DAAD Ready v0.3

This document uses parts from the DAAD v2 Manual (C) Infinite Imaginations 1988/89/91, which was written by Tim Gilberts, compiled back to modern format by Pedro Fernandez from Andres Samudio's disks, and had some amendments appointed or made by a few people more, and very specially by Stefan Vogt. This manual has been expanded, modified, cut, and cropped by Uto.

PLEASE NOTICE THIS IS NOT THE DAAD MANUAL, BUT AN ADAPTED VERSION MADE FOR DAAD READY PACKAGE.

READ THIS MANUAL TO USE DAAD READY, YOU WON'T NEED TO READ DAAD ORIGINAL MANUAL, BUT ONCE YOU KNOW WELL HOW DAAD READY WORKS, MAYBE YOU WANT TO HAVE A LOOK AT IT.

TABLA OF CONTENTS

Overview

DAAD Ready makes easy for developers to have their game running in emulators, without having to care about building disks, transferring files, and may other options that required a lot of manual work both in the current emulator world, and in the original real machines one.

With *DAAD Ready* you will just have to concentrate in creating your game, cause building a .DSK, .D64 or whatever other format required for your target platform, is as easy as double clicking the .BAT file for your target (i.e. C64.BAT to generate the .D64 file for Commodore 64).

Due to all this, in this manual you may find references to DAAD itself, the original tool, and *DAAD Ready,* the package you have in your hands, which includes DRC (the new compiler, by Uto), original DAAD interpreters (by Infinite Imaginations), Maluva Extension (to use disk based graphics instead of vector ones, by Uto), the new MSX2 interpreter (by NatyPC), the new Plus/4 interpreter (by Imre Szell, based on the C64 one) and several tools for testing and packaging (emulators and other tools). The use of all these is transparent to the writer/programmer, that, as said above, just must click a BAT file to get the game running in the included emulator for each target.

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 Ready can create games for:

- **PC/DOS** (VGA 16 colours)
- **Sinclair Spectrum 48k** (tape, no graphics)
- **Sinclair ZX Spectrum Next** (SD Card)
- **ZX-Uno** (SD card)
- **ZX Spectrum +3** (floppy disk).
- **Amstrad CPC** (floppy disk, so either 664 or 6128). Also, M4 interface works.
- **Commodore 64 or 128** (floppy disk)
- **Commodore Plus/4** (floppy disk)
- **MSX 1** with at least 64K of RAM
- **MSX 2** computers
- **Amstrad PCW** 8000/9000 Series

Please notice *DAAD Ready* cannot generate the Amiga or Atari ST disks for you, despite DAAD itself supports Amiga and ST. That happens because we were not able to find tools for Windows able to handle disk images without human intervention. Thus, DAAD Ready cannot make Amiga or ST games. It would not be difficult for you though, to create Amiga or ST games once you have a little experience.

DAAD uses a programming language designed especially for writing Adventure Games. It is loosely based on the QUILL and PAW systems (by Gilsoft), and a familiarity with their operation is useful. There is a section further on this manual where you can find some help if you are/were a Quill or PAW author.

DAAD Ready can be divided into four main functional area:

The Source

The source (.DSF files, DAAD Source File) are a collection of tables, which contain all the information to define an adventure game. That includes location descriptions, objects description & weight, and the game logic.

The .DSF files are in the format of an ASCII Text file using ISO 8859-1 encoding. Please bear this in mind cause many modern editors tend to use UTF-8 by default, and that can have side effects if you are creating a non-English game with special non English characters (i.e. “ñ”, “á”, “è”, etc.). For English, you do not really care about that.

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. *DAAD Ready* includes DRC as compiler, DRC is a new compiler made by Uto in late 10s decade. You don’t really have to care too much about the compiler, cause DAAD Ready takes care of using it for you, but you can find your game does not compile well if you make some mistakes when changing DSF files. For those cases, you will see compile errors that you would have to fix. In case you are not sure, reverse latest changes in DSF file, and try again.

The Interpreters

These are the real heart of DAAD. There is an interpreter for each computer supported on the DAAD system. It runs the game using its collection of routines to carry out the tasks needed by adventure games. It provides the machine independent aspect of DAAD. *DAAD Ready*

also takes care of the interpreters, and creates a .DSK file, .D64 , etc. file for each target machine, which already includes the interpreter, so you don't really have to care about choosing the right interpreter, *DAAD Ready* does it for you.

The Graphics

Unlike original DAAD, DAAD Ready is not using vector graphics for location images, it is using raster graphics in lieu. To add raster graphics *DAAD Ready* is using Maluva extension by Uto. We will provide more details about this later in this manual.

Help

Should you need help understanding this or other DAAD topics, you may join the following groups in Telegram:

https://t.me/Advent8bit (English)
https://t.me/RetroAventuras (Spanish)

Languages supported

Despite DAAD does only support English and Spanish, some limited support for Portuguese and German has been added to *DAAD Ready*. You can skip this section if you are creating Spanish or English games.

Limited means that although you will be able to create games which use the special characters of those languages as "ß" or "õ", the interpreter below will always be the English one (for German) and the Spanish one (for Portuguese), so player won't be able to provide orders with those characters, and parsing will be made thinking the language is English/Spanish. That also influences object listing:

- If an object description starts with "a" or "some" in English games, when added to a message like "You take the _." that underscore will be replaced by the object text without the article. That way an object named "a lamp" will be display "You take the lamp." when taken, instead of "You take the a lamp."
- If an object description stars with "una", "un", "una", "unos" in a Spanish game, that definite article would be replaced by the indefinite one, so "Tomas _." would work, so "una lámpara" will show "Tomas la lámpara." when taken.

Sadly, those are very language specific behaviors, so for German and Portuguese articles are neither removed nor modified, as those languages articles do not match the removal/replacement rules. Thus, a reasonable approach has been taken for each language: in Portuguese, the message will use the indefinite article, and in German, messages like those will not show the object text, and will just say "Taken.".

Other languages may be used with DAAD, in a similar way than Portuguese or German are. Please join some of the Telegram groups above if you need help using DAAD Ready with your own language. Please notice these are the special characters supported by DAAD, other than English ones:

ª ¡ ¿ á é í ó ú ñ Ñ ç Ç ü Ü à ã â ä è ê ë ì î ï ò ô ö õ ù û Ý
Á É Í Ó Ú Â Ê Î Ô Û À È Ì Ò Ù Ä Ë Ï Ö Ü ý Ý þ Þ å Å ð Ð ø Ø ß

If your language uses special symbols other than these, still it can be done, provided it still uses roman alphabet in general.

Graphics

Overview

As stated above, *DAAD Ready* uses raster graphics for adventures, rather than vector graphics, and it uses Maluva extension for it.

To add graphics, of course you would need different formats for each machine, as they differ a lot.

DAAD Ready is only handling graphics for 8-bit machines (except PCW), for adding graphics to 16-bit machines and PCW please refer to original DAAD Manual.

To add graphics for your adventures, you must place images in the IMAGES subfolder in the DAAD Ready folder, except for CPC games, that you must add the images in the IMAGES/CPC folder.

All the images you will have to add will be full screen images, what means 256x192 for Spectrum or MSX1, 160x200 for C64 (HiColor), 320x200 for Plus/4, etc. On the other hand, *DAAD Ready* will only use first 96 scanlines of every picture, so whatever is below that will be ignored. So, the rule is: you create full screen images for each target you want to have pictures, and *DAAD Ready* will show first 96-pixel rows per each image.

To add graphics, you will have to place images in specific formats in the mentioned folders, those formats are:

- Spectrum SCR format for Spectrum +3 or ESXDOS
- 256 color indexed PCX format for Spectrum Next
- MLT or MLT+PAL file for ZX-Uno
- Spectrum TAP version does not support graphics in DAAD Ready
- Amstrad SCR + PAL format, mode 0, for Amstrad CPC
- Koala (.KLA) format for C64 HiColor or ART for HiRes
- PRG format for Commodore Plus/4
- SC2 format for MSX 1
- SC8 format for MSX 2

All these formats are very common and easy to create for each platform.

For graphics to work in DAAD Ready they should be named after the location where they should be shown, using three digits. So, if you want a graphic to be the one for location 12, you should put a file named 012.KLA, or 012.PRG, or 012.SCR, etc. depending on the target, in the required folder.

Also, you can add graphics which are not related to locations, using any number of a non-existing location, for instance you can add a graphic file 200.KLA, that won't be shown automatically because there is not a location #200, but can be loaded on demand by using the XPICTURE condAct (See Maluva CondActs section below).

Please notice DAAD Ready is a simple tool to help you start coding with DAAD and don't worry about many things, that is why pictures in *DAAD Ready* games are always placed on top of the screen and are 96 pixels high. It is possible to change that of course, but that is out of *DAAD Ready* scope.

Recommended tools

Many of the image formats required for each platform are made easily with emulation tools, this is a list of recommended tools:

Spectrum .SCR 256x192	This is exported by almost all emulator. ZX Paintbtush can handle them too.
Spectrum Next .PCX 256x192	This is easy to generate with Gimp, just make sure the palette is set to indexed and you picture has no less than 17 colors, cause otherwise Gimp will generate an invalid 16-color PCX.
ZX-Uno MLT/PAL 256x192	MLT files can be created with ZX-Paintbrush or Image2Ulaplus. If the MLT is 12,288 bytes long, the palette information is not in there, please also add a tap file with same name with the palette information (ZX-Paintbrush can export it).
CPC .SCR and .PAL 320x200	CPCImgConv is a good tool to get images finished. It is important to have in mind color 0 of the image would be used as background for all the screen, and color 1 for text, so either you handle palette so 0 is black and 1 is white, or you

	find a way to make text readable by ensuring colours 0 and 1 have a decent contrast. Remember that you have to export both the .SCR and the .PAL file for each location, and put them in the folder.
C64 .KLA 160x200 and .ART 320x200	Multipaint is a very good tool for C64 Multicolor images creation.
Plus/4 320x200	The images should be in Boticceli HiRes format and named 001.PRG, 002.PRG, 003.PRG, etc. Botticeli is a Plus/4 application
MSX 1 .SC2 256x192	There is a nice converter at https://msx.jannone.org/conv/. Make sure you use standard MSX 1 palette.
MSX 2 .SC8 256x212	Same converter than MSX1 but use SC8 format.

Please notice original DAAD interpreter used HiRes mode for C64 and Mode 1 for CPC. Those are not the best modes for C64 and CPC when you think about the pictures, although are the best ones that suit both text and pictures. Maluva extension used by *DAAD Ready* allows you to use what is called a "split mode", where the upper part of the screen is in one mode and the lower one uses another one. That allows C64 games to user HiColor graphics in top of the screen, while still using HiRes mode at the bottom, so the text is readable. Same goes for CPC, that uses Mode 0 on top and Mode 1 at the bottom.

Sadly, keeping those modes running like this requires complicated interrupt routines that are hard to keep on running while reading from disk. That's why you would see images or the whole screen is hidden when loading graphics, and you may see some flickering when loading text messages when you use the XMESSAGE o XMES condActs (see Maluva Condacts).

If you do not like that to happen, you can always use traditional modes, and have all the game in HiRes (C64) or mode 1 (CPC). To do that just make sure you edit CONGIF.BAT and change the line showing SET SPLITSCR=SPLIT to SET SPLITSCR=NOSPLIT. Then make sure you use the proper images (Mode 1 images for CPC, and ART files instead of KLA for C64).

How to use DAAD Ready

Here is a summary of the major points needed to produce a game:

To begin with, click on any of the BAT files (ZXPLUS3.BAT for instance). You will be asked which language the game will be using, then an empty game will be compiled, and you would be asked to press enter to continue, do it, and then just close the emulator launched after you press any key.

Well, once you have run any of the BAT files, you will find a TEST.DSF file in your *DAAD Ready* folder. That one is your source file. Start editing your source file with your preferred text editor. Make sure encoding is correct (ISO 8859-1) unless you are making an English game (if so, encoding does not matter).

You can make changes in the source file by knowing how to develop in DAAD language/format, check further into this document.

Each BAT file run compiles the game for the specified target (i.e. C64.BAT for Commodore 64) and tests the game. Please notice there will be a pause before launching the emulator, you could press CTRL+C in order to cancel emulation being launched if you just wanted to know if your changes were ok, but not testing.

Also, notice in that pause, sometimes you get instructions about what to do when emulators are launched, cause sometimes you must do something after the emulator runs (game does not start automatically).

Finally, you should know the game you are testing it is stored at the RELEASE folder, where you can find a folder per target. You could find the D64, DSK or whatever file in there once you want to release the game.

Other details

- Once you have selected a language your DAAD Ready setup will consider the game you are making is in that language. In case you want to create a different game at the same time, just unzip the DAAD Ready zip file in some other folder, and work with that game there. I know, is like using a sledgehammer to crack a nut, but for the time being, that is the way to do it.
- Visual Studio Code is a recommended editor as it has an addon made by Chris Ainsley which provides syntax highlighting, check for DAAD in the addons option of the editor.

DAAD Programming

Please notice that apart of this detailed information, there is an online tutorial made by Uto you may follow at the following URL: https://medium.com/@uto_dev/a-daad-tutorial-for-beginners-1-b2568ec4df05

Overview

When the game interpreter starts, it will run the information you entered in the TEST.DSF file. If you have had a look at those files, you may have found several sections to define objects, locations, etc. and at the end the processes. The processes are the DAAD programming code, which will determine how the game works. When a game starts, process 0 is run first ("/PRO 0" section).

Please notice TEST.DSF file already contain code that makes DAAD behave very much like PAW, so if you ever used PAW, it would be easy for you to adapt.

The Flags

The flags are like containers that can hold a value, you can change the value and check later, so you know something has happened. For instance you can set a flag to value 1 when an evil troll has run away due to some player action, and then you can check if that action has been performed when you want to cross a bridge where the evil troll was, and determine if he run away or not.

There are 256 flags available in DAAD, so you have plenty of them for your puzzles, despite some of them have an internal meaning. For instance, flag 38, also known as "fPlayer", holds the number of the location where the player is right now, so if you change it, you are moving the player somewhere else, but also you can use it to check if the player is at a given location or not, in order to determine if he can perform some action (i.e. sleep in the bedroom, but not in the bathroom). System flags, as they are named, are explained later in this section.

Initialization

When the game starts, the system initialization 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 flag 38, *fPlayer*, is now zero.

Start

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 initialize.

Search Process Table 0

Process table 0 will normally be the main loop of the interpreter.

Figure 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 OS rather than a calling table.

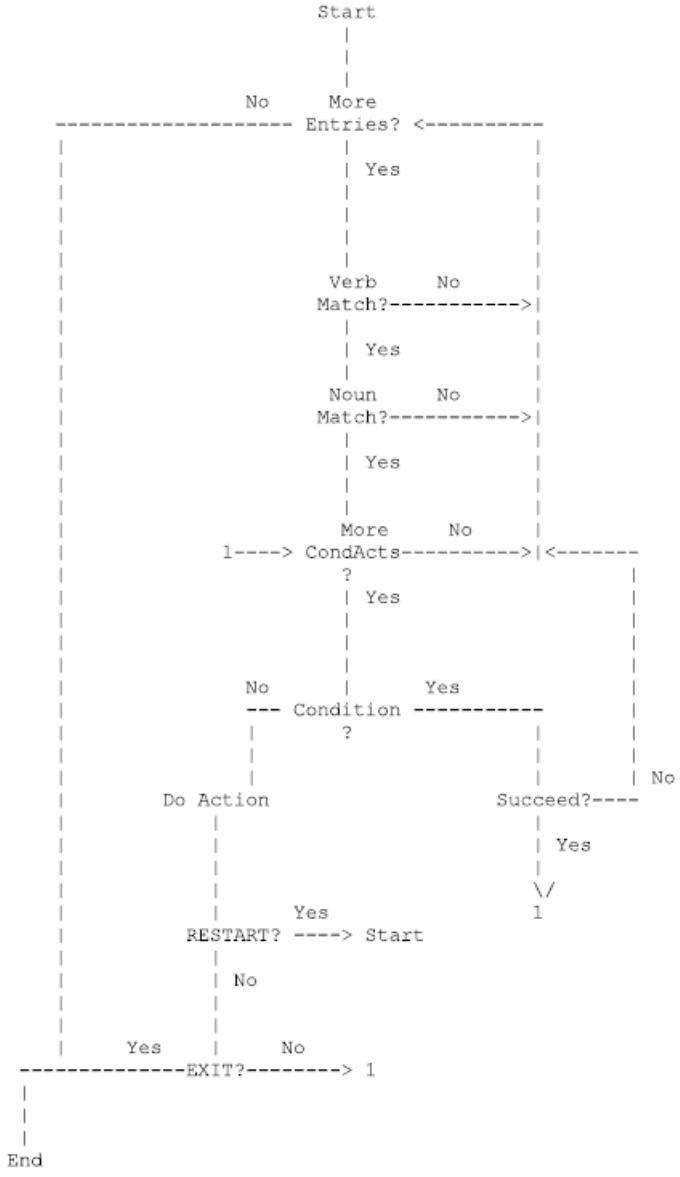

Figure 1

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. Please notice ">" sign marks a new entry and should be followed by VERB and NOUN or the "_" sign(s).

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:

- 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.

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

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

- 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.

- 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. The use of DESC may be specially confusing, as in DAAD it is much like MESSAGE but for the locations table, while the way DESC worked in PAW is done by RESTART condact.

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 fTimeout', 'LET fPrompt x' and 'LET fGFlags 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 any match as normal.

value is a value from 0 to 255.

Indirection

The first parameter on most actions can use indirection. This is indicated by placing at sign (@) before 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!

Conditions

Conditions which deal with the location of the player

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.

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.

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

Succeedsif 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 is not actually 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.

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. That is, although you can force player to write "UNLOCK DOOR WITH KEY" don't do it, and allow "UNLOCK DOOR" to work while key is carried, unless it's reasonable in the game script making the player specify he wants to unlock the door with the key and not with something else, or when unlocking the door with the key is not the most logical thing to do.

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.

Conditions 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 DAAD to look at the next CondAct only if the random number generated was between 1 and 50, a 50% chance of success.

Conditions providing a boolean TRUE/FALSE for subprocess 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.

Conditions for object attributes

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 TEST.DSF which check certain parts of the DAAD system flags:

Symbol	Flag	Attr. Number	Description
WEARABLE	57-Bit7	23	Current object is wearable
CONTAINER	56-Bit7	31	Current object is a container
LISTED	53-Bit7	55	If objects listed by LISTOBJ
TIMEOUT	49-Bit7	87	If Timeout last frame
GMODE	29-Bit7	257	Graphics available
MOUSE	29-Bit0	240	Mouse available

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

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 fGFlags 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 implemented in DAAD by:

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 this HASAT/HASNAT conditions. They are a true general-purpose bit tester for the first 64 flags.

Conditions to interact with the player

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.

Actions

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 present. This must be carried out by yourself 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 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 DAAD 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 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 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.

Some actions to interchange flag and objects data

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 runtime error. Setting an object to an invalid location will still be accepted as it presents no danger to the operation of DAAD.

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 - do not 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!

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 DAAD considers the next entry in the table - if present.

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 content 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!

Actions to manipulate the flags for screen mode, format, etc.

MODE option

Allows the current window to have its operation flags changed. 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 centered 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 separate 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 according to the lookup table given in the graphics editors, or as a palette 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. or MES "string"

Prints Message mesno in first case or prints Message between quotes if used the second way.

Please notice you can either user double or single quotes for the string but use the same for starting and ending.

MESSAGE mesno. MES "string"

Prints Message mesno in first case or prints Message between quotes if used the second way. After printing either message, it prints a new line.

Please notice you can either user double or single quotes for the string but use the same for starting and ending.

SYSMESS sysno.

Prints System Message sysno. You can use also "string" for SYSMESS.

DESC locno.

Prints the text for location locno. without a NEWLINE. You can use also "string" for DESC.

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 "You have "
PRINT 100
```

```
MESSAGE " 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.

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.

Actions to load or save the current state of the game

0 - Normal action
1 - Save to TAPE (i.e. does not 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.

Actions to allow the game to be paused

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.

Actions to deal with control of the parser

PARSE n

The parameter 'n' controls which level of string indentation is to be searched. Now 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 preceded 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 initialization 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.

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 runtime error will be generated.

REDO

Will restart the currently executing table.

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 preceded 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 implemented with rear patching. The symbol is defined by placing it in the source file immediately 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

> _ _
…
```

Actions which completely exit Response/Process execution

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 Initialize. 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 specify a part number to jump to on AUTOLOAD versions. Only the PCW supports this feature now. 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!

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 DAAD 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 DAAD 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.

Action to deal with sound

BEEP value value

Being first value the duration 1/50th of a second (i.e. 50 = 1 second), and the second value the frequency.

Actions to call external DAAD Routines

There are four actions used to call an external routine to DAAD: **SFX**, **GFX**, **EXTERN** and **CALL.** Despite they are powerful, they are also complicated, and EXTERN is used internally by DAAD Ready, so we will not detail them in the DAAD Ready manual. Please refer to original DAAD manual.

Actions to implement the primary graphics handling

Much as in the previous section, **PICTURE** and **DISPLAY** are used mainly for vector graphics, and DAAD Ready is not using them. It may affect you when creating games with graphics for DOS, PCW, Amiga and ST, but once again, we suggest you check DAAD original manual.

Maluva CondActs

Maluva is a DAAD extension which is included de facto in DAAD Ready Maluva handles many things and standardizes some others. For instance, using Maluva you can use new condActs that allow you to load pictures from disk, add additional texts, etc.

Please bear in mind DRC (the new compiler) knows about these condacts, and sometimes may replace them with old condacts automatically, when the original interpreter does not support them, but supports others. For instance it will replace any XPICTURE call with PICTURE calls in all 16 bit targets and PCW, or will replace XBEEP with BEEP in interpreters already supporting BEEP.

This is a list of Maluva condacts supported by target. None of the 16-bit targets support Maluva condacts:

-	ZX ESXDOS	ZX +3	CPC	C64	Plus/4	MSX	ZX Next	ZX Uno	PCW
XPICTURE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Native
XSAVE	Yes	Yes	Native	Native	Native	Yes	Yes	Yes	Native
XLOAD	Yes	Yes	Native	Native	Native	Yes	Yes	Yes	Native
XMESSAGE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
XPART	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
XBEEP	Native	Native	Yes	Native	Native	Yes	Native	Native	No
XSPLITSCR	No	No	Yes	Yes	No	No	No	Yes	No
XUNDONE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
XNEXTCLS	No	No	No	No	No	No	Yes	No	No
XNEXTRST	No	No	No	No	No	No	Yes	No	No
XSPEED	No	No	No	No	No	No	Yes	Yes	No

XPICTURE locno

Draws the picture for location locno on top of the screen if the picture file exists.

XSAVE opt

Saves the current game status to disk. The opt parameter is included for backwards compatibility with SAVE condact, but it is ignored. You can use XSAVE 0.

XLOAD opt

Loads the current game status from disk. The opt parameter it is also ignored.

XMES "string"

Prints the string in between quotes. Please notice this condact allows you to add additional messages apart of those in the message tables, so in case you need more text you can expand your game. XMES has the following limitations:

- Each message cannot be longer than 511 characters
- All messages created with XMES or XMESSAGE condacts, once compressed, cannot exceed 64K.
- Xmessages are read from disk on demand, so for some machines or devices this can be slow. Please use it with care. That also means that some glitches can appear in machine using split modes (C64, CPC and ZX-Uno) when using Xmessage, although for ZX-Uno it is hard to happen as it reads from a very fast SD card.

XMESSAGE "string"

Same as XMES but executes NEWLINE condact after printing text.

XPART value

If you have a two-part game and want to hold the files in the same disk, you may run into a problem when you must store two different XMB (xmessage) files. As Maluva expect them to be named the same for both parts writing them on same disk would make one overwrite the other. XPART may be run before any XMESSAGE/XMES is executed, so Maluva knows it is running some other part.

- For Spectrum and MSX, the parameter will be added to the '0' at'0.XMB'. So, for instance if value is 1, then Maluva will load 1.XMB instead of 0.XMB, if value is 7 then 7.XMB would be loaded, etc. Do not try to use values above 9, it will not work as you expect.

- For C64 and CPC, if value is not zero, then 50 will be added to file names. That means instead of loading 00.XMB, 01.XMB,

etc. the files 50.XMB, 51.XMB will be loaded. On C64 instead, of file "00" Maluva will read "50", instead of "01", it will take "51", etc.

XSPLITSCR value

For machines supporting split screen (currently Amstrad CPC and Commodore 64), this condact allows defining which mode will be used. Please notice the value has nothing to do with specific video modes in the target machine, but for a given configuration of modes. This is what mode means for each machine:

target		upper half video mode	lower half video mode
CPC	0	CPC Mode 1	CPC Mode 1
CPC	1	CPC Mode 0	CPC Mode 1
CPC	2	CPC Mode 2	CPC Mode 1
C64	0	C64 HiRes	C64 HiRes
C64	1	C64 Multicolor	C64 Hires
ZXUno	0	Standard	Standard
ZXUno	1	Timex HiColor	Standard

XUNDONE

XUNDONE changes the "done" status to not done. Every time DAAD executes an action, the internal interpreter status of "done" is set. That means if ISDONE or ISNDONE conditions are executed, they will succeed or not accordingly.

Please notice unlike NOTDONE, XUNDONE just clears the done status, but does not go to the end of the current process.

There are cases where even when an action is executed in a process, we do not want to consider the process "done". A clear example is the SYNONYM action. If you write an entry like this, so TURN ON LIGHT it is synonym to SWITCH ON, as soon as SYNONYM is executed, the internal status is already "done".

```
> TURN LIGHT
    SYNONYM SWITCH _
```

If later you write this entry, it may happen if the user forgot to type "ON", as PREP will fail, the entry will not be successful:

```
 SWITCH _
 PREP ON
 MESSAGE "You turn on the lights."
 DONE
```

But as SYNONYM was executed and DAAD considers something has been done, when the response table ends, as status is "done", there won't be a "I don't understand" or "You can't do that message", just nothing, so the game will look like this:

```
> TURN LIGHT
> (next prompt)
```

For this case and others were, despite something having been done, you do not want the "done" status to remain set, you can use XUNDONE, which will clear the "done" flag.

Pleas notice there is no XDONE or like activate he DONE flag, because you can activate it by running any action condact. If you want a condact which does nothing, you can use "GOTO @38".

XBEEP duration tone

- Duration is in 1/50 for second, so duration 50 equals one second.
- Tone is a value from 48 to 238, which allow playing notes for 8 different octaves. Values above or below will be taken as silence.

Only even values are valid, that is C is 48, C# is 50, D is 52, D# is 64, E is 66, F is 68, etc. If you use an odd value, you will probably just get noise.

For each octave this are the ranges:

Octave	Range
1	48-70
2	72-96

Octave	Range
3	98-118
4	120-142
5	144-166
6	168-190
7	192-214
8	216-238

For machines not supporting XBEEP, DRC compiler will replace XBEEP with BEEP.

XPLAY "string"

Although not exactly a Maluva condact, but a fake condact, XPLAY allows you to play melodies using BEEP in much convenient way. The string expected is just like the one in Spectrum +3 or other machines, i.e. "CDE" for (Do-Re-Mi). XPLAY basically supports most of the Spectrum +3 features but:

1) It is for all 8-bit machines except PCW. Please notice this is not a Spectrum-only feature.
2) XPLAY will be converted by the compiler to a sequence of XBEEPs, so the way it sounds will depend on how the BEEPs are played in each machine (i.e. in Spectrum they use the speaker, in CPC they use the AY chip, etc.)
3) It is mono-channel

As the format of the string is quite complicated to explain, please refer to Spectrum 128k/+2/+3 manual and search for the PLAY feature.

XNEXTCLS

This is a Spectrum Next exclusive function, which clears and disables layer 2. As in Spectrum Next games the pictures are painted in layer 2, DAAD cannot delete them with CLS. To allow the author to remove the pictures you can call this Maluva function.

XNEXTRST

This Spectrum Next exclusive function just makes a soft reset. The reason behind this function is if you use END condact to finish a Next game, it makes a reset without clearing and disabling the layer2, so you may find the game resets but the picture is still there beside your Next menu. To avoid that, it is recommended to avoid using END, and replace it with a combination of INKEY and this function. See "Ending Next games" section at the end of this manual.

XSPEED value

This function allows you to set CPU Speed in ZX-Uno and ZX Spectrum Next targets. Value can be 0 or 1, so 0 is 3.5Mhz and 1 is 7Mhz. Although both machines support faster speeds, they do sometimes interfere with some Maluva features, so we have kept 7Mhz as safe speed. Please notice if value is not 0 or 1, XSPEED call will be just ignored.

Increasing the speed makes the game text be printed faster and helps a lot with image loading. Sadly, original BEEP implementation in the ZX Spectrum interpreter is also speed up, so please make sure you restore standard speed before a BEEP or XPLAY, or it will sound at a higher frequency (sharper).

Error handling in Maluva condacts

Some Maluva functions may fail cause of external reasons. For instance, reading a picture from disk by XPICTURE may fail cause the file is not there, cause the user extracted the disk, etc. Maluva does nothing in those cases, at least apparently, so if for instance a picture is not there, no picture will be loaded, and if the XMessage file is not there, the message will not be shown but nothing else.

But Maluva does something: when a call to Maluva fails, the 7th bit of flag 20 is set, when it succeeds, the 7th bit of flag 20 is cleared. In other words, if after a XPICTURE call you do "GT 20 127", that condition will succeeded if the picture failed to load. That way, you can check and display a message (i.e. Warning: image not found, please insert disk):

-
```
_ _
XPICTURE @38
GT 20 127
MESSAGE "Warning: image not found, please insert disk"
```

or even this if you want to wait until the disk is inserted

```
_ _
XPICTURE @38
GT 20 127
MESSAGE "Warning: image not found, please insert disk and press
any key"
ANYKEY
SKIP -1
```

On the other hand, Maluva can also report errors via de "done" flag status. If you set bit 0 of flag 20 (LET 20 1), Maluva, aside of saving result to flag 20, will also update the internal "done" flag accordingly, so if the call succeeds, the done status will be set but if not the done status will remain as it was when the call was executed. That way you can use ISDONE or ISNDONE after Maluva call. For instance, if in the initialization process you have done that LET 20 1, you can later:

```
_ _
AT 200
LET 20 1
XPICTURE 120
ISNDONE
MESSAGE "Error: picture not found, please insert disk"
```

DAAD for PAW or Quill authors/developers

The processes

```
0 -> Do not change it
1 -> Do not change it
2 -> Do not change it
3 -> This works like PAW's process 1
4 -> This works like PAW's process 2
5 -> This works like PAW's response table
6 -> Initialization process, you can show some messages here, or pic-
tures, etc.
```

Objects

DAAD allows object attributes, which means each object has 16 values that can be active (1) or inactive (0). You can see that in the /OBJ section. Using HASNAT and HASAT condActs, you can check if a given object has a given value for some attribute. For instance, you can define that a specific attribute means the object is sharp, then you can make puzzle where you can use any object with the sharp attribute to solve.

Indirection

It has been explained above, and it was not supported by PAW.

Message printing

Rather than DAAD, this is an advantage of DRC, the new DAAD compiler. As you have seen in the description of MESSAGE, MES, SYSMESS and DESC, you can now just make MESSAGE have a string as parameter, so you do not have to bother with message numbers.

Also, Maluva provides XMESSAGE condAct, that also expects a string as parameter, and will give you 64K more of messages for longer adventures. Please notice xmessages are limited by memory used (64K) and not by number of messages.

Also is DRC the one that handles international character printing and takes care about repeated messages, so if you write same message twice, only one is used internally.

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 centered on 16 bits, 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** → Something wrong happened while loading or saving a file.

- **BREAK** → Player pressed break

- **Error n** → This is a machine specific error, please refer to that specific machine manual to find what error number n is.

- **Game Error n** →

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)

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 recognized. 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 do not 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. I.e. '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!

Spanish

If a Verb is less than 5 letters you will need to include the lo, la, los and las versions in the vocabulary. Obviously if it is four letters you only need 'l' ending as a synonym (TOMA -> TOMAL) if it is three letters you need 'lo' and 'la' (PON -> PONLO, PONLA) synonyms and if it is one or two letters you need 'lo','la','los' & 'las'! (DA -> DALE, DALOS, DALA, DALAS)

If you have a plural noun in the game which changes its stress then you need to include the stressed and unstressed.

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 compound 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.

Other languages

Despite DAAD it is only supporting English and Spanish, DAAD Ready has a limited support for other languages as Portuguese and German. That means, DAAD Ready is able to print special characters for those two languages, as "õ" or "ß", but doesn't make any change in the

parser, so the Portuguese games will be using the Spanish interpreter and the German games will be using the English interpreter. Please bear that in mind if creating games for those languages.

Now there is no support for other languages, but DAAD Ready supports printing characters from other languages, basically most of the characters in the LATIN-1 alphabet are printable.

The system flags

The normal flags are free for use in any way in games. But if you look at the TEST.DSF file, you will see it 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 will be true if a mouse is present.

0	When nonzero indicates game is dark (see also object 0)
1	Holds quantity of objects player is carrying (but not wearing)
2 to 28	These are not actually used by the DAAD interpreters anywhere. Thus, they would be free for use in your own games, but on the other hand the TEST.DSF file included within DAAD Ready it is using **flag 28** for internal functionalities, and Maluva extension is using **flag 20.** Thus, you should not use those flags, and in fact our recommendation is, to avoid conflicts with future expansions, that you start using flag 254 for your own tasks, then 254, then 253, etc. That way you will be far from any conflict and your game may be recompiled in 20 years without problems.
29	Bit 0 - Mouse present (16 -it only).
30	Score flag - not actually used directly by DAAD but its traditional!
31/32	(LSB/MSB) holds number of turns player has taken (actually, this is the number of phrases extracted from the players input).
33	holds the Verb for the current logical sentence
34	holds the first Noun for the current logical sentence
35	holds the Adjective for the current logical sentence
36	holds the Adverb for the current logical sentence
37	holds maximum number of objects conveyable (initially 4) Set using ABILITY action
38	holds current location of player
39/40	Unused
41	Gives stream number for input to use. 0 means current stream. Used Modulo 8. I.e. 8 is considered as 0!

42	holds prompt to use a system message number - (0 selects one of four randomly)
43	holds the Preposition in the current logical sentence
44	holds the second Noun in the current logical sentence
45	holds the Adjective for the second Noun in the current logical sentence
46	holds the current pronoun ("IT" usually) Noun
47	holds the current pronoun ("IT" usually) Adjective
48	holds Timeout duration required
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
50	holds object number for DOALL loop. i.e. value following DOALL
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.
52	holds players strength (maximum weight of objects carried and worn - initially 10). Set with ABILITY action.
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 DAAD list objects on the same line forming a valid sentence.
54	holds the present location of the currently referenced object
55	holds the weight of the currently referenced object
56	is 128 if the currently referenced object is a container.
57	is 128 if the currently referenced object is wearable
58/59	are the currently referenced objects user attributes
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.
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.
63	defines the currently active window. Note that this is a copy so changing its value will not affect the DAAD system.
64-254	Available for your own use
255	It was used by a previous version of TEST.DSF, so you better not use.

The Source File

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

Sections

The Control (CTL) section

This section is obsolete and not used anymore. There should be an underscore character in there, but it is just for historical reasons.

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.

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 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. With DAAD Ready you don't really need to add messages here like in the old PAWS or original DAAD, you can just write things like this in the processes, and the compiler will take care of creating a message for you. It will even find if you are using the same text twice or more, and use same message in that case:

MESSAGE "Hello World!"

Still, the table may be used because you want to have some messages with some specific number, to be used with indirection (i.e. MESSAGE @100, which will print the message whose number is at flag 100).

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 several 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 its synonyms to location 7 and NORTH or its 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: -

- The object's number
- The location at which the object is situated at the beginning of the adventure.
- The object's weight.
- Whether the object is a container.
- Whether the object can be worn/removed.
- The state of the other object attributes
- The noun and adjective associated with the object.

The Process tables (PRO) section

This section forms the heart of the source file providing the main game control. Each table consists of several 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 initialization 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.

Processes bundled with DAAD Ready

Please notice the DAAD Ready TEST.DSF file already contains more processes, to simulate the PAW way of working. Thus, Process 0, 1, 2 and 6 are internal to build that, but process 3 is much like process 1 in PAW, process 4 is much like process 2, and process 5 just like the response table.

Escape chars

While printing message, you can use some specific characters that will print not just what you they look like, but something in place. Please look at this table to understand what is printed:

Escape code	Description
_ (underscore)	It is replaced by the current referenced object description, if an indefinite article is found, it's removed. That allows system messages be like "You take the _.", so object "a lamp" gets "You take the lamp." as associated message.
@	Same as the underscore, but the article has its first letter uppercased. Only works for Spanish interpreter.
#b or #s	Prints a blank space
#k	When this character is printed, the game is paused and waits for a key to be pressed.
#n or \n	Prints a carriage return
#g	Print nothing, but from this moment on, every text printed will be using the upper 128 characters font. So, for instance #gt will print the character "t" of the upper font
#t	Restores the lower part of the character set.
#e	Prints the euro sign (€).

The pre-processor commands

Please notice this section is not for newbies, so do not worry if you don't get a word. You can even skip it if starting with DAAD once you have seen the #define and #ifdef definitions.

All preprocessor commands are preceded by a hash or gate character ('#'). They must occupy a line of their own. Generally, they can be placed almost anywhere, although it may not be appropriate or useful!

#define symbol expression

Define the case sensitive label to have the value given by expression. This can consist of other symbols, numbers, and operations between quotes. 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.

```
#define hours 24
#define days 2
```

```
#define weekendHours "days*hours"
#define noon "hours/2"
```

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 for machine target:

"pc","zx","cpc","msx","msx2","c64","cp4",etc.

Also, defines two symbols for machine subtarget when present, for instance "MODE_plus3" and "plus3" or "mode_next" and "next" for Spectrum.

It also creates the symbol BIT8 or BIT16 depending on target.

Also creates two symbols ROWS and COLS whose values are the text rows and columns for the whole screen for that target.

Please notice #define is there for two main reasons:

1) Being able to refer things by name instead of number, for instance if object 2 is a lamp, then you can make a define and reference it like this:

```
#define oLamp 2

CARRIED oLamp
DESTROY oLamp
```

Instead of using this old approach:

```
CARRIED 2
DESTROY 2
```

It also increases code readability if you must come back some day and fix something you wrote months or years ago.

2) To use with "ifdef" (see below).

#ifdef "symbol"
{#else}
#endif

This group of commands will occur together. Note that the #else is optional. Any lines following the #ifdef (up until the next #else or #endif) will be included in the compilation only if the symbol has a non zero (TRUE) value. If you use #ifndef instead of #ifdef, then it happens if FALSE.

```
#ifdef "zx"
MESSAGE "It's hard to do that with a Spectrum"
#else
MESSAGE "It's hard to do that without a Spectrum"
#endif
```

If you add that to the code, when the target is "ZX" (compiling for any ZX Spectrum target) players will get the first message, otherwise they will get the second one.

Although this is a simple example, you can use that to create different code that compiles depending on whether symbols have been defined or not.

#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 not be nested.

#echo "text"

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

e.g.

```
 #ifdef "PC"
 #echo Including IBM Display handler
 #include \LIB\PCDISP.DSF
 #endif
```

#incbin "filespec"

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

#defb "expression"

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

```
 #DEFB 1
 #defb PSIFLAG-1
```

#defw "expression"

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

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

#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.

#dbaddr symbol

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

#userprt 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 like #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 recognize 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.

Appendix

A - The character set

Throughout the whole DAAD system the same character set of 256 characters is used.

DAAD Ready is using some of the chars in upper 128 bytes to represent international character other than the original Spanish ones already supported.

DAAD Ready allows you to change the font for your game:

If you want to change your game font, you just must change the files at the ASSETS/CHARSET folder. These are the files:

- AD8x6.CHR is used by Spectrum, MSX1, MSX2, Amstrad PCW and PC.
- AD8x8.CHR is used by Amstrad CPC.
- C64bold.CHR is used by C64.

To modify the font, you can use GCS application at TOOLS\GCS.ZX-Paintbursh can also open them. If you save the font from GCS, save it for 8 bit, even if you want to use it for PC.

Please consider this:

- AD8x6.CHR font is shown as a 6 pixels wide font, so you must avoid using three rightmost columns. The two rightmost ones will not be even shown, the third one you better keep empty to avoid characters appear to close from each other.
- AD8x8.CHR font is shown using 8 pixels wide font. Despite that, it is the same font, so text in CPC is shown with more gaps between characters. You can modify the font to take two more columns, try not to use last column for the same reason said above.
- C64bold.CHR for C64 is a bold font, because CRT TVs connected to C64 via RF connection have problems showing thin letters in a way that is readable.

Please take in mind if you redefine characters in the upper font used for languages other than English, you won’t be able to use the for that language, but if you need to redefine the characters, you can always use one of the character you are not using (i.e. “ñ” if your game is in German or “õ” if your game is in Spanish).

B – Ending Spectrum Next Games

There is a small problem with ending Next games, as restarting the game does not clear the Layer2 memory area. To avoid that you will need to replace any END condAct in your code, with calling a specific process, whose code would have to be the following:

```
#define yesScancode 121;"y" keyboard code, for Spanish replace with 115
#define noScancode 110;"n" keyboard scancode

   > _     _       SYSMESS 13; ¿Play again?
                   NEWLINE
                   NEWLINE
                   PAUSE 10

   > _     _       INKEY
                   SKIP 1; Some key was pressed, let's check which one


   > _     _       SKIP -2; Nothing pressed, wait for keypress


   > _     _       EQ 60 yesScancode
                   GOTO 0
                   RESTART

   > _     _       EQ 60 noScancode
                   XNEXTCLS
                   XNEXTRST        ; Next cleanup and machine reset

   > _     _       SKIP -5; neither "y" not "n" was pressed
```

Then, instead of writing "END" condact, call the process you created by calling "PROCESS x".
This has not been included in the default code because is very Spectrum Next specific and it is difficult to determine when you would use END condact (probably in every player death point).

C - DAAD Ready customization

DAAD Ready saves some settings in the CONFIG.BAT file, which is loaded by all the other .BAT files (one per target). Also, DAAD Ready checks if CUSTOM.BAT file exists, and if so, loads it after CONFIG.BAT. That allows third-party tools, as Adventuron, integrate with DAAD Ready and modify values in CONFIG.BAT file.

D -Greetings

- Tim Gilberts, from Gilsfot/Infinite Imaginations, for creating DAAD
- Andrés Samudio, from Aventuras AD, for allowing public and free distribution of DAAD
- Richard Wilson, for creating WinAPE
- César Hernández, for creating ZEsarUX
- Mochilote, for creating CPCDiskXP
- Attila Grósz, for creating Yape
- Natalia Pujol, for the new MSX2 interpreter
- Imre Szell, for the new Plus/4 interpreter
- Marcin Skoczylas, for creating C64Debugger
- Habi, for creating CP/M Box, PCW Emulator
- To all those working in dosbox, VICE64, OpenMSX, etc.
- To Javier San José, for GCS font editor
- To John Newbigin, for DD for Windows
- To Amstrad and Locomotive, for allowing using their ROMS in Spectrum and CPC emulators.
- To Juan José Torres, for the DAAD logo.
- To Chris Ainsley, for his ideas, and for Visual Studio Code addon.

E -Licenses

DAAD Ready contains software of a lot of parts and has been built by Uto (@utodev), which is author of Maluva (DAAD extension AKA extern) and DRC (New DAAD compiler). All software used is either free or open source with free distribution, and some special cases where license was not specified, permission has been granted by authors.

- DRC and Maluva are (C) Uto and are both subject to their respective licenses. See more information and source code at https://github.com/daad-adventure-writer/DRC/wiki#LICENSE and
- ZEsarUX is (C) César Hernández and it is subject to its own license, please see license file at TOOLS/ZESARUX folder. Also find more information and source code at https://github.com/chernandezba/zesarux/
- WinAPE is (C) Richard Wilson. WinAPE is Freeware.
- OpenMSX is (C) several authors. Find more information at https://openmsx.org/
- C64DEbugger is (C) Marcin Skoczylas, although is also using some of the Vice64 code. If you like it and use it consider donating beer to slajerek@gmail.com, or if you prefer money, at http://tinyurl.com/C64Debugger-PayPal
- Yape is (C) Attila Grósz and its freeware. Please find more information at http://yape.plus4.net
- DOSBOX is (C) Peter "Qbix" Veenstra, Sjoerd "Harekiet" van der Berg, Tommy "fanskapet" Frössman, Ulf "Finster" Wohlers. Find source code and info at dosbox.com

- CP/M Box (PCW emulator) is (C) Habisoft. Find more information at http://www.habisoft.com/pcw/index_uk.asp
- MSX2DAAD is (C) NataliaPC, find more information and source code at https://github.com/nataliapc/msx2daad/wiki.
- CPCDiskXP is (C) Mochilote. Get more information at http://www.cpcmania.com/news.htm
- C1541 is part of Vice64. Find more information at https://vice-emu.sourceforge.io/
- dsktool (C) Ricardo Bittencourt, Tony Cruise and NataliaPC. Find source code and lastest info at https://github.com/nataliapc/MSX_devs/tree/master/dsktool
- PHP is (C) several users. Please find more information at php.net
- DD for windows is (c) John Newbigin and it's licensed under the GPL v2
- GCS is (C) Javier San José and Uto, and license it's pending to define
- Several ROM images supplied with WinAPE and ZEsarUX are still copyright to Amstrad Plc and Locomotive Software. Amstrad and Locomotive have given permission for these ROM images to be distributed with CPC emulators, but retains the copyright.