 3")

```
XM      LOAD d1"xm" will start the XModem+ comms program (not+3)
DG      LOAD d1"dg" will start the Spectrum graphics editor
DS48GE  the graphics editor
DRE     LOAD d1"dre" allows an English game to be run
DRS     LOAD d1"drs" allows a Spanish game to be run.
DS48IS  Spanish interpreter
DS48IE  English interpreter
n.b.  No special provision is made for the +3 at the moment.  The
system is provided on both formats for mastering purposes.
```

CBM 64 (160K, 5.25 inch)

```
DG      LOAD "dg",8,1 starts the CBM graphics editor.
DR      LOAD "dr",8,1 will start the parallel recieve prog
SDI     The Spanish interpreter
```

EDI   The English interpreter
LS1   are the loaders to load a Spanish game called PART1/2
LS2

AMSTRAD CPC 464/6128 (160K, 3 inch)

Side 1 - Use under CPM (V3 on 6128 is recommended, although 2.2 should work)

DCPCIS Z80   The Spanish interpreter
DCPCIE Z80   The English interpreter
MCRF   COM   Make an AMSDOS 'RUN' program
UKM7AMS COM   A public domain MODEM7 installed for CPC RS232
P11CPM3 EMS   Patched version of CPM3 to allow Modem7.

Side 2 - For AMSDOS

DG   BIN   The CPC graphics editor. (RUN "dg from AMSDOS)
BLANK   BIN   empty graphics file for above

ST (720K, 3.5 inch)

SDIn   PRG   The Spanish interpreter (filename of PART1-n)
EDIn   PRG   English version of the above
Copies are also supplied with 'L' after the no of parts. These only allow the game to be run in LoRes.
DMG   PRG   Graphics file manager
DMG   RSC   and its resource file

AMIGA (800K, 3.5 inch)

SDIn   The Spanish interpreter (filename fixed at PARTn)
EDIn   English version of the above

VT100   A public domain comms program
There is no graphics editor as the Amiga merely uses the ST file.

PCW (160K, 3")

DPCWIS Z80   The Spanish interpreter
DPCWIE Z80   The English interpreter

Other Programs/Hardware

In addition to the programs supplied you will need or are advised to obtain some of the following programs which will make life easier.

QEDIT   PC   - Shareware Text editor (very good)

PROCOMM   PC   - Shareware comms program
3.5" DRIVE PC   - Simple transfer to ST (recommended)

VT100 AMIGA - A comms program (included on DAAD disc as PD)
DOS2DOS AMIGA - Load and Save ST/IBM files on Amiga (better)

FLASH ST - Comms program (any one will do actually)
5.25 DRIVE ST - for transfer to PC (3.5" for PC is better)

FASTHACK CBM64 - High speed disk and file copier

In fact a 720K external 3.5" for the PC will make transfer simpler than a comms program to both ST and AMIGA (if you have DOS2DOS)! You will find that an ST will happily read AND write disks formatted to 720K on a PC. The reverse is not true, so you should make a habit of using PC format disks if you need to transfer the data on them between the machines.

Development Installation on a PC Hard Disc

The simplest method of using the DAAD package is to create a directory \DAAD and copy the contents of Discs 1 & 2 into it. You may find it useful to create and OBJ directory for the .OBJ files.

LINK the development interpreter for the graphics method you are going to use (see PC section of the interpreter details)

Add the directory to your path (PATH command of MSDOS).

Then create a directory for each game into which you place your source files.

Section 3 - A Worked Example

Here is a summary of the major points needed to produce a game (using the default of the Spanish language).

1/ To start editing your source file (called TUTOR) assuming you have QEDIT installed for your machine:

```
QE TUTOR.SCE
```

Now we have a blank file to start work on. To make things easier there is a file on disc which contains the minumum data needed in a file plus some useful words in the vocab etc. This is called BLANK.SCE, this can now be read in to become part of our file (CTRL K & R on QEDIT). Once you have typed in your game, you will need to save it and exit (CTRL K & X) to the command line to test it out.

2/ Now we need to make a file the computer can understand. Here we use a compiler to make a copy of your SCE file in the form of a DDB file.

```
DC TUTOR
```

Will do the job and create a TUTOR.DDB file.

3/ To actually play this you use an interpreter which reads the information in the DDB file to actually make the game

```
INTSD TUTOR
or      INTSDM TUTOR
```

Will allow you to play the game. If you find a problem then go back to edit the file as at step 1, but you will not need to load the BLANK file again!

4/ At this point you can start drawing your graphics. You will need to draw one set on the Spectrum which will later be converted to the MSX,CBM and CPC. The other set are drawn on the ST (well anywhere actually as long as they are finally saved from DEGAS in uncompressed 16 colour mode - .PI1 extension).

5/ Making the running copy for each machine:

ST

Compile a version of your source without debug information and compressed called PART1.DDB using compiler option -m5. Transfer it to the ST - Either using comms programs (such as FLASH/PROCOMM), or an external drive.

Draw you graphics and save in PI1 format. Use DMG to make a graphics file suitable for the game, called PART1.DAT.

Put the .DDB and .DAT files on the same disk along with a 6x8 character set file - with extension .CH0. See the section on character sets for details of redefining the set. For text mode you need a special 8x8 character set. A duplicate of the CH0 file called .CHR will work if you don't mind the letters spaced out a bit. Then put SDI.PRG on the disk and rename it as required.

A standard DEGAS PI1 file renamed to PART1.SCR can be included on the disk as a backdrop during loading if required.

The game is started by clicking on the .PRG file as normal on an ST.

AMIGA

Make a copy of the Minmum OS disk. Put all files on this disk.

Compile a version of your source without debug information and compressed called PART1.DDB using option -m6. Transfer it to the AMIGA (using VT100/PROCOMM or DOS2DOS from a PC disc in the external drive). Note that if there are no conditional differences between ST and AMIGA you can use the ST .DDB file as it will work perfectly.

Transfer the .DAT and .CH0 files from the ST. Rename the .CH0 file to .CHR.

Use the DSTAM utility on ST to load the .PI1 backdrop picture and save a BIT file. Transfer it to the Amiga. Rename it to be PART1.SCR

Copy SDI onto the Working disk and call it AD. Or if you change the name in the "s/startup-sequence" file you can call it anything you like. You should also make an Icon (tool type) with the same name and copy it onto the disk.

The disk is now autoboot, or if you added an icon, clickable from workbench.

PC

Compile a version of your source without debug information and compressed called PART1.DDB

EGA Using old system DMG:-

The graphics under DEGAS on the ST must be amended to use the EGA 16 colour fixed pallette.

Send them to the PC using a comms package, or copy from external 3.5" drive.

Use DMG on the PC to make a file called PART1.EGA consisting of these pictures.

Make an INTS.EXE file for the correct number of parts and place it (renamed as you like) along with the .DDB file, the .EGA file and a .CHR file on the same disk.

CGA Old system:-

The graphics can come from either ST or CPC. Use DMG to make a file called PART1.CGA.

Just putting the .CGA file on the EGA disk will automatically select the correct file according to fitted graphic card. But if you wish to make a special CGA version, follow the above steps but use the INTSCGA.EXE file as the interpreter.

Using new system multi-machine graphics (ST version of DMG):-

You should have EGA and CGA versions of the palette loaded for each picture in DMG on the ST. Any buffer flags set and a buffer sort carried out. It doesn't matter if you have compressed yet but conversion will be faster if you have.

Use the Make IBM option to write a PARTx.IBM file to a disk. Send it to the IBM and rename it to be PARTx.DAT

Make an INTSM.EXE file for the correct number of parts and place it (renamed as you like) along with the .DDB and .CHR files on the same disk.

Send a DEGAS PI1 file with a full colour palette to PC - this will normally be the ST loading screen. Use CSTVGA to make a VGA file. It will have an extension .V13 so rename it to be .VGS for a loading screen.

For both systems:-

Send a DEGAS PI1 file using only the EGA 16 colour palette to PC. Use CSTEGA to make an EGA file. It will have an extension .E13 so rename it to be .EGS for a loading screen.

The CGA loading screen is made using the CACGA program and is given an extension of CGS. If you want it to come from ST use the CSTA program to convert to Amstrad first! There will hopefully be a CSTCGA program in the future.

CPC

Compile a version of your source without debug information and compressed called xxx.DDB (etc) using option -m3c. Send it to the Amstrad using PROCOMM and UKM7AMS.

Send a copy of the Spectrum graphics file to IBM using XM on Spectrum and XMP on the IBM - calling it yyy.SDG. Or PROCOMM/UKM7AMS from CPC having saved the file on a CPC DATA format disc from the +3. Don't forget to remove the +3 header.

Use DSA to make a file for Amstrad (which will be called yyy.BIN). Send it to the Amstrad using PROCOMM and UKM7AMS.

Use the DG editor to clean up the pictures - you will normally have to use the -fs option on DSA to delete fills and shades and then re-insert them manually in DG. Note that the character set is actually 8x8 on CPC and you may like to design a special one for it which can then be sent to the CBM and the ST (for text mode).

From CPM use MCRF to make a final run file for AMSDOS. This is described in the section on this program.

Spectrum

Compile a version of your source without debug information and compressed called xxx.DDB (etc) using option -m1.

DISCiPLE:-

Send the database to the Spectrum using XMP on the PC and XM on the Spectrum.

As long as your graphics and database have the same name (the graphics will have a + sign in character position 10), you can use LOAD d1"drs" to play the game.

+3:-

Use ASH to add a Spectrum header to the .DDB file, rename it to PARTx.DDB and send it to the CPC using Procomm & UKM7AMS. The +3 will accept a CPC Data format disc quite happily!

Save the graphics file from DG onto the disc, and a copy of the DS48Il interpreter. A small BASIC loader is provided on the release disc called "DRS" ("DRE" for English), but you may like to load all files into memory and save a block.

CBM

Compile a version of your source without debug information and compressed called PART1.DDB (etc) using option -m2c.

Send the database to the CBM using DR on the CBM and GOCBM on the PC. You must give the load address as 0x3880.

Send a copy of the Spectrum graphics file to IBM using XM on Spectrum and XMP on the IBM (calling it PARTx.SDG). Or see note

about +3 under CPC.

Use DSC to make a file for CBM (which will be called yyy.CDG). Send the file to the CBM using DR on the CBM and GOCBM on the PC. You must give the load address as that given as the lowest address by DSC.

Use the DG editor to clean up the pictures - you should need to do very little here if you have been careful in drawing the pictures on the Spectrum. The character set is actually 8x8 on CBM and you may like to design a special one for it (and CPC/ST Text mode).

Copy the LS1 file to your work disk (FAST HACK EM) and rename it (using OPEN 15,8,15,"R0:newname=LS1").

MSX

Compile a version of your source without debug information and compressed called PARTx.DDB (etc) using option -m4.

Send a copy of the Spectrum graphics file to IBM using XM on Spectrum and XMP on the IBM (calling it PARTx.SDG).

Use DSM to make a file for MSX (which will be called PARTx.MDG).

Ensure a copy of the relevent interpreter is in the directory renamed to be DMSXI.Z80.

Load a copy of PDS56000 on your MSX machine (HitBit only at the moment) Use PPL PARTx to send and start the game.

N.B. the PC disk is your master, as there is no simple way to save files to cassette on MSX.

PCW

Compile a version of your source without debug information and compressed using option -m7c called PARTExxx.DDB where XXX is the part number (001,002 etc lwading zeros are required). Send this using PROCOMM/UKM7AMS to a CPC DATA format disk.

You have to make a CGA graphics file using DMG on the PC. Then use the write PCW (W) option to make a file DMG.PCW. Rename this file to PARTExxx.DAT and send to the disc. We have to change DMG Atari to provide the write facility...

Send a CHR (8x8 characters) in Amstrad format to the PCW called PARTExxx.CHR. The .CHR set used for the ST will serve for this purpose.

The loading screen is made using CST1PCW, and is called PARTE001.SCR (always) this comes from a 4 colour CGA ST screen.

(n.b. This will not change when we switch to new style graphics).

Finally place a copy of DPCWIS.Z80 on the disk renamed as you like.

Section 4 - The Interpreter

4.0 Overview

The following description of the interpreter applies to all versions. Machine specific things are described under the section for that machine.

Initialise

The system initialise is carried out only once. All flags and object positions are cleared. The screen is cleared (although this may change on 8 bit to provide loading screen maintenance). Note that clearing the flags has the effect that the game always starts at location zero. This is because Player, the current location of the player, is now zero.

Main Loop

The program makes a call to process 0. On exit from that table a return is made to the operating system or to restart the game - with only a partial initialise.

On machines which feature the autoload facility a new database part may be specified as part of the EXIT action. This will cause that part to be loaded and the main loop to be restarted without any initialisation taking place allowing the game state to be passed onto another database.

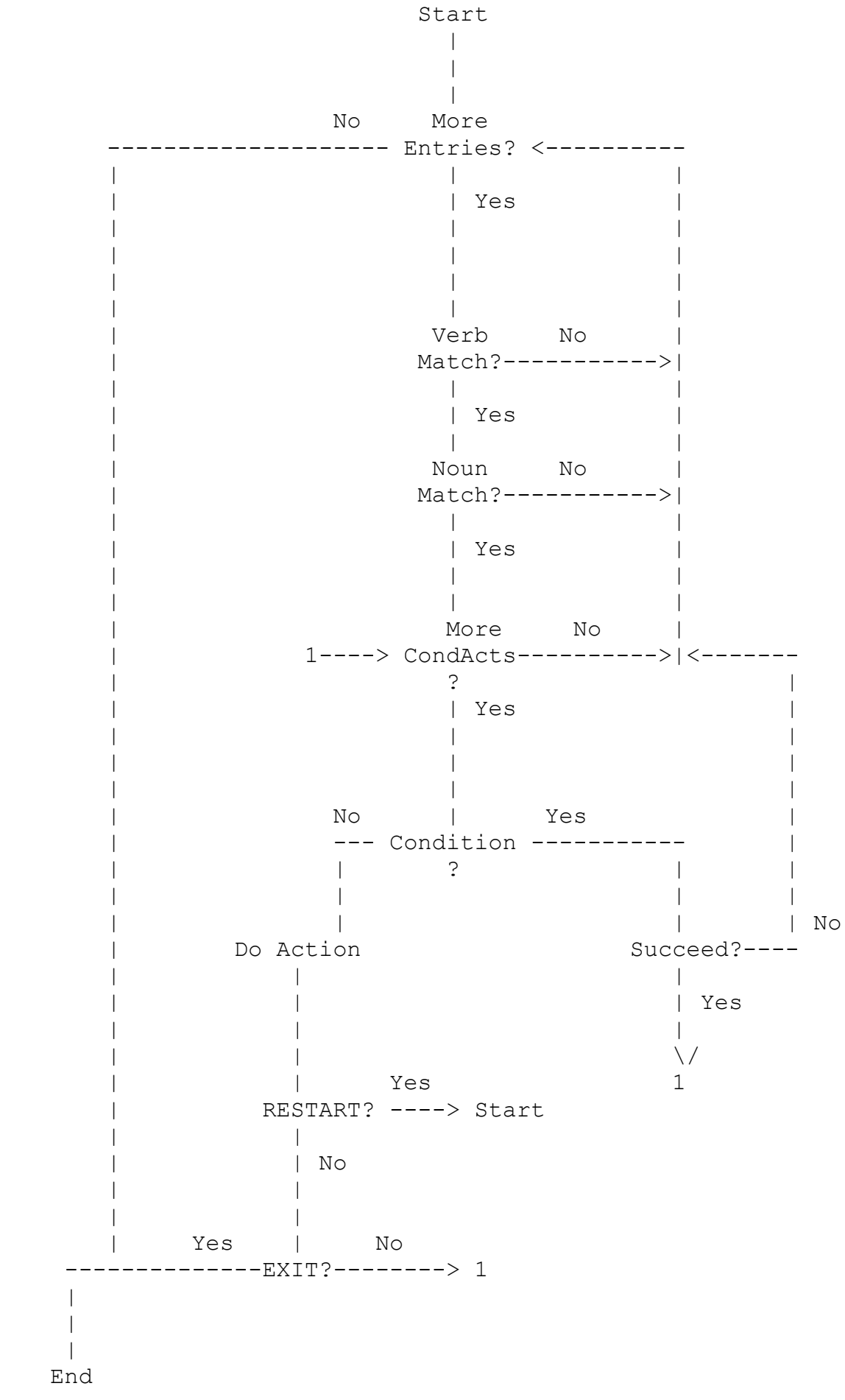
Search Process Table 0

Process table 0 will normally be the main loop of the interpreter, although there is no reason why it can't just carry out some actions and return to the O/S.

Flowchart 1 and the next section describe the scanning of a process table, this applies to Process 0 as it operates just like any other table. The exception being that a DONE or falling off the end will cause a return to the O/S rather than a calling table.

The source file BLANK.SCE gives an example of a minimum game written under DAAD.

#Flowchart 1#


Start
No
More
Entries?
Yes
Verb
No
Match?
Yes
Noun
No
Match?
Yes
More
No
1
CondActs
?
Yes
No
Yes
Condition
?
No
Do Action
Succeed?
Yes
1
Yes
RESTART?
Start
No
Yes
No
EXIT?
1
End

Scan a Process Table

Essentially DAAD will look at each entry in the table until it is exhausted - the table of entries, not DAAD! Assuming there is an entry, it will ensure the Verb and Noun match those of the current Logical Sentence (LS) - This will be set up by a PARSE statement somewhere in the main loop for a traditional text game.

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 a table of "_ _", will cause a trigger of the entry no matter what the LS. This will be the normal entry in Process 0 as it will be very unlikely that a valid LS is present or meaningful for the main loop of the game.

DAAD will then look at each CondAct in turn; if it is a condition, which succeeds, then DAAD 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. If it is an action it will be carried out. Actions can be divided into five main groups:

1/ RESTART; which will completely exit the execution of #all#
tables (i.e. even if in a 10th level sub-process) and jump to the start of Process 0.

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

3/ Exit; any action which will stop processing of the current table and exit to the calling table. e.g. DONE etc.

4/ Conditional Exit; any action which will stop processing of the current table and exit to the calling table 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 DAAD to continue looking at the next CondAct in the current entry. e.g. COPYFF, PLUS etc

Note: Several CondActs in DAAD behave differently to those with the same name in QUILL/PAW, so ensure you check in this manual before using them. QUIT is an example, in DAAD it is a true condition.

4.1.0. CondActs

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

For those used to the PAW systems. Be careful! Several actions have been removed, while others have changed in function. Make sure you check on their function within DAAD. E.g. TURN & SCORE are deleted - they must now be soft coded. Also TIMEOUT, PROMPT and GRAPHIC are gone as 'HASAT TIMEOUT', 'LET Prompt x' and 'LET GFlags expression' will do the same.

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.

4.1.1. Indirection

The first parameter on the majority of actions can use indirection. This is indicated by placing square brackets ([]) around the first parameter of a CondAct. This will cause not the number itself to be used, but the contents of the flag corresponding to that number. Only the first parameter may be indirected but this provides a powerful facility. Although not all commands can indirect, most that have a flag, object or number as the first parameter can be indirected.

E.g. 'MESSAGE [100]' will print the message number that is given by the value in Flag 100 - as opposed to 'MESSAGE 100' which would print message 100!

4.1.2 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 ten 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 .

BIGGER flagno 1 flagno 2

Will be true if flagno 1 is larger than flagno 2

SMALLER flagno 1 flagno 2

The reverse of above - it isn't actualy needed as reversing the parameters of BIGGER would do the same, but it may make programs

more readable and indirection can be used with 'either' flag by using the appropriate condition.

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.

Two conditions provide a boolean TRUE/FALSE test for Sub_process calls:-

ISDONE

Succeeds if the last table ended by exiting after executing at least one Action. This is useful to test for a single

succeed/fail boolean value from a Sub-Process. A DONE action will cause the 'done' condition, as will any condact causing exit, or falling off the end of the table - assuming at least one CondAct (other than NOTDONE) was done.
See also ISNDONE and NOTDONE actions.

ISNDONE

Succeeds if the last table ended without doing anything or with a NOTDONE action.

Object attributes are dealt with by two conditions also:-

HASAT/HASNAT value

Checks the attribute specified by value. 0-15 are the object attributes for the current object. There are also several attribute numbers specified as symbols in SYMBOLS.SCE which check certain parts of the DAAD system flags:-

Symbol	Flag	Att No.	Description
WEARABLE	Flag 57 - Bit7	23	Current object is wearable
CONTAINER	Flag 56 - Bit7	31	Current object is a container
LISTED	Flag 53 - Bit7	55	If objects listed by LISTOBJ
TIMEOUT	Flag 49 - Bit7	87	If Timeout last frame
GMODE	Flag 29 - Bit7	247	Graphics available
MOUSE	Flag 29 - Bit0	240	Mouse available (not supported on all machines at the moment)

The option bits can be set/tested using the defined values given as attributes and bit values in SYMBOLS.SCE

For example the CPC magic draw option which loads the palette colours after drawing using black ink on black paper can be enabled with:-

```
HASNAT  GA_MDRW          ; If we don't have magic
                           draw
PLUS    GFlags GO_MDRW
```

Note; Symbols in the format GA_xxx are the attribute test values for HASAT/HASNAT testing, while GO_xxx is the number used to set/clear (using PLUS/MINUS) the bit.

As a further example; The TIMEOUT condition of PAW is implimented in DAAD by:

```
HASAT TIMEOUT
```

this also allows a 'NOTTIMEOUT' condition to be created using HASNAT!

You can of course assign your own values to parts of a flag and

test them simply with thes HASAT/HASNAT conditions. They are a true general purpose bit tester for the first 64 flags.

Interaction with the player is carried out by two conditions:-

INKEY

Is a condition which will be satisfied if the player is pressing a key. In 16Bit machines Flags Key1 and Key2 (60 & 61) will be a standard IBM ASCII code pair. On 8 bit only Key1 will be valid, and the code will be machine specific.

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 the remainder of the entry is discarded and the next entry is carried out.

4.1.3 Actions

There are twenty one 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 .

RESET

This Action bears no resemblance to the one with the same name in PAW. It has the pure function of placing all objects at the position given in the Object start table. It also sets the relevant flags dealing with no of objects carried etc.

There are five 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).

SETCO objno.

Sets the currently referenced object to objno.

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 the 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 .

COPYBF flagno1 flagno2

Same as COPYFF but the source and destination are reversed, so that indirection can be used. This will hopefully be replaced by a comprehensive system of dual parameter redirection for COPY actions in the future.

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.

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 option

Allows the current window to have its operation flags changed. 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 - Use the upper character set. (A permanent ^G)
2 - SM32 ("More...") will not appear when the window fills.

e.g. MODE 3 stops the 'More...' prompt and causes all characters to be translated to the 128-256 range.

INPUT stream option

The 'stream' parameter will set the bulk of input to come from the given window/stream. A value of 0 for 'stream' will not use the graphics stream as might be expected, but instead causes input to come from the current stream when the input occurs.

Options:
1 - Clear window after input.
2 - Reprint input line in current stream when complete.
4 - Reprint current text of input after a timeout.

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).

Actions to deal with screen output and control;

WINDOW window

Selects window (0-7) as current print output stream.

WINAT line col

Sets current window to start at given line and column. Clipping height and width to fit available screen.

WINSIZE height width

Sets current window size to given height and width. Clipping as needed to fit available screen.

CENTRE

Will ensure the current window is centred for the current column width of the screen. (Does not affect line position).

CLS

Clears the current window.

SAVEAT
BACKAT

Save and Restore print position for current window. This allows you to maintain the print position for example while printing elsewhere in the window. You should consider using a seperate window for most tasks. This may find use in the creation of a new input line or in animation sequences...

PAPER colour
INK colour

Sets current window colours acording to the lookup table given in the graphics editors, or as a pallette number for machines with a palette.

BORDER colour

Sets main screen border colour - this is machine specific.

PRINTAT line col

Sets current print position to given point if in current window. If not then print position becomes top left of window.

TAB col

Sets current print position to given column on current line.

SPACE

Will simply print a space to the current output stream. Shorter than MES Space!

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.

DESC locno.

Prints the text for location locno. without a NEWLINE.

The following 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.

DPRINT flagno

Will print the contents of flagno and flagno+1 as a two byte number.

i.e. a number in the range 0-65535 generated as:
(flagno+1) * 256 + (flagno)

DPRINT 255 is meaningless and will produce a random result.

Two actions dealing with listing objects on the screen. They are controlled/set by 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.

Four actions to allow the current state of the game to be saved & restored; LOAD and SAVE are both followed by a number which means:-
0 - Normal action
1 - Save to TAPE (i.e. doesn't ask "Disc or Tape?")
2 - Save to DISC (ditto)

SAVE opt

This action saves the current game position on disc or tape. SM60 ("Type in name of file.") is printed and the input routine is called to get the filename from the player. If the supplied filename is not acceptable SM59 ("File name error.") is printed - this is not checked on 8 bit machines, the file name is MADE

acceptable!

LOAD opt

This action loads a game position from disc or tape. A filename is obtained in the same way as for SAVE. A variety of errors may appear on each machine if the file is not found or suffers a load error. Usually 'I/O Error'. The next action is carried out only if the load is successful. Otherwise a system clear, GOTO 0, RESTART is carried out.

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. The next action is always carried out.

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 1: The RAM 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 the main loop which does an automatic RAMSAVE every time the player enters a move.

Note 2: These four actions allow the next Condact to be carried out. They should normally always be followed by a RESTART or describe in order that the game state is restored to an identical position.

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.

Three actions to deal with control of the parser;

PARSE n

The parameter 'n' controls which level of string indentation is to be searched. At the moment only two are supported by the interpreters so only the values 0 and 1 are valid.

0 - Parse the main input line for the next LS.
1 - Parse any string (phrase enclosed in quotes [""]) that was contained in the last LS extracted.

Mode 0 is the primary method for converting the current input line
of the player to a logical sentence (LS). The detailed description of the parser provides further details. The command will extract the next LS from the current input line.

Mode 1 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 at level 0 then the input line is called preceeded with a random prompt - this gets a new input line from the player automatically when required, removing the problem of handling multiple phrases by the programmer. At level 1 and above the next CondAct is carried out. This occurs at all levels if the LS is invalid. Note that DAAD will look at the next condact in what can be considered a fail situation - this is different to what you might expect. 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 carefully - this is also the reason for doing the next CondAct in a fail situation, think about it!

If you are using a text based command structure you will need at least one PARSE 0 CondAct in the main loop somewhere.

e.g. the minimum process 0 without an initialisation might be:-

```
_     _     PARSE 0
            MESSAGE "I don't understand"
            REDO

_     _     PROCESS x   ; Deal with any commands
            REDO
```

To use it to speak to a PSI there will be two or more calling entries (in process x of the above example) 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
```

With a PROCESS y similar to:-

```
     _     _     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
```

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!

SYNONYM verb noun

Substitutes the given verb and noun in the LS. Nullword (Usually '_') can be used to suppress substitution for one or the other - or both I suppose!

```
         e.g. MATCH ON    SYNONYM LIGHT MATCH

              STRIKE MATCH SYNONYM LIGHT _

              LIGHT MATCH ....    ; Actions...
```

will switch the LS into a standard format for several different entries. Allowing only one to deal with the actual actions.

Several actions which deal with jumping, looping and subroutine control:-

PROCESS procno.

This powerful action transfers the attention of DAAD 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.

REDO

Will restart the currently executing table, allowing

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 DAAD 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.

SKIP distance | label

where distance is -128 to 128, or to the specified label.

Will move the current entry in a table back or fore. 0 means next entry (so is meaningless). -1 means restart current entry (Dangerous). There is no error checking so it should be possible to jump out of the table (Fatal).

Skip can also accept as a parameter a local symbol. This is a symbol preceeded by a $ sign. They are local to each process table and will not affect any global symbols used. The big advantage is that they can forward reference.

This is implimented with rear patching. The symbol is defined by placing it in the source file immediatly before the entry it refers to:

e.g.

```
$backloop
_       _       PRINT   Flag
                MINUS   Flag 1
                NOTZERO Flag
                SKIP    $backloop

_       _       ZERO    Error
                SKIP    $Forward

_       _       EXIT    0

$Forward
_       _       ...
```

The three actions which completely exit Response/Process execu# tion:

RESTART

Will cancel any DOALL loop, any sub-process calls and make a jump to execute process 0 again from the start.

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 the operating system - by doing the command EXIT 0.

EXIT value

If value is 0 then will return directly to the operating system. Any value other than 0 will restart the whole game. Note that unlike RESTART which only restarts processing, this will clear and reset windows etc. The non zero numbers actually specify a part number to jump to on AUTOLOAD versions. Only the PCW supports this feature at the moment. It will probably be added to PC as part of the HYPERCARD work. So if you intend using it as a reset ensure you use your PART number as the non zero value!

Three exit table actions;

DONE

This action jumps to the end of the process table and flags to DAAD 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.

There are four actions used to call an external routine to DAAD. Two SFX and GFX are already provided with a function on several machines while EXTERN and CALL provide general purpose call mechanisms. All these commands are documented fully in the chapter on external commands.

EXTERN value

Calls external routine with parameter value. The address is set by linking or the #extern pre-compiler command - see the machine details and extern section for specifics.

CALL address

Allows 'address' in memory (or in the database segment for 16bit) to be executed. See the extern secion for more details.

SFX value1 value2

This is a second EXTERN type action designed for Sound Effects extensions. e.g. It has a 'default' function which allows value 'value1' to be written to register 'value2' of the sound chip on 8 bit machines. This can be changed with #sfx or through linking - see the machine details and extern section for specifics.

GFX value1 value2

An EXTERN which is meant to deal with any graphics extensions to DAAD. On 16bit it is used to implement the screen switching facilities. This can be changed with #gfx or through linking. See the machine details and extern section for specifics.

There are two actions to implement the primary graphics handling facilities of DAAD. The descriptions especially for 16bit users should be read in conjunction with the details of the GFX command, which provides the extended graphics handling.

PICTURE picno

Will load into the picture buffer the given picture. If there is no corresponding picture the next entry will be carried out, if there is then the next CondAct is executed.

e.g. The describe of a location in Process 0 would look something like:-

```
_       _       PICTURE [Player]    ; Try to load/set picture
                DISPLAY [Dark]      ; If not dark display

_       _       DESC    [Player]    ; Text description anyway
```

Take a look at BLANK.SCE for the full entry dealing with darkness fully etc.

DISPLAY value

If value=0 then the last buffered picture is placed onscreen.

If value !=0 and the picture is not a subroutine then the given window area is cleared. This is normally used with indirection and a flag to check and display darkness - see example for the PICTURE action.

On a drawstring machine when a location picture (non subroutine) is drawn, the drawing cursor position is set to the origin and the colours to those in the colour table. For a subroutine the picture will start from the endpoint of the last draw. Note that only the cursor position is maintained - the colours always start at the current colours for the stream.

On pixel type machines the action will introduce the required window data and clipped picture to the current stream. If you size your graphics correctly you can use this to erase a displayed picture by selecting the same stream and using the CLS action.

Miscellaneous actions:-

MOUSE option

This action in preparation for the hypercard system implements a skeleton mouse handler on the IBM.

4.2 The Machine Interpreters

This section provides information on the use of the interpreter for each machine. It also provides any relevent information about machine specific items related to extern handling/mastering etc.

4.2.0 The IBM

Running the interpreter

There are a large number of versions of the interpreter, depending on language, debugging features, graphics system etc. All are supplied as an .OBJ file to allow linking with EXTERN routines.The method used to name them is explained below.

The final .EXE files are standard MSDOS programs. They require the operating system to be present to do their disc handling - most screen output is handled within the system.

The most common file you will be working with will be INTSDM if you are using the Spanish language. This is the 'Interpreter, Spanish, Developer, Multi-graphics' version. The features of the development interpreter are also explained in the section on setting up the development machine.

In order to use the OBJ files you must combine them using LINK with any external routines you are using.

e.g. In the \DAAD directory (with LINK.EXE somewhere on the PATH):-

```
LINK \OBJ\INTSDM+EXTERN+CHARS,INTSDM;
```

Will create an INTSDM.EXE file.

The EXTERN module contains any routines to deal with EXTERN. While the CHARS module contains several external variables which can be changed by reassembling CHARS.ASM. The CHARS module must always be the last module listed in the link.

The above link used minimum EXTERN and CHAR files that are supplied in an .OBJ format for those who don't have an assembler. In order to change the number of parts in production games (set to 2 in the supplied CHARS.OBJ) and to add external routines you will need to be able to assemble the .ASM files. This will require an assembler such as MASM which produces LINK compatible object files.

Non developer interpreters have the form:-

```
INTl{M {V}}
```

The developer interpreters:-

```
     INTlD{M} name {M|C|E|V}
```

where l is the language

M present indicates multi-machine (.DAT) graphics files (The V option is valid only with these machine versions) otherwise .CGA/.EGA files from DMG will be searched for.

name is the name of a .DDB file (you cannot specify a DRIVE/PATH or extension here). - Developer Only

```
M selects text MDA mode                                  \
C selects CGA four colour 320x200 mode (default)  |Developer only
E selects EGA sixteen colour 320x200 mode         /
V selects VGA 16 colour palette 320x200 mode
Developer/Release
```

(the V option can be used only with the 'M' type interpreters)

examples:

Assuming you have LINKed the versions of the interpreter you require and placed them in \DAAD or any directory on the PATH:-

```
A>INTSDM CAVE
```

would run the Spanish adventure CAVE.DDB from the default drive in CGA using multi-machine graphics (a .DAT file sent from the ST), providing full debug information.

```
A>INTED CAVE E
```

would run the English adventure CAVE.DDB in EGA using a PC-DMG produced .EGA graphics file, providing full debug information.

```
A>INTSM
```

would run the Spanish adventure PART1/2/x (as defined in CHARS) - selected by the user. Using multi-machine graphics in the highest graphics mode available in the order EGA,CGA. If no graphics card can be found then a switch is made to MDA text mode.

```
A>INTEM V
```

would run the English adventure PART1/2/x (as defined in CHARS) - selected by the user. Using multi-machine graphics in VGA mode if available else in the order EGA,CGA. If no graphics card can be found then a switch is made to MDA text mode.

Language special characters

The IBM interpreter for Spanish recognises the standard 'dead' keys to generate accented characters and special symbols. This will of course only work if the correct keyboard driver (using KEYB) has been installed.

Special Notes

The IBM (and indeed the ST/AMIGA) interpreter uses a method of fitting 53 text columns onto the 320x200 pixels in the graphics modes. This is achieved by using 6 pixels for the width only. There is a catch though. All scrolling and clearing is carried out on the 40 columns provided by an 8pixel character width. This is similar to the method used on the Spectrum to obtain 42 col# umns.

So you should take care when specifying window sizes. Their size will always be modified to the nearest 8 pixels suitable to contain the required number of text columns. The actual text printer allows true 53 column windows, it is only with clearing and scrolling that any problem may occur. A general rule is that at least one text column gap should be left between windows if you want them to scroll or clear without affecting adjacent windows.

Files to do with the game

The following files will need to be present in the current directory for the game to run. The Interpreter files can be anywhere as long as they are on the path of course.

1/ Old PC DMG graphics:

```
INTl.EXE        ;Interpreter
```
This file should be linked with a CHARS module with the correct number of 'parts'. The CHARS.OBJ supplied has it set to two. The file should be renamed to your game name.

```
PARTx.CHR       ;Character set data (from Spectrum usually)
PARTx.DDB       ;Text database (from .SCE files and DC)
PARTx.CGA       ;CGA graphics file (from DMG)
PARTx.EGA       ;EGA graphics file (from DMG)
```
These three files must be present for each part of your game

```
PART1.CGS       ;CGA loading screen (use CSTA/CACGA utilities)
PART1.EGS       ;EGA loading screen (made from PI1 using CSTEGA)
```
The screen files are optional, a blank screen is presented if no file is found.

for an autoswitching CGA/MDA version only remove the EGA files and replace the interpreter with 'INTlCGA.EXE'.

2/ New ST DMG graphics:

INTlM.EXE The interpreter where l is S or E at the moment
This file should be linked with a CHARS module with the correct number of parts set. The CHARS.OBJ supplied has it set to two. The file should be renamed to your game name.

PARTx.CHR A character set
PARTx.DDB The database (from .SCE files and DC)
PARTx.DAT Common graphics file (from DMG on the ST)
These three files must be present for each part of your game

PART1.CGS ;CGA loading screen (use CSTA/CACGA utilities)
PART1.EGS ;EGA loading screen (made from PI1 using CSTEGA)
PART1.VGS ;VGA loading screen (made from PI1 using CSTVGA)
The screen files are optional, a blank screen is presented if no file is found.

Files needed for a standalone copy

In addition to the files listed above if you want the disc to autoboot then you need to have formatted the disc with the '/S' option. Then the files COMMAND.COM and AUTOEXEC.BAT (which will contain at least one line with the name of the game) will need to be present.

REMEMBER You are not allowed to release software with MSDOS present - it is against the distribution agreement.

4.2.1 The Spectrum

Running the interpreter

On DISCiPLE

```
LOAD d1"drs"
```

On +3

```
LOAD "drs"
```

Files required to run the game

You will need the database and graphics files.

Files required for a standalone copy

This is best achieved by combining each of the memory blocks and saving them as a single file. A simple BASIC loader based on the 'drs' or 'drl' programs can be used to load a screen and the file.

Language special characters

The special characters of Spanish are entered by selecting EXTENDED MODE (i.e. Single key on 128/+2, or CAPS and SYMBOL SHIFTS on 48K), then pressing the corresponding key as shown in the table below:-

!,?,a,e,i,o,u,n.
,¨, ,,,¡,¢,£,¤

In addition if key D or F are placed after EXTENDED MODE you will get ¤ and ¥ respectivly, which are the keys they are on in Spain!

Special Notes

The Spectrum interpreter (and indeed the MSX which emulates it) uses a 40 column mode which crams 4 text columns into a group of three normal columns. Note that this means colour boundaries for windows must be worked out carefully as only whole Spectrum columns are cleared and scrolled by the window system - as though the attribute screen is stuck at 32 columns by the hardware. If two 40col columns fall in a 32col column, then clearing, scrolling or setting attributes will affect both 40cols #whichever# you are using actually. This does not apply to printing as the print system masks its characters correctly allowing windows to contain text as you would expect.

4.2.2 The CPC

Running the interpreter

For the Amstrad you will need to create a runable program. The program MCRF.COM is used under CPM to create a fully self con# tained program which can be started using RUN"name". The format is:-

MCRF oufile{.BIN} interp{.Z80} text{.DDB|.BIN} graphics{.BIN}

Note that you must specify the type of text database. DDB is from the compiler direct and BIN is assumed to have a CPC disc header. The default at the moment is BIN, but we advise you send DDB files and give the full name. This means you don't need to use 'AAH' on the PC to add an Amstrad disc header.

Files required to run a game

The disc will have to contain CPM if working with CPM2.2 as you need to run MCRF.
DCPCIl.Z80 ;The interpreter
Where l is E for English or S for Spanish
name.DDB ;A text database from the PC
name.BIN ;A graphics file drawn/edited with DG.

MCRF.COM

Files required for a standalone copy

Only the runable BIN file created by MCRF is needed. You may add a small boot program to load a screen, set colours etc if required.

Language Special characters

The special letters are as follows on DCPCIS.Z80

CTRL a,e,i,o,u,n - ,,,¡,¢,£,¤
CTRL v - O
CTRL 1 -
The /? key now gives ¨?
¤ works as expected

Notes

The INK & PAPER options set the ink to use 0-3 for writing characters, these depend on the colours set in the graphics database. BORDER sets the actual physical colour for the border 0-26.

4.2.3 The CBM

Running the interpreter

The Spanish interpreter is called SDI or DC64IS.

The interpreters contain a small BASIC program which allows them to be saved to TAPE and reloaded using SHIFT/RUNSTOP, or from disc using LOAD"name",8 and RUN. Although for anything other than small programs it is better to use the loaders described below as the LOAD"",8,1 memory load is unreliable.

There are two small programs on the release disk called LDS1 and LDS2 these can be renamed to anything you like.

You start them with:-

```
LOAD "name",8,1          ; Where name is the renamed LDSx
```

They will then load SDI and the text and graphics databases which you must call PARTx, then start the game.

Language Special characters

The special letters are obtained as follows on the CBM for the Spanish interpreter DC64IS.

CTRL a,e,i,o,u,n - ,,,¡,¢,£,¤
CTRL v - O
<- (next to 1 topleft) -
The /? key gives ¨?
[: is now ¤¥ (upper and lower case)

4.2.4 The MSX

Works just like all the other versions really. The Spanish interpreter recognises all the accented letters as typed from the keyboard.

Also see the note about 40 columns on the Spectrum.

There is a subtle difference between the MSX and Spectrum: Each pixel line on MSX has its own colour attribute. This can cause some slight problems in the graphics when ported. In order to maintain speed only the pixel attributes above and below the one being plotted are set (this is not strictly true, they are only set if the correspond to an 8x8 line,column position). This will provide a fair emulation of the Spectrum but can cause hassle if the graphics are drawn assuming that the entire charcter cell colour will be set when drawing a line through it. N.B. IT IS TOO SLOW to set all 8 pixel lines!

4.2.5 The ST

At the moment we supply two versions of the interpreter:

lDIn.PRG
lDInL.PRG
Where l is the language and n the number of parts.

The xxxL.PRG versions do not allow the program to run in Medium Res. This effectively stops use in text mode.

Running the interpreter

Just click on the PRG file. This can of course be renamed to whatever you want.

Files required to run the game

lDIn.PRG or lDInL.PRG (renamed to the game name)

PARTx.CHR A character set for hi-res 8x8 mode
PARTx.CH0 A character set for lo-res 6x8 mode
PARTx.DDB The database (from .SCE files and DC)
PARTx.DAT Common graphics file (from DMG)
These four files must be present for each part of your game. You can omit the CHR file if using the lo-res only interpreter.

PART1.SCR ;loading screen (renamed PI1 file)

The screen file is optional, a blank screen is presented if no file is found.

You may also like to open a window, centre the PRG file in it and save the desktop. As this may move which file is displayed (because there is now another file on the disk) you may need to repeat the process.

The requirements for a standalone copy are the same!

Language special characters

You do not need the ACCENTS program in an auto folder. In fact if you add it you will have to press the accent keys TWICE! Other than that the 'dead' keys work as you would expect.

4.2.6 The Amiga

There is only a single interpreter for each language on the Amiga

lDIn

Where l is the language and n the number of parts.

Running the interpreter

From the command line type the name of the interpreter. If you create a 'TOOL' type icon and give it the same name as the inter# preter you can just click on the program icon. They can of course be renamed to whatever you want.

Note that the interpreter is a true multi-tasking process. You can grab the top line of the screen to drag it down to reveal the desktop. If memory allows you can start another copy of the game or a different part or even different game/program! Also you can click on the top right part of the screen just as if the usual intuition 'move window to the back' and 'move window to the front' gadgets were present.

Files required to run the game

lDIn (renamed to the game name)
lDIn.info (a TOOL type icon)

PARTx.CHR A character set for lo-res 6x8 mode
PARTx.DDB The database (from .SCE files and DC)
PARTx.DAT Common graphics file (from DMG on the ST)
These three files must be present for each part of your game.

```
PART1.SCR      ;loading screen (PI1 file converted to .BIT with
                DSTA on ST)
```

The screen file is optional, a blank screen is presented if no

file is found.

Files required for a standalone copy

You will need to create a disk which contains the Minimum AMIGA OS. (If you keep a copy of this disk you can just make a copy each time you make a master).

Place all the above files on the disk then change the last line in the 's/startup-sequence' directory to the name of your interpreter.

Language special characters

You must ensure that the correct langauge keyboard driver is present in the device folder of the boot disk. If so all the accents will work as expected!

4.2.7 The PCW

Only one interpreter is supplied for each laguage on the PCW:

DPCWIl.Z80
where l is the language. This needs to be renamed as a .COM file before you can use it.

Running the interpreter

From the A: prompt type the name of the interpreter.

Files required to run the game

DPCWIl.Z80 renamed to gamename.COM

PARTExxx.CHR A character set for 8x8 mode
PARTExxx.DDB The database (from .SCE files and DC)
PARTExxx.DAT Common graphics file (from DMG on PC)
These three files must be present for each part of your game. You can omit the CHR file if using the lo-res only interpreter.

PART001.SCR ;loading screen (CST1PCW from a PI1 file)
The screen file is optional, a blank screen is presented if no file is found.

The requirements for a standalone copy are the same!

Language special characters

All the accent keys will work as expected.

Notes:-

The screen on the PCW has 31 lines by 96 columns and this will

need some special processing. All the window commands can use this size screen, but we suggest that you will need quite a few #if PCW statements!

This is the first interpreter to feature the AUTOLOAD system. This allows the EXIT n command to select part 'n' which will then be loaded automatically.

All filenames have to have the format PARTExxx.eee.

The interpreters expect to find all of part 1 files on side A, and all of part 2 on side B. We will hopefully produce an install utility in the future, but this should be OK for the moment.

4.3 Errors

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

The interpreters can throw up several types of error. Usually in a little window at the top left on 8 bit and centred on 16 bit, but if during a save they will be printed in the current input stream as they do not cause the game to restart.

The errors are:

I/O Error	These two are obvious
BREAK	
Error n	Where n is the normal machine error
Game Error n	Where n is one of
	0 - Invalid object number
	1 - Illegal assignment to HERE (Flag 38)
	2 - Attempt to set object to loc 255
	3 - Limit reached on PROCESS calls
	4 - Attempt to nest DOALL
	5 - Illegal CondAct (corrupt or old db!)
	6 - Invalid process call
	7 - Invalid message number
	8 - Invalid PICTURE (drawstring only)

In the debug version of the interpreters, Game/Runtime Errors are followed by four numbers in the form p:v,n>c where p is the process number, v and n are the word numbers of the verb and noun of the entry which caused the error, and c is the CondAct number as given in the section on the compiler.

a) The error Nos. have the meanings given above.

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	32	AUTOD	64	BEEP	96	INPUT
1	NOTAT	33	AUTOW	65	PAPER	97	SAVEAT
2	ATGT	34	AUTOR	66	INK	98	BACKAT
3	ATLT	35	PAUSE	67	BORDER	99	PRINTAT
4	PRESENT	36	SYNONYM	68	PREP	100	WHATO
5	ABSENT	37	GOTO	69	NOUN2	101	CALL
6	WORN	38	MESSAGE	70	ADJECT2	102	PUTO
7	NOTWORN	39	REMOVE	71	ADD	103	NOTDONE
8	CARRIED	40	GET	72	SUB	104	AUTOP
9	NOTCARR	41	DROP	73	PARSE	105	AUTOT
10	CHANCE	42	WEAR	74	LISTAT	106	MOVE
11	ZERO	43	DESTROY	75	PROCESS	107	WINSIZE
12	NOTZERO	44	CREATE	76	SAME	108	REDO
13	EQ	45	SWAP	77	MES	109	CENTRE
14	GT	46	PLACE	78	WINDOW	110	EXIT
15	LT	47	SET	79	NOTEQ	111	INKEY
16	ADJECT1	48	CLEAR	80	NOTSAME	112	BIGGER
17	ADVERB	49	PLUS	81	MODE	113	SMALLER
18	SFX	50	MINUS	82	WINAT	114	ISDONE
19	DESC	51	LET	83	TIME	115	ISNDONE
20	QUIT	52	NEWLINE	84	PICTURE	116	SKIP
21	END	53	PRINT	85	DOALL	117	RESTART
22	DONE	54	SYSMESS	86	MOUSE	118	TAB
23	OK	55	ISAT	87	GFX	119	COPYOF
24	ANYKEY	56	SETCO	88	ISNOTAT	120	internal
25	SAVE	57	SPACE	89	WEIGH	121	COPYOO
26	LOAD	58	HASAT	90	PUTIN	122	internal
27	DPRINT	59	HASNAT	91	TAKEOUT	123	COPYFO
28	DISPLAY	60	LISTOBJ	92	NEWTEXT	124	internal
29	CLS	61	EXTERN	93	ABILITY	125	COPYFF
30	DROPALL	62	RAMSAVE	94	WEIGHT	126	COPYBF
31	AUTOG	63	RAMLOAD	95	RANDOM	127	RESET

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

4.4 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 - a Noun with a word value <20 - which 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!

4.4.1 Spanish

If a Verb is less than 5 letters you will need to include the lo,la,los and las versions in the vocab. Obviously if it is four letters you only need 'l' ending as a synonym, if it is three letters you need 'lo' and 'la' synonyms and if it is one or two letters you need 'lo','la','los' & 'las'!

If you have a plural noun in the game which changes its stress then you need to include the stressed and unstressed - See talones in the demo game for an example.

The Spanish Parser deals with NOUNS,PRONOUNS and ADJECTIVES differently to the English. Specifically it assumes that adjectives FOLLOW nouns, and does not deal with compund nouns. A compound noun is where an object is described by two nouns such as PARK BENCH. where the player may use either or both words to describe the item. E.g. GET BENCH, GET PARK, GET PARK BENCH. PARK and BENCH would probably be synonyms and the problem does not arise until you use a second noun. E.g. PAINT PARK BENCH WITH BRUSH. Here BRUSH should be NOUN2, but the parser would assume PARK was NOUN1 and BENCH was NOUN2. The English parser deals with this situation, but it is complicated to do for Spanish.

4.5 The flags

The normal flags are free for use in any way in games. But see the SYMBOLS.SCE file. This defines a use for every flag from 0-63 - the 'system' flags. It also defines symbolic names for the system flags. Although DAAD does not reference all of these (only those shown below) they may be treated specially by future upgrades so treat them with care.

The best way to test the bit defined flags is to use the HASAT CondAct. For example HASAT MOUSE (assuming SYMBOLS.SCE has been included) will be true if a mouse is present.

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)

Flags 2 to 28 are not actually used by the DAAD interpreters anywhere. Thus they are free for use in your own games. Athough the DAAD SYMBOLS.SCE file defines some predefined uses which we suggest you adhere to.

Flag 29
7 - Set if there are graphics available. If the flag is less than 128 then you can assume that you are in 80 column text mode (because only the ST and the IBM will not set this bit, and that is when they are running in 80 column text mode!
6 - Invisible draw mode for Drawstring machines. When set this bit causes drawstring machines that have a palette (i.e. CPC!) to refrain from setting the colours until the entire picture has been draw. If the pictures are complicated then it looks a little too much like the machine has crashed!
5 - Pictures OFF (drawstring only)
4 - Wait for a key after drawing picture (drawstring)
3 - Change BORDER to picture colours (drawstring)
2/1 Undefined
0 - Mouse present (16 bit only).

Flag 30 Score flag - not actually used directly by DAAD but its traditional!
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 39/40 Unused

Flag 41 Gives stream number for input to use. 0 means current stream. Used Modulo 8. I.e. 8 is considered as 0!

Flag 42 holds prompt to use a system message number - (0 selects one of four randomly)

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
7 - Set if timeout occurred last frame
6 - Set if data available for recall (not of use to writer)
5 - Set this to cause auto recall of input buffer after timeout
4 - Set this to print input in current stream after edit
3 - Set this to clear input window
2 - Set this so timeout can occur on ANYKEY
1 - Set this so timeout can occur on "More..."
0 - Set this so timeout can occur at start of input only

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
7 - Set if any object printed as part of LISTOBJ or LISTAT
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

Flag 58/59 is the currently referenced objects user attribs

Flags 60 & 61 are the Key flags which give the key code returned after an INKEY condition succeeds. Flag 61 is only relevant on IBM,ST and AMIGA where it is used to provide the extended codes when a cursor or function key is pressed. In this case Flag 60 is

zero and Flag 61 contains the IBM extended code.

Flag 62 on ST and PC gives the absolute screen mode in use on the machine. This allows checks to be made as to size of screen etc, but to determine if you are in graphics mode see Flag 29.
On the ST:
0 - lo-res
1 - med-res
On the PC:
4 - Means CGA
7 - Means mono character only
13 - is EGA or VGA
+128 (Bit 7 is set) in VGA to indicate you have palette switching.

Flag 63 defines the currently active window. Note that this is a copy so changing its value will not affect the DAAD system.

Flag 64 to 255 are available for your own use.

Section 5.0 - The Source File

The source file consists of a number of inter-related sections describing the adventure. They usually correspond with the areas found in the database. The sections are:-

The Control (CTL) section

This section tells the compiler which character has been chosen as a null word. Throughout this manual the null word is assumed to be an underline _. It should not be necessary to change this setting, it is not advised anyway.

The Vocabulary (VOC) section

Each entry in this section contains a word (or the first five characters of a word), a word value and a word type. Words with the same word value and type are called synonyms. An optional language specifier can be given to fix a word to a specific language.

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) The noun and adjective associated with the object.
d) Whether the object is a container.
e) Whether the object can be worn/removed.
f) The state of the other object attributes

The Process tables (PRO) section

This section forms the heart of the source file providing the main game control. Each table consists of a number of entries. 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. The LS can of course be set using a method other than the PARSE action. This would allow the creation of a menu system, multiple-choice entry etc.

Process table 0

This contains the main control of a DAAD program. It is entered after initialisation with the current LS empty. It will normally consist of '_ _' word entries and some form of looping.

Process 1 (and upwards)

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

Compiler Pre-processor commands

This is not really a section, although some of the commands may generate data in the database. The compiler accepts a number of commands to carry out a variety of tasks at compile time which may be interspersed within the source file. Their syntax is included in following section along with an exceptions to their use. Although there is generally no restriction on the placing of pre-compiler commands, there are places where their use is inappropriate or pointless!

5.1 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 BLANK.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 the first column is regarded as a comment. Blank lines are only allowed in the process tables. Any line starting with a # is considered to contain a pre-compiler command as is any line starting with /LNK. Within the process tables only, a line beginning with a $ will be considered to contain a local label.

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.
d) When a number is required, unless otherwise indicated, you may use an expression made up of numbers and/or symbols joined by the operators plus '+' and minus '-'. The expression is evaluated left to right. It must evaluate to a single number. i.e. Undefined symbols cause an error. This means that you cannot use forward references.
   e.g. (assuming PSIBase=100):-
   PSIBase+9 will evaluate to 109.
e) 'comment' is any sequence of characters, they will be ignored. Although the compiler allows them without any leading character in certain places it is advised that you always place a semicolon ';' in front.

CTL

The Control section consists of 3 lines as follows:-

```
/CTL {comment} EOL
NULLWORD {comment} EOL
```

Note:
a) NULLWORD is the character to be used as a null word. One character not a-Z or 0-9. It is not advised that you change this definition.

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{LANG}{w-s}{;comment}} EOL
```

Notes:
  a) Only the 1st 5 chars of WORD are significant,
  b) They are converted to upper case & only A-Z 0-9 allowed, plus any special characters as defined in the current language. For example if command line option 'l1' (Spanish) is selected ,,,¡,¢,£,O,¤ can be used in Vocab words.
  c) Duplicate words are not allowed.
  d) VALUE is in range 1-254
  e) WORDTYPE must be one of; VERB, NOUN, ADJECTIVE, ADVERB, PREPOSITION, PRONOUN or CONJUGATION.
  f) In the interpreter Nouns < 20 can be used as verbs if no verb is entered. Verbs < 14 are used as movement words.
  g) LANG is an optional one letter language tag. E.g. S for Spanish. E for English. This ensures that the word is included only when the SCE file is compiled with that language selected on the command line. This is used with the #lookup pre-compiler command to allow multi-language .SCE files.

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 /

Notes:
  a) 'n' must evaluate to consecutive numbers & start at zero.
  b) STX limit is 60-255
  c) MTX limit is 1-255
  d) OTX limit is 1-255
  e) LTX limit is 1-252

f) The text lines should only contain normal ASCII printable characters, or an escape code as described below. 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 or greater than 127 is considered a control code including TAB's (which are code 9). 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.

g) Special characters can be included in the text using escape sequences generated by the escape character backslash '\':-

\s is accepted in text as a space. It is needed for putting a space on the end of lines - or at the start to avoid TAB codes. This makes listings more obvious and overcomes the fact that some text editors remove spaces from the end of lines during editing.

\f is used as a forced space. It is actually a seperate character in the character set (code 127 funnily enough) which looks like a space. This allows two words to be kept together that the word#wrapping system would split up. This is most commonly needed for names or trademarks which should be kept together. E.g. Kit\fKat in the source file would ensure the two words Kit and Kat were always on the same line.

\b (borrar!) will activate the CLS function of the print routine to clear the current window.

\k will activate a wait for a key during printing. You would normally use this with \b. e.g. \k\b wait for a key, clear the window and continue with further text

\g will activate a shift to the (normally graphic) upper half of the character set. In IBM text mode this will cause all charac#ters to be ignored except the CAPS A-Z which are displayed as such. This is because it is assumed that you will use these for special text words (such as names of spells etc in another character set) and they will need to be visible in text mode.

\t will restore the lower (text) part of the character set

\A -> \P will generate the DAAD character code 16-31. These are the special language characters which vary for the different languages. An appendix gives further details of the character set.

\\ will generate a single backslash character.

h) When a language is specified using command line option '1'

from the compiler several extra characters can be used directly in the text. Your text editor must be able to handle the IBM special characters to be able to use this facility. The codes which are accepted when 'l1' Spanish is selected are shown below:-

Symbol	IBM code	DAAD code	Escape
¦	166	16	A
	173	17	B
¨	168	18	C
®	174	19	D
¯	175	20	E
	160	21	F
¸	130	22	G
¡	161	23	H
¢	162	24	I
£	163	25	J
¤	164	26	K
¥	165	27	L
‡	135	28	M
€	128	29	N
O	129	30	O
š	154	31	P

Any of these can be used in text (and will appear correctly on the target machine).

If you need to access the characters in any language then use the escape sequences \A to \P (using capital letters). This will not then require the language switch to be selected. Although what the character appears as may change from language to language.

i) 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.
```

j) 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

```
I am big.
                    I am big.
```

So are you. | So are 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

Notes:
- a) There must be the same no of entries as in LTX.
- b) 'n' must evaluate to consecutive numbers & start at zero.
- 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

Notes:
- 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 each of which starts with the line:-

/PRO {w-s} n {w-s{comment}} EOL

Notes:
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 1 process table.
c) Blank lines between entries or lines starting w-s; are ignored.
d) Lines starting with $ are assumed to specify a local label. They should normally be placed on the line immediatly before an entry. They record the entry number of the next entry for use with SKIP.

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

Notes:
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 nullword character.
c) KEYWORD can be a CONDITION or ACTION.
d) Anything in col 1 is assumed to be the start of the next entry.
e) PARAM1 only may be enclosed in square brackets '[]' for most CondActs which have a parameter. This indicates that indirection is required. This will cause the CondAct to use the contents of the flag specified by the parameter as the actual parameter. e.g. (assuming flag 100 contains 9):-

```
SETCO 12       ; Sets the current object number to 12
SETCO [100]    ; Sets the current object number to 9
```

Valid CondActs are summarised below. Details of their action is included in the description of the interpreter:-

Conditions

AT	locno	ADVERB	adverb	
NOTAT	locno	HASAT	objno	attrib
ATGT	locno	HASNAT	objno	attrib
ATLT	locno	ISAT	objno	locno+
PRESENT	objno	ISNOTAT	objno	locno+

```
ABSENT  objno                 PREP    preposition
WORN    objno                 NOUN2   noun
NOTWORN objno                 ADJECT2 adjective
CARRIED objno                 SAME    flagno  flagno
NOTCARR objno                 NOTEQ   flagno  value
CHANCE  percent               NOTSAME flagno  flagno
ZERO    flagno                BIGGER  flagno  flagno
NOTZERO flagno                SMALLER flagno  flagno
EQ      flagno  value         BIGGER  flagno  flagno
GT      flagno  value         ISDONE
LT      flagno  value         ISNDONE
ADJECT1 adjective
```

Actions

```
SFX     value   value         MINUS   flagno  value
DESC    locno                 LET     flagno  value
QUIT                          NEWLINE
END                           PRINT   flagno
DONE                          SYSMESS smesno
OK                            SETCO   objno
ANYKEY                        SPACE
SAVE    device                LISTOBJ
LOAD    device                EXTERN  value
DPRINT  flagno                RAMSAVE
DISPLAY value                 RAMLOAD flagno
CLS                           BEEP    value   value
DROPALL                       PAPER   colour
AUTOG                         INK     colour
AUTOD                         BORDER  colour
AUTOW                         ADD     flagno  flagno
AUTOR                         SUB     flagno  flagno
PAUSE                         PARSE
SYNONYM word    word          LISTAT  locno+
GOTO                          PROCESS prono
MESSAGE mesno                 MES     mesno
REMOVE  objno                 WINDOW  stream
GET     objno                 MODE    value
DROP    objno                 WINAT   line    col
WEAR    objno                 TIME    value   value
DESTROY objno                 PICTURE picno
CREATE  objno                 DOALL   locno+
SWAP    objno   objno         MOUSE   value   value
PLACE   objno   locno+        GFX     value   value
SET     flagno                WEIGH   objno   flagno
CLEAR   flagno                PUTIN   objno   locno
PLUS    flagno  value         TAKEOUT objno   flagno

NEWTEXT
ABILITY value   value
WEIGHT  flagno
RANDOM  flagno
INPUT   stream
```

```
WHATO
CALL    address
PUTO    locno+
NOTDONE
AUTOP
AUTOT
MOVE    flagno
WINSIZE height  width
REDO
CENTRE
INKEY
EXIT    value
SKIP    offset8
RESTART
TAB     col
COPYOF  objno   flagno
COPYOO  objno   objno
COPYFO  flagno  objno
COPYFF  flagno  flagno
```

Notes:

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.
m) 'stream' is a valid window/stream 0-7
n) line,col,height & width are not range checked in the compiler as the interpreters clip to fit screen as required.
o) offset8 is a skip distance from -127 to 128, which specifies an entry to jump to. If a local label is specified then the distance is calculated auomatically.

5.2 The pre-processor commands

All preprocessor commands are preceeded by a hash or gate charac# ter ('#') except for LNK. They have to occupy a line of their own. Generally they can be placed almost anywhere, although it may not be appropriate or useful!

/LNK {w-s} filespec

A line starting with /LNK may appear anywhere in the source file and it forces the Compiler to switch to a different source file. Note that everything following the /LNK statement in the current file is ignored.

eg
/LNK srcfile2 will switch to SRCFILE2.SCE

/LNK x:\game\common\psi.inc will switch to PSI.INC on drive X

Compare this with #include which returns to the original file to continue compilation when the new file terminates.

#INCLUDE filespec

Will switch the compiler to use the specified source file. At the end of the file the compiler returns to the line after the #include in the original source file. Includes may be nested to a depth of 10. This allows included source files to include further files that they require and so on to a depth of 10.

#DEFINE symbol expression

Define the case sensitive label (maximum of 20 chars at the moment) to have the value given by expression. This can consist of other symbols, numbers and the addition (+) or subtraction (-) operators. The symbols can usually be used anywhere a number is required along with fixed numbers and the operators. This does not apply to IF statements which accept a single symbol only.

Note that as the compiler is single pass you must define all symbols before you use them. This does not apply to local symbols (those preceeded with a $) as detailed elsewhere.

Once a symbol has been defined an error will result if you attempt to define it again. If you need to do this use the #var command.

The compiler defines several symbols automatically. It assigns a non-zero (TRUE) value according to the command line options:-

From the 'm' machine option.

opt	symbol
0	PC

1 SPE
2 CBM64
3 CPC
4 MSX
5 ST
6 AMIGA
7 PCW

The following symbols are given a non zero value by machine options 8-12. Note that the relevant symbol from the above list is also non zero.

opt	symbol	machine symbol
8	VGA	PC
9	EGA	PC
10	CGA	PC
11	S48	SPE
12	SP3	SPE

The 'l' language option generates a non zero value for one of the following symbols:-

0 ENGLISH
1 SPANISH

The 'd' debug option sets as true (non-zero) the symbol 'DEBUG'.

If the 'm' option selected a machine which has PLOT,FILL and DRAW graphics - at the moment this is SPE,CBM64,CPC and MSX - the symbol 'DRAW' has a non-zero value.

C42,C40,C53 - true (non zero) if the 'm' machine features this number of columns on the screen. n.b. on ST and IBM it is possible the game is running in text only 80 column mode, so you may need to use the COLS symbol in calculations.

COLS - Actually contains the number of columns as a number. This can be used in compile time calculations (or run time if it is assined to a flag by LET flag COLS).

The files SYMBOLS.* can be included in your compilation. They use the #define directive to define many more symbols. The use is explained in those files.

#VAR symbol expression

This is similar to the #define command but it redefines a symbol which has already been defined. This has a use in counting the use of an item in a file or conditional compilation etc.

#IF {!}symbol
{#ELSE}
#ENDIF

This group of commands will occur together. Note that the ELSE is optional. Any lines following the #if (up until the next #else or

#endif) will be included in the compilation only if the symbol has a non zero (TRUE) value.

Additionaly any symbol can be rendered FALSE using an exclamation mark/pling ("!"). e.g.

```
#if !DRAW
```

will be true if on a machine which loads its graphics from DISC in a pixel based format!

The #else causes the lines between it and the next #endif to be compiled in the opposite case to the original #if test. This is not a duplicate of the function of the exclamation mark as you will often not need to do anything if the symbol is not true. In this case you would use the exclamation mark to negate the symbol rather than an empty #else#endif clause.

You cannot use an expression in place of the symbol. If you need to test for a group of conditions then use a #define to set up a temporary label. e.g.

```
#define COL53DISC DISC+COL53
#if COL53
 ...
#endif
```

will compile the lines between the #if and the #endif only if we are on a 53 column disc machine!

These conditional statements can be nested to a depth of 10, during the entire compilation.

All data inside #IF,#ELSE,#ENDIF must be indented by the correct number of spaces, one space for each nest level. This does not apply to files included using a #include within a conditional block. Any exception to this will cause a 'Fatal loss of conditional nesting sync' error.

There must be a corresponding #endif for every #if or some strange effects may occur, such as the compiler ignoring the remainder of the source file!

The following gives an example of the use of conditional compilation to deal with differences between machines and modes on that machine. Note the indentation at each #if level.

```
#if PC
 CLS
 PAPER 0
 #if EGA
  INK 15
 #else
  INK 7
 #endif
#endif
```

#ECHO

Output text to the console during compilation - usually to indicate titles and the inclusion of a file with conditional options.

e.g.

```
#if PC
 #echo Including IBM Display handler
 #include \LIB\PCDISP.SCE
#endif
```

#LOOKUP lang

Causes the compiler to search the specified language dictionary for the Verb & Noun of Process table entries. This allows you to have a process table written in one language to be compiled into a game written in another language. This must be combined with a multi-language vocabulary. See the section on multi-language games.

The valid values of 'lang' are 0 for English or '1' for Spanish. They may be assigned to symbols if required.

The following are several pre-processor commands designed mainly to deal with external data and code. Further explanations are given in the document on external code inclusion.

#INCBIN filespec

Will include a memory image file at the current position in the database. This alows just about any type of data to be included in the database segment.

#DEFB expression {{{expression} expression}...}

Will include the indicated byte value(s) in the database at the current address. For example:

```
#DEFB 1 2 3 4
#defb PSIFLAG-1 PSILAST-PSIFLAG
```

#DEFW expression {{{expression} expression}...}

Will include the indicated word value(s) in the database at the current address. For example:

```
#defw 6578
#defw IOADDR+4 65535
```

#defb/w commands may be used in preference to INCBIN if you need to include only a few values or ones that are calculated from symbol values.

#HEX hexpair{{w-s}{hexpair}...}

Will include direct hex data in the db. This is similar to the HEX statement of PDS. There must always be an even number of hex digits, even if you seperate elements with whitespace. For example:

```
#hex FFDE7898
#HEX 00 89 FD F3
```

#DBADDR symbol

Will give symbol the current address in the database. This can be used for CALL, #userptr or #defw commands.

#USERPTR n

Where n is from 0-9.
This command is designed to overcome the forward reference limitation of the single pass compiler. It places the current database address in one of ten vectors whose position is fixed at the start of the database. Thus an external routine can locate inserted data in the database by looking at the fixed vector.

#EXTERN {filespec}
#SFX {filespec}

For 8 bit only. These three commands are similar to #USERPTR, but the value stored by them is copied by the interpreter on database load to the corresponding EXTERN vector. Note that SFX already has a default effect of writing to the sound chip registers. So any routine using this vector will replace this function. The EXTERN.SCE file uses EXTERN to achieve the sound so that the SFX command continues to function.

Now any EXTERN or SFX commands will be directed to your routine.

The interpreters recognise 3 external vectors. The third is for a

50Hz interrupt:-

#INT {filespec}

For 8 bit machines only. Any routine placed on this vector will be called 50 times a second for the entire period that the game is running. Thus there is no command in the DAAD language to call the routine. The Z80 ones save only the AF and HL registers (as HL is given as your address), so be sure to save any other registers you use. The 6502 interpreters save the entire processor state.

If you include the filename in the #EXTERN, #SFX or #INT commands then it will cause the given file to be included in the database - as if the command was followed by #INCBIN.

Section 6 - The Compiler

The compiler allows the use of full pathnames for Source, Destination and Print filenames. Several options are added which are introduced with a minus ("-") sign. The command line format is...

```
DC infile outfile {-c|l?|s|m?|n|d|t|pprnfile} {-w|h|a?}
```

```
c - Apply compression to text (except objects)
l - Language, where ? is 0 or 1
  0 - English
  1 - Spanish (default)
m - Machine, where ? is 0 to 12
  number                symbol in source file
  0 - IBM PC(default)   PC
  1 - Spectrum          SPE
  2 - CBM 64            CBM64
  3 - CPC               CPC
  4 - MSX               MSX
  5 - ST                ST
  6 - Amiga             AMIGA
  7 - PCW               PCW
  ;These symbols allow special versions to be generated
  8 - PC VGA            VGA
  9 - PC EGA            EGA
 10 - PC CGA            CGA
 11 - Spectrum 48K      S48
 12 - Spectrum Plus 3   SP3
n - No output file
d - Include DEBUG sections
t - Write a token scan file called FINDTOK
p - Specify file for text output (except major errors)
s - Give symbol table
```

the second group of options can only be used with m0 and are for system use only. You should never need to use them unless instructed by Infinite Imaginations.

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)
```

Errors and Warnings

Compiler WARNINGS

Anything that is a warning will not stop the database being produced, but the interpreter may find some of the codes a little strange, or jump to the wrong entry for symbol warnings.

There are 10 types of warnings that can be issued:-

1) Token is longer than a previous token

   The tokens should really be in order of size

2) Token contains char >127 - Value(n)

   The Compiler has found a character with value n (128-255 decimal). These values are used for storing the tokens so cannot be used in a token.

3) ESCAPE code expected

   You have included a backslash ("\") in a token or piece of text and not followed it with a character. This will normally only occur at the end of the line. The \ is ignored.

4) ESCAPE code invalid

   The character following a backslash ("\") is not a valid token. See the source file syntax for a full description of legal ESCAPE codes. The code is ignored - this will usually mean the loss of a letter from the text.

5) 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.

6) 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.

7) Local symbol already defined: symbol

   You can only use a symbol once withing a process table. The sybols are cleared at the end of the table and can be reused in other tables.

8) Symbol already defined: symbol

   The global symbol table already contains an entry for this symbol. Be careful your compilation may need scrapping if the symbol is important. The first definition holds.

9) Symbol not defined: name

   A name that the compiler has assumed is a symbol is not defined in the global symbol table.

10) Invalid hex pair on line

    You must specify an even number if digits in #hex statements. The rest of the line will be ignored.

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.

In the case of errors that are marked as (fatal) the compiler will stop and print the error to the screen, even if a print file is active - this is not repeated in the print file so ensure you make a note of the number!

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.

This is also caused if there is at least one PROCESS table which is not called at the end of the database. i.e. it will not detect unused process tables only unused ones at the end of the database.

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) (fatal) Invalid File Name

The file name in the command line, /LNK statement or #include 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. We suggest that only '_' should be used anyway!

28) (fatal) Adventure Name not given

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

29) (fatal) Invalid drive x

An invalid drive was specified.

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 .DDB

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) Database not correct length - Disk may be full

The compiler has checked the actual length of the database on disk with the length it thinks it should be, if it is shorter then ther was probably 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!

55) Stream not found or too big

The WINDOW stream was not in the range 0-7.

56) Line not found or too big

57) Column not found or too big

58) Height not found or out of range

59) Width not found or out of range

60) *** Not used

61) Input option not found or out of range

62) Colour not found or out of range

63) Already processed 128 tokens

    You have more than 128 tokens in the /TOK section. This can occur if including a TOK file which contains an extra /TOK at the start.

64) Insufficient tokens found

    Opposite of 63!

65) Token is too long

    A token must be less than 9 characters.

66) Token is too short

    A token must be greater than 2 characters.

67) (fatal) Invalid option 'opt'

    The option given on the command line is invalid.

68) (fatal) Bad/Missing parameter for option 'opt'

    The parameter for the option is out of range or missing.

69) (fatal) Too many files specified

    You should only specify a maximum of two filenames on the command line (except for the PRN file which is specified using -p).

70) Invalid compiler command

    The preprocessor (#) command is not recognised.

71) (fatal) Symbol table full

Thats it folks, the symbol table is of a specific size.

72) (fatal) Maximum conditional nesting exceeded

You can only #if a maximum of 10 times in any file.

73) Imbalanced conditional

A #endif was found without a preceeding #if.

74) (fatal) Fatal loss of conditional nesting sync

You must include at least the same number of spaces in front of every line as the depth of the conditional block the are contained in.

75) Can't indirect this parameter

You have attempted to indirect the second parameter of a CondAct or the first one of the few that don't support indirection.

76) Missing bracket on indirection

The compiler thinks you have missed the closing bracket ']' on an indirected parameter.

77) (fatal) Maximum include nesting exceeded

You can only nest the #include statement upto 10 times.

78) Symbol not found, or invalid

The symbol was not found in the global symbol table.

79) Not enough user attribute indicators found

You must specify 16 user attributes (even if null) for every object in the game.

80) User attribute indicator invalid 'attrib'

The character must be 'Y' to set a user attribute.

81) Unresolved forward references on line(s):

You have done a SKIP to a local symbol which was subsequently not defined. Note that this error is given at the end of the process table to which it refers. Due to the operation of the compiler it is not feasible to print the line affected, only its line number.

82) (fatal) Forward reference data cache full, increase it

The internal memory area used for forward references has filled up. Try and reduce the nesting level of forward SKIPS, or if all else fails contact Infinite Imaginations to get the buffer size increased.

83) Maximum forward references, reduce nesting or quantity

Similar to the above but you have reached the limits. The same applies as to error 82.

84) Out of forward reference index space, increase it

See references to 82 and 83.

85) Backward SKIP out of range

The maximum reverse skip is -128 entries.

86) SKIP label illegal within an entry

A local ($) label can only occur between entries. This implies that it must come after the blank line which ends the previous entry. In fact the best place is on the line immediatly preceeding the next entry.

87) Forward SKIP(s) to this label out of range

The maximum forward skip is 127 entries. There is at least one SKIP to this label that is outr of range. The only easy remedy is to search forward in the file for references to the symbol and remove the first (or first few if they are obviously also out of range), as this is the furthest away!

88) (fatal) Internal fri error, patch code not SKIP

An illegal skip patch has occured. This will happen only if the internal compiler tables become corrupt, save the .SCE file and report it to Infinite Imaginations.

89) Language specifier not found or invalid

You have not specified a language for the #lookup command

90) Vector already defined

The user vector has already been allocated.

91) (fatal) Size exceeds cache space

The process table has become too big, try and reduce it by splitting into sub-process'.

92) (fatal) Error reading file

A disk error has occured during compilation.

93) Can't include binary data in this section

You have attempted a #incbin or one of the commands that include data in a compiler section which doesn't allow it - e.g. in /VOC

94) Invalid address

You have specified an address not in the nummeric range 0-65535.

95) Invalid user vector number

The user vector number is not in the range 0-9.

96) Can't open file FINDTOK.TXT for write

The compiler can't open the file 'FINDTOK.TXT' for writing. This might occur if the directory is full or the file exists with the 'read-only' attribute set.

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. If all else fails contact us!

Section 7 - Graphics

There are two groups of graphics editors:

1/ Drawstring

All versions are very similar in operation and will be discussed together.

2/ Image

These are more graphics/data managers as they do not include any way of editing graphics (or in the latest edition sounds) as such, but use images from an Art Package.

7.1 Drawstring Editors

We suggest that you should be familiar with the Spectrum PAW graphics option, CPC and CBM 64 Illustrator. The DAAD editors are very similar.

7.1.1 Main menu

A - Graphics	Allows graphics to be inserted, amended, printed etc.
B - Characters	Allows characters to be inserted, amended etc the characters double as shade patterns and as single characters for creating detail
C - Windows	Sets up the windows that the graphics will use when running in the final game.
D - Text Colours	Sets what actual Colours the 16 DAAD colours look like and the colour of the screen border - for the final game
E - Set SPARE	Sets up the first byte available for graphics in memory. When the compiler has compiled a game, it gives an end address for the database. Type this in on this option and the F option will then correctly maintain your free memory. Note that the compiler must be compiling for the same machine as you are setting it on! Note you can set this at any time. I.e. you can write the game and graphics seperatly.

F - Free Memory Once option E has been used will give an indication of the amount of free memory available.

G,H,I & J Fairly obvious functions described below:

Spectrum

SAVE Graphics
Equivalent to the BASIC command SAVE f CODE m,n where m is the start address of the database, n is the length and f is the filename.

VERIFY Graphics
Equivalent to the BASIC command VERIFY f CODE m,n

LOAD Graphics
Equivalent to the BASIC command LOAD f CODE, i.e. it will load any file of bytes back to the address it was saved from.

Very Important
If BREAK is pressed or a tape error detected during a load then the database held in memory would be corrupt. If CAPS SHIFT & 6 is pressed while a name is requested the graphic database will be unaffected.

Amstrad CPC

Disc/Tape
For those with both a disc drive and tape deck. Allows you to choose between them for Save, cat etc.

SAVE & LOAD Graphics
These options allow the graphic database to be saved or reloaded and in each case you will be prompted to "Type in name of file". When loading, the computer will search for a file of bytes with the name specified and then load it. A null filename may be used when you are using tape but not with disc. Care should be taken when using Load Graphics with a null filename because it will load any binary file it finds. LOAD will load any binary file back to the address it was saved from. e.g. you would use 'E' to load an SCDUMP routine.

Very Important
If ESC is pressed or an error detected during a load then the database held in memory would be corrupt, so a call is made to set a minimum database. This means your graphics will be lost, but you haven't corrupted the database and can use any option available. If ESC is pressed while a name is requested the graphic database will be unaffected.

CAT
Can be used to catalogue a tape or disc in the same way as from

BASIC.

Commodore 64

Save, Verify & Load Graphics

These options on the Main Menu allow the graphic database to be saved, verified or reloaded and in each case you will be prompted "Disc or Tape?" reply D or T as required and "Type in name of file". When loading, the computer will search for a file of bytes with the name specified and then load it.

Very important
If RUN STOP is pressed or an error detected during a load then the database held in memory would be corrupt, so a call is made to set a minimum database. Result: one blank location! This means your graphics & shade patterns will be lost but the database is not corrupt and can use any option available. If RUN STOP is pressed while a name or media is requested the graphic database will be unaffected.

7.1.2 Graphics Menu

Pictures may be inserted, amended, printed or have their length calculated:-

Insert **I**

The next available picture number is used and a null entry is made for it in the picture table. An entry of is also made for it in the location flag table. Processing then continues with an automatic call to the amend routine to allow the user to amend the null entry already set up in the picture table.

Amend **A picno.**

The graphic database is expanded to provide a gap at the end of the required picture. The main loop of the Graphic Editor described below is then entered. When return is pressed any gap still remaining is removed. n.b. the database itself is changed, thus you cannot abandon an edit.

Size **S**

The number of bytes between the start of the drawstring and the start of the next is calculated and printed on the screen.

Print **P picno.** or **L picno.**

The required picture is drawn on the screen and if L was selected the SCDUMP routine is called. picno. must be specified.

Points to note:

* There is a limit of 254 pictures.

Spectrum

* Locations which are subroutines have PAPER 1, INK 7 as start up colours.

Amstrad CPC

* Locations which are subroutines use INKS 1,24,18 & 26 as start up colours.

* Due to the variety of printers/plotters available the SCDUMP subroutine provided saves the screen to disc/tape with a filename of PICxxx. This file may be reloaded from BASIC and you may then use any of the published or commercially available screen dump routines to produce a hard copy.

  Alternately the address of the machine code SCDUMP subroutine is given on the start up screen. If you wish you can write your own printer driver and patch it in using the Load Graphics option on the Main Menu. The subroutine should end with a RET instruction, it should preserve all registers, it should not exceed lK bytes and obviously it should not 'mess up' the firmware.

Commodore 64

* Locations which are subroutines use PAPER 6 as a start up colour.

* SCDUMP is a routine which will copy the hires screen to the printer. You can if you wish change the routine to use a different printer by creating a file (max 1000 bytes) at the address given on the title screen, and loading it in using LOAD Graphics.

Dump **D**

Copies the picture specified to tape or disc as an 8K Hires screen (Saved from $8000 hex) followed by a 1K Colour screen (Sved from $CC00 hex) under the filename specified, but preceded by a B & C respectively - these could be used in your own programs, the programmers guide gives details of using a normal hires screen from BASIC.

Notes for ILLUSTRATOR/PAW users.

ALL versions
* Subroutines now can be rotated about either the X or Y axis to provide the mirror function.

* Shade cannot mix patterns, but can use any graphic characters upto the 256 maximum.

CBM & CPC
* Subroutines can now be nested to a depth of ten as on the Spectrum
* There is no Freehand option
* The CTRL & C (CTRL & T on CPC) allows single graphic characters to be placed on screen at the Base Cursor to add detail to pictures.

CBM
* The Help function on H is still available despite the Spectrum/CPC not having one!
* Do not draw 'off' the bottom of the screen, this will corrupt the bottom two text lines in the interpreter as it does not use a split screen like the editor.

7.1.3 Characters Menu

This option allows the standard 256 characters (described elsewhere) to be amended, saved loaded etc.

The Save and Load options allow an arbitrary size collection of characters o be manipulated. E.g. You could load a standard PAW character set merely by doing L 0, to load to character 0. This facility as well as allowing useful characters to be transferred between databases can also be used to move them around in the set. E.g. S 10 14 followed by a L 20 would, via the disc, move characters 10,11,12,13 & 14 to be characters 20,21,22,23 and 24!

The character editor

This section of the editor allows a shade pattern/character to be amended. Shade patterns consist of an 8 by 8 pixel grid (8 by 4 on CPC) which is repeated by the SHADE command over the entire area of shade. Characters can be placed on the screen from within the drawstring. They are obviously displayed by any text output in a game.

The editor provides three areas of screen:-

1/ The Grid — The bit patterns of a particular shade are shown on a much enlarged (x8) grid of grey squares, any set bits being shown by a black square. Also present on the grid is a flashing red square showing the current cursor position.

2/ The Test Pattern — To the right of the Grid is a test pattern of the current shade in both normal & Inverted forms. (Normal on top). This does not change as you change the Grid but it can be updated by presssing

F7.

3/ Status Area — This shows the current pattern number and gives a summary of the commands available.

Commodore 64

In order to modify the pattern use the cursor keys (marked CRSR) along with SHIFT as necessary to move the flashing cursor. The state of the bit under the cursor can be changed at any time using the SPACE BAR.

RETURN will store the amended pattern back in the database.
RUN STOP will abandon the edit leaving the pattern as it was.

Amstrad CPC

Unless you understand the method the CPC uses to code four colour mode characters then only use characters you have edited in shade mode for the SHADE command.

7.1.4 Window Menu

Can be used in one of three ways:

A picno

Will set the given picture to be a subroutine - I.e. not having a picture

A picno paper ink

Will give picture picno the given paper and ink to start drawing with.

A picno paper ink line column height width

Will give picture picno the given paper and ink and define an area of the screen which will be cleared by the interpreter when drawing this picture. Note that the Grid option when editing a graphic will highlight only this region.

Spectrum

On the Spectrum this area may not be what you expect as the windows are set in 40 column mode, so the grid shows the closest fit on the 32 column attributes. You could consider this useful as it shows what actual area of the screen is occupied by the 40 column window. If you draw outside of the window, it will still show up during the game it will just be outside the window! I.e. DAAD does not 'clip' its graphics. See the note under CBM as well for a similar problem.

7.1.5 Text Colours

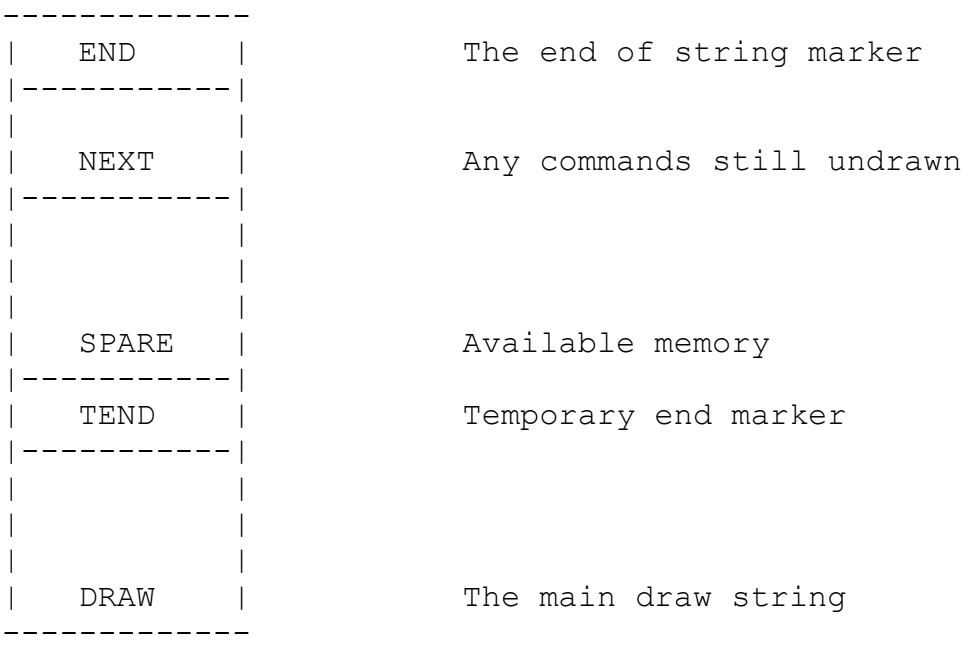
This option sets what actual machine colours the interpreter will use when it uses the INK and PAPER CondActs. This is so that the writer of the database does not need to worry about the different colour numbers on various machines. I.e. In the versions you have so far DAAD colour 0 is always BLACK, colour 1 is always WHITE and colour 2 is always RED. Despite the fact that white and red are different colour numbers on the CBM and SPECTRUM.

Amstrad CPC

* The text colour option on the CPC sets the initial Pallette for the start of the game. Any part of the pallette can be selectivly changed for each picture drawn at a location. But you are stuck with the current pallette for those drawn using PICTURE. This is no strain as there are only four colours available, so character faces etc should use the fixed bit of the pallette used by text, or the look will change! See the note on ST/Amiga pallette management.

7.1.6 Basic Operation

When editing, the string is laid out in memory as follows;



It is important when using the editor to imagine the above model. You will find otherwise that it is easy to step back through the drawstring leaving undrawn commands in the NEXT section. These can get 'out of context' and when drawn either on re-editing or using Next commands may cause a screen error.

7.1.7 Spectrum Graphic Editor

A rubber banded line is used for drawing; the base point of the line (Known as 'point') shows the last point plotted, moved to etc, the rubber banded end of the line shows the next position of point or the start point for a fill/shade etc.

The Editor provides four groups of commands. Any which insert a command into the drawstring require the SYMBOL SHIFT key to be held down;

1) Drawing Commands

ABS MOVE A — Moves point to the x,y position of end of the line setting only the attributes. This is coded as a PLOT with Inverse and Over set on.

PLOT P — Sets the pixel at the end of the line according to Inverse and Over, then moves point to that position.

REL MOVE R — Moves point to the end of the line without affecting the screen. This is coded as a relative offset from the old point.

LINE L — Draws (or fixes) a straight line from point to the end of line according to Inverse and Over, then moves point to end of the line. The line is coded as a relative offset from the old point.

FILL F — The area from the end of line (relative) is filled using solid pixels. Fill works by passing a pattern to the SHADE routine so the notes on SHADE apply also

All the above use 3 bytes in the database.

SHADE S — The area from the end of line (relative) is shaded with one of a large number of patterns. The database contains 256 characters, which can be changed using the Character Editor.

The pattern used for shading is determined as follows:-
a) You are asked for a pattern number in the range 0 to 255.
b) If INVERSE was 'on' the resultant pattern is inverted, i.e. SET/RESET pixels are swapped.

Note 1. The shade first works in a downward direction and then in an upward direction. For speed, when it is going down it doesn't look up and vice versa. Any areas the shade

misses must be shaded separately, although careful choice of the start position for the shade will minimise this.

Note 2. If the area to be shaded is too complex then the shade will be abandoned. It has to do this to enable it to detect when it comes across an area which has already been shaded. Thus an area can only be shaded once as an already shaded area will be too complex to shade again. You should not shade an area and then try to fill in the background with a fill command, use the Inverse option!

TEXT T A character code in the range 0 to 255 is requested. This character is placed on screen in the character square (as shown by the Grid command) that the tip of line is contained within. It is mainly designed to allow "the fiddly bits" of as picture to be drawn using the character editor - thus using less memory than lots of lines!

Text and Shade use 4 bytes in the database each.

BLOCK B Causes a block of the currently selected colours to fill the rectangle of attribute squares which the line defines the diagonal of.

Block uses 5 bytes in the database.

2) Colour Commands

INK X The current ink is set to the value selected. INK 8 as in BASIC causes all ink to be taken from the existing screen attributes.

PAPER C Sets the current paper to the value selected. PAPER 8 as in BASIC.

FLASH V The new value of Flash is requested (0,1 or 8).

BRIGHT Z The new value of Bright is requested (0,1 or 8).

all the above use one byte in the database.

INVERSE I The state of Inverse (on/off) is toggled.

OVER O The state of Over (on/off) is toggled.

Neither Inverse nor Over use any memory but their state is encoded as part of each future instruction which is affected by them.

3) Subroutine Command

GOSUB G A picture number is requested which must be in the

range 0 to locno. A scale value for the picture is then requested. This can be from 0 to 7 where the number indicates the size of the picture in eighths - 0 means 'no scale' (i.e. 8/8).

You are next asked if you wish to rotate the design about wither the X, the Y or both axis.

Please Note;

a) Scale only affects certain commands, these are MOVE RELATIVE,LINE,FILL and SHADE. MOVE ABSOLUTE, PLOT, BLOCK and TEXT commands will not be scaled or relocated and should generally not be used in subroutines (although they will work and can be used usefully sometimes).

b) You may only nest subroutine calls to a level of ten. (nesting means calling a subroutine from within a subroutine).

c) Scale does not affect GOSUB commands, i.e. if a GOSUB is used within a subroutine the string drawn will be at a fixed size and not scaled.

d) Calling the same routine you are drawing will cause a "Limit Reached" error as the limit of 10 subroutine levels will have been reached.

Gosub uses two bytes in the database.

4) Editing commands

START — Puts the Drawstring pointer at the start of the drawstring.

NEXT — Executes next available drawstring command: if there isn't one the command is ignored.

PREVIOUS — Moves the drawstring pointer back one command and updates the screen.

DELETE — (CAPS SHIFT & 0 on 48K) deletes the previous command in the drawstring and updates the screen.

DELETE NEXT — GRAPH(ICS) (CAPS SHIFT & 9 on 48K) deletes the next command if there is one.

GRID Y — Has a toggle action for a character grid of INK 0, PAPER 7 and PAPER 6. This allows exact positions of colour boundaries to be taken into account while drawing.

JOYSTICK J Toggles the Kempston~ joystick option on and off.

The keys around S move the end of line around one pixel at a time (this can be accelerated to eight pixels a time by holding down the CAPS SHIFT key) thus:

Q	W	E
A		D
Z	X	C

The joystick should be plugged into Port 2 of Interface 2 or the Spectrum Plus 2. Alternatively it can be plugged into a Kempston~ interface in which case SYMBOL SHIFT & J should be used to enable DAAD to read it. CAPS SHIFT will also accelerate the rate of movement on the joystick. The Fire button will act like SYMBOL SHIFT and L to draw a line.

Spectrum Editor Error Messages and their meanings

BREAK	BREAK was pressed during a peripheral operation.
STOP in INPUT	CAPS SHIFT & 6 pressed.
Tape loading error	as BASIC. Note that a tape error during a load database means that the database is corrupt and only reloads should be attempted.
Database full	There is not enough room in the database for what you were attempting.
Limit reached	The maximum number of graphics, or pages are already present. Alternatively the maximum subroutine depth has been reached - 10.
Integer out of range	While drawing a picture a LINE command has gone out of range. This is usually due to a change of position of the starting point while editing.
Out of memory	This occurs when attempting to start a new RAM page on a 48K spectrum, loading a 128K game on a 48K spectrum or if

insufficient workspace has been left e.g. by setting up fixed channels before loading or by writing too big a program for use with EXTERN.

Note: After an error during editing of a drawstring; the Drawstring pointer is positioned just before the command which caused the error (i.e. a NEXT command will cause the error again). If you are unable to correct the problem then DELETE NEXT can be used to delete the erroneous command. Note that this may still leave further commands undrawn (which may cause another error).

7.1.8 Amstrad Graphic Editor

A two cursor system is used for editing; the Base cursor shows the last point plotted, moved to etc, the Rubber cursor shows the next position of the Base cursor or the point for a fill.

The Editor provides four groups of commands;

1) Drawing Commands

PLOT P Sets the pixel at the position of the Rubber Cursor (RC) according to the current Pen then moves the Base Cursor (BC) to that position. The position plotted is an absolute position and only positions on the visible screen can be plotted.

MOVE M Moves BC to RC without affecting the screen. This is coded as a relative offset from BC.

MOVEA A Moves BC to RC without affecting the screen. The position moved to is an absolute position and must be on the visible screen.

LINE L Draws a straight line from BC to RC according to the current Pen, then moves BC to RC. The line is coded as a relative offset from BC.

FILL F The area below RC (relative) is filled using the current pen. Fill works by passing a pattern to the SHADE routine so the notes on SHADE apply also to FILL.

All the above use 3 bytes in the database. Relative distances are limited to #+# 1022 in the X direction and #+# 510 in the Y direction.

SHADE S The area below RC (relative) is shaded with one of a large number of patterns.

The pattern used for shading is determined as follows:-

a) You are asked for a pattern number in the range 0 to 255. This corresponds to a character.

b) Any Ink 3 in the resultant pattern is changed to the current Pen. (If the current Pen is Pen 3 then this has no effect).

c) If SHIFT S was pressed you are asked for a colour number and the resultant pattern has background (INK 0) pixels set to that colour.

Notes on the shade (and fill) routine:-

1) The shade first works in a downward direction and then in an upward direction. For speed, when it is going down it doesn't look up and vice versa. Any areas the shade misses must be shaded seperately although careful choice of the start position for the shade will minimise this problem.

2) The area to be shaded is defined by the ink of the start pixel. Any other Inks or the edge of the visible screen mark the edge of the area to be shaded.

3) The start point of the shade must be on the visible screen.

4) If the area to be shaded is 'too complex' then the shade will be abandoned. It has to do this to enable it to detect when it comes across an area which has already been shaded. Thus an area can only be shaded once as an already shaded area will be 'too complex' to shade again. You should not shade an area and then try to fill in the background with a fill command - Use SHIFT & S to select the correct paper.

5) N.B. The firmware line drawing routine on the 664 is slightly different to that on the 464 such that a line draw from A to B may not plot exactly the same pixels on the two machines. This is unlikely to cause problems but when selecting a start point for a shade you are advised not to go immediately next to any lines already on the screen.

TEXT T A character code in the range 0 to 255 is requested. This character is placed on screen in the position given by the RC cursor.
It is mainly designed to allow "the fiddly bits" of as picture to be drawn using the character editor - thus using less memory than lots of lines!

Text and Shade use 4 bytes in the database.

BLOCK B The rectangle defined by BC & RC (Absolute) is completely filled using the current Pen. The left hand edge will be rounded down to start on a pixel

which is divisble by 8. The right hand edge will then be rounded up so that the width of the rectangle is a multiple of 8 pixels. This command uses the firmware routine SCR FLOOD BOX so it is faster than FILL but much less flexible.

Block uses 5 bytes in the database.

2) Colour Commands

PEN CTRL/0-3 Changes to the Pen specified.

Pen uses 1 byte in the database.

3) Subroutine Command

GOSUB G A picture number is requested which must be in the range 0 to max. picno. A scale value for the picture is then requested. This can be from 0 to 7 where the number indicates the size of the picture in eighths - 0 means 'no scale' (i.e.8/8).

You are next asked if you wish to rotate the design about wither the X, the Y or both axis.

Please Note;

a) Scale only affects the relative commands, these are MOVE,LINE,FILL and SHADE. The other commands will not be scaled or relocated and should generally not be used in subroutines (although they will work and can be used usefully sometimes)

b) Scale works by multiplying the relative distance by the scale value, dividing by 8 and rounding down. Thus only relative distances which are multiples of 8 will be scaled precisely.

c) Subroutines can be nested to a depth of 10. Exceeding this will cause a 'Limit reached' error. If this happens, pressing any key will redraw the picture to just before the GOSUB, then the CLR key (which deletes the next command) can be used to delete the erroneous GOSUB.

4/ Editing commands

START Puts the Drawstring pointer at the start of the drawstring.

NEXT Executes next available drawstring command: if there isn't one the command is ignored.

PREVIOUS		Moves the drawstring pointer back one command and updates the screen.
DELETE	DEL	The previous command is deleted and the screen redrawn.
DELN	CLR	Deletes the next command if there is one.

Amstrad CPC Editor Error Messages and their meanings

BREAK	ESC was pressed during a peripheral operation or while editing, or a disc error occured.
I/O Error	An I/O error has occured. Note that an error during a load Graphics will set up a null database.
Database full	There is not enough room in the database for what you were attempting.
Limit reached	The maximum number of pictures is already present or a nested GOSUB has been found.

Note: After an error during editing, the Drawstring pointer is positioned just before the command which caused the error (i.e. a NEXT ( ) command will cause the error again). If printing then a return is made to a menu.

7.1.9 Commodore Graphic Editor

A two cursor system is used for editing; the Base cursor shows the last point plotted, moved to etc, the Rubber cursor shows the next position of the Base cursor or the point for a fill.

The Editor provides four groups of commands;

1) Drawing Commands (active if CTRL held down)

PLOT	P	Sets the pixel at the position of the Rubber Cursor (RC) according to the current state of Inverse/Over (see later) then moves the Base Cursor (BC) to that position. The position plotted is an absolute position and only positions on the visible screen can be plotted.
REL MOVE	R	Moves BC to RC without affecting the screen. This is coded as a relative offset from BC.
ABS MOVE	A	Moves BC to RC without affecting the screen. The

position moved to is an absolute position and must be on the visible screen.

LINE L Draws a straight line from BC to RC according to the current Pen, then moves BC to RC. The line is coded as a relative offset from BC. (n.b. final point of line is not set).

FILL F The area around RC (relative) is filled using solid pixels. Fill works by passing a pattern to the SHADE routine so the notes on SHADE apply also

All the above use 3 bytes in the database. Relative distances are limited to #+#511 in the X direction and #+#256 in the Y direction.

SHADE S The area around RC (relative) is shaded with one of a large number of patterns.

The pattern used for shading is determined as follows:-
a) You are asked for a pattern number in the range 0 to 255. This corresponds to a character.
b) If INVERSE was 'on' the resultant pattern is inverted, ie SET/RESET pixels are swapped.

Notes on the shade (and fill) routine:-

1) The shade first works in a downward direction and then in an upward direction. For speed, when it is going down it doesn't look up and vice versa. Any areas the shade misses must be shaded seperately, although careful choice of the start position for the shade will minimise this problem.

2) The start point of the shade must be on the visible screen. (Although it might be below the screenbreak and thus invisible! This is not advised as the screen break is not in the same place in the interpreter)

3) If the area to be shaded is 'too complex' then the shade will be abandoned. It has to do this to enable it to detect when it comes across an area which has already been shaded. Thus an area can only be shaded once as an already shaded area will be 'too complex' to shade again. You should not shade an area and then try to fill in the background with a fill command.

CHAR C A character code in the range 0 to 255 is requested. This character is placed on screen in the character square (as shown by the Grid command) that the RC cursor is contained within. It is mainly designed to allow "the fiddly bits" of as picture to be drawn using the character

editor - thus using less memory than lots of lines!

Character and Shade use 4 bytes in the database.

BLOCK B The rectangle defined by BC & RC (Absolute) is completely filled using the current Colours. This command is faster than FILL but much less flexible. (Note it does not set any pixels only colours).

Block uses 5 bytes in the database.

2) Colour Commands

PAPER CTRL & 0 Allows a paper from 0 to 16 to be selected.

INK CTRL & I Allows an ink from 0 to 16 to be selected.

Both INK & PAPER use one byte in the database. Note that a value of 16 is not a colour as such but can be thought of as transparent, ie. It causes the existing screen colours to show through.

INK and PAPER each use 1 byte in the database.

INVERSE F3 Inverse is selected which causes the inverse bit of any future FREEHAND, PLOT LINE & SHADE commands to be set. This causes all pixels to be reset instead of set in the case of FREEHAND, PLOT & LINE and the pattern to be Inverted in the case of SHADE. SHIFT & F3 will reset the Inverse flag.

OVER F5 Over is selected which causes the over bit of any future FREEHAND, PLOT & LINE commands to be set. This causes any set pixels to be reset and any reset pixels to be set (i.e. the new state of a pixel is the EXCLUSIVE OR of its previous state).

Neither Inverse nor Over use any memory as their state is encoded as part of each instruction.

3) Subroutine Command

GOSUB G A picture number is requested which must be in the range 0 to max. picno. A scale value for the picture is then requested. This can be from 0 to 7 where the number indicates the size of the picture in eighths - 0 means 'no scale' (i.e.8/8).

You are next asked if you wish to rotate the design about wither the X, the Y or both axis.

Please Note;

a) Scale only affects the relative commands, these are REL MOVE,LINE,FILL and SHADE. The other commands will not be scaled or relocated and should generally not be used in subroutines (although they will work and can be used usefully sometimes)

b) Scale works by multiplying the relative distance by the scale value, dividing by 8 and rounding down. Thus only relative distances which are multiples of 8 will be scaled precisely.

c) Subroutines can be nested to a maximum of 10. Exceeding this number will cause a 'Limit reached' error. If this happens, pressing any key will redraw the picture to just before the GOSUB, then the CLR key (ie SHIFT & HOME which deletes the next command) can be used to delete the erroneous GOSUB.

4/ Editing commands

START — (SHIFT & CRSR DOWN) Puts the Drawstring pointer at the start of the drawstring.

NEXT — (CRSR RIGHT) Executes next available drawstring command: if there isn't one the command is ignored.

PREVIOUS — (SHIFT & CRSR RIGHT) Moves the drawstring pointer back one command and updates the screen.

DELETE INST — (SHIFT & DEL) The previous command is deleted and the screen redrawn.

DELN CLR — (SHIFT & HOME) Deletes the next command if there is one.

GRID F7 — Overlays a grid on the drawing allowing the colour boundaries to be seen. As only one Paper & one Ink colour can be in each 8 x 8 pixel square some very clever positioning is needed to prevent colour clashes.
SHIFT & F7 will remove the grid (redraws screen).

Commodore Editor Error Messages and their meanings

RUN STOP	RUN STOP was pressed during a peripheral operation or while editing.
I/O Error	An I/O error has occured. Note that an error during a load Graphics will set up a null database.
Database full	There is not enough room in the database for what you were attempting.
Limit reached	The maximum number of pictures is already present or a nested GOSUB has been found.

Note: After an error during editing, the Drawstring pointer is positioned just before the command which caused the error (i.e. a NEXT (CRSR RIGHT) command will cause the error again). If printing then a return is made to a menu.

NB: The Non-Drawstring Data Managers (DMG) for PC and Atari ST are dealt with in a separate document.

Appendix A - The character set

Throughout the whole DAAD system the same character set of 256 characters is used. The codes are as follows:

0-15 are internal codes and cannot be used in text. Thus they are available only for shade patterns and graphic characters from the graphic editors.

16-31 are the special language specific codes. These change according to the language option. They mean nothing in the English interpreter. In Spanish:

Symbol	IBM code	DAAD code	Escape
¦	166	16	A
	173	17	B
¨	168	18	C
®	174	19	D
¯	175	20	E
	160	21	F
¸	130	22	G
¡	161	23	H
¢	162	24	I
£	163	25	J
¤	164	26	K
¥	165	27	L
‡	135	28	M
€	128	29	N
O	129	30	O
š	154	31	P

32-126 are the standard ASCII set

127 is used as a forced space code.

128-255 are whatever you want. Using the \g escape they can become a second set, which take on the same codes as 0-127. On the IBM in text mode only the ASCII uppercase 'A' to 'Z' will appear, all others are suppressed.