002e4fd0-0

Per Thulin

COLLABORATORS

	TITLE : 002e4fd0-0	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Per Thulin	December 25, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

002e4fd0-0

1.1 "

```
                                GRAAL ON-LINE REFERENCE
                    ========================

2.2 (c) Per Thulin / Performance Software 1997


Read~the~News!!!

~A~Very~Short~Introduction~~~~~~~~~~~~~~

~Machine~Requirements~~~~~~~~~~~~~~~~~~~

~About~this~Reference~~~~~~~~~~~~~~~~~~~

~Player~interface~and~shortcut~keys~~~~

~The~Structure~of~a~GRAAL~Game~~~~~~~~~~

~Syntax~Conventions~~~~~~~~~~~~~~~~~~~~

~Limitations,~Ranges,~Reserved~Numbers~

~Special~Characters~in~Text~Strings~~~~

~Statements~in~the~graal.main~file~~~~~

~Statements~in~the~n.section~files~~~~~

~Statements~in~the~n.room~files~~~~~~~~

~Conditions~~~~~~~~~~~~~~~~~~~~~~~~~~~~

~Commands~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

~Trouble-shooting~~~~~~~~~~~~~~~~~~~~~~
```

1.2 A Very Short Introduction

A Very Short Introduction

What is GRAAL?

GRAAL is a computer language that lets you create graphic adventures in a~"classic" format. All the tools you need are included except an art/animation package and music/sound sampler programs and equipment.

GRAAL has come a long way since its first release, and you can now customize most aspects of your adventure's look and feel. Starting with version 2.1, you can even control multiple characters.

If you want to see what the result may look like, just play the demo of "Olaf Longhair Goes East", which is included in the delivery package. You should also look out for "The GRAAL Herald", a diskmag using GRAAL itself to display the articles and run mini-demos showing stuff that isn't included in the "Olaf" demo

NOTE: The registered version of GRAAL contains some programming~tools essential for the serious adventure creator, for example~devices to compress and encrypt your scripts to make it impossible to~crack the game by looking at the files.

However, the function of the freely distributable version is not limited~in~any other way - you may create as large an adventure as you wish using~only~the GRAAL program contained in the demo package, if you have the stamina~and~perseverence to do so...!

1.3 Machine Requirements

MACHINE REQUIREMENTS

GRAAL should run on any machine with enough RAM:, although I haven't tested it with anything below Workbench 2.05. The games are a bit sluggish on standard A500's and the like. Isn't everything these days?

To develop GRAAL games, you need the following:

* A hard disk
* 2MB RAM

The following is very much recommended:

* A machine with at least the speed of an A1200
* Fast RAM

1.4 News in version 2

```
                News in GRAAL 2.2


As usual, the new or updated statements and commands added since 2.1 are
listed here for your convenience:


                VERB_ZONE:~Statement

                COMMAND_AREA:~Statement

                COMGR~command

                WALK_SPEED~command

                CHAR:~statement

                SELECT_CHAR:~statement

                Additions~to~the~cx~parameter~of~OBJECT~definitions

                TCURS~command

                3D:~statement

                MAKE3D~command

                PUT~command

                DLG_LAYOUT:~statement

                COMAREA~command

                BACKDROP:~statement~replacing~BG_IFF:

                MAX_ACTION: statement
                Be sure to read the README file for a walk-through of the new  ↩
                   features
and changes to the system, as well as a description of the bug fixes.

pethu@hotmail.com
```

1.5 About this Tutorial

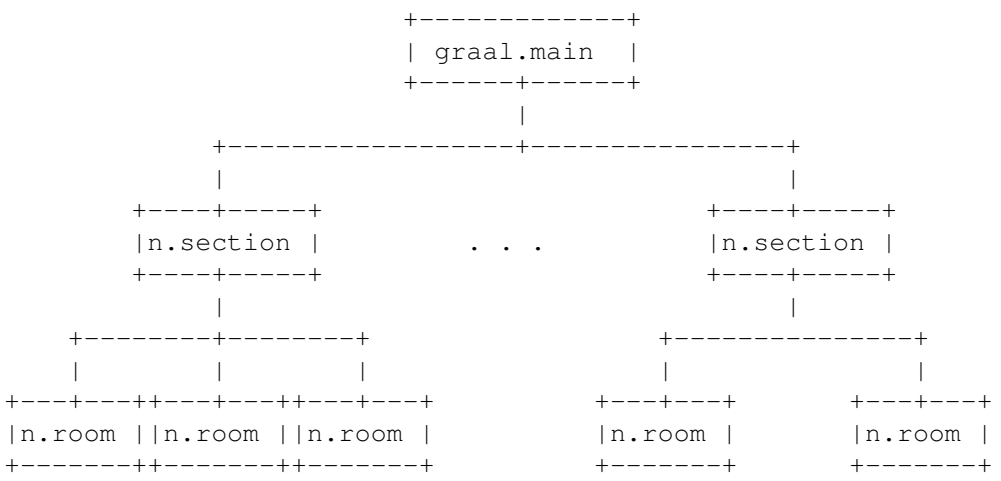
```
About this Reference


This guide is the definitive authority on all details of all GRAAL
statements, conditions, and commands. However, before you can write your
own games, you also need to know about the overall structure and idea
behind GRAAL, which are subjects we only touch upon in this file.
```

Therefore, I recommend all developers to read the GRAAL Manual prior to emerging too deep into the art of GRAAL writing. The GRAAL Manual is included in the GRAAL unregistered development package.

1.6 The Structure of a GRAAL Game

GRAAL script structure

A GRAAL adventures is based on a number of script files,~related~to~each~other~in the following way:



There is always a
graal.main
file, describing the main~characteristics~of~the~game and of the main character (which is the one~the~player~controls~and commands).

The entire adventure is divided into locations or "rooms"~as~they~are~called~ in GRAAL. The specifics of each room is defined in a
n.room
file

The rooms may be grouped into sections, as~shown~above. Each section has a
n.section
file.

The general idea is that if a player can issue a certain~command~or~do~a~ certain thing in just one specific room, the GRAAL code~to~handle~that~action~ should be placed in the corresponding .room script file.~If~the~action~can~ occur anywhere in the section , the code is placed in~the~.section~file;~and if~the action is something that can occur anywhere in~the~adventure,~it is~ taken~ care of in the graal.main file.

Whenever the player inputs a command, first the current
room~file~is~scanned~~for appropriate action, then the section file,
and~lastly~the~graal.main~file.~ This is why the graal.main file should
always~contain~general~"safety~nets" to~ handle most of the totally
off-the-wall~and~generally stupid~things~a player~ may try during the
game!

Apart from the three script file types above, there are two other
types~with~~specific purposes:

.scene files, containing the commands making up an
animated,~~non-interactive~"cut-scene".

.ptrn files, containing animation patterns too big and complex to
fit~into~the~code of a script file.

1.7 Syntax Conventions

GRAAL Syntax

Script Syntax

Scripts can normally contain empty lines, comment lines, and statement
lines. (Exceptions are the .scene files, which only contain commands,
and .ptrn files, which only contain animation patterns.)

Comment lines always begin with the characters /*.

Statement lines always begin with the statement followed by a colon and
one blank space. After that comes the parameters separated by
semicolons, but without spaces in between, like this:

STATEMENT: parameter;parameter;parameter;...;parameter

In the following statements, each parameter is a condition or command:

ACTION
- Actions taken for player input

DACT
- Actions executed when entering a new room

LACT
- Actions taken when player chooses a dialogue line

Each condition and command then has its own "internal" syntax. Usually,
parameters within a command are separated by colons (,).

Some statements and command allow you to leave parameter positions
"blank" to retain a previous value or set a default value. "blank" does
not mean "empty": You MUST enter a blank space in the position! For
example, if you wish to leave the two middle parameters of the SHOW

command empty, this is the way to do it:

SHOW 2, , ,4

(This example tells GRAAL to use image number 4 to display object 2, but let the image remain in its old place, since we left the x and y parameters blank.)

Command and Statement Notation

UPPERCASE

Type as written.

lowercase

Replace with value of specified type.

list

A list of values separated with | characters may be used. In conditions, any value in the list will make the condition TRUE. In commands, one of the alternatives in the list will be chosen at random each time the command is encountered during gameplay.

option1|option2

One of the options can be chosen.

[parameter,]

This parameter is optional.

Referring to the Contents of the Input Sentence

When the scripts are searched for commands to execute, what happens is based entirely on how the current input sentence from the player looks.

IN this reference, the first object in the current command sentence input by the player is referred to as OBJ1, and the second - if any - as OBJ2. The "command" itself is referred to as the VERB.

Referring to Objects and Images

An object number can be an ordinary number (n), a room object number (ROBJn) or a section object number (SOBJn).

An image number can be an ordinary global image (n), a room image (RBOBn) or a section image (SBOBn).

1.8 Limitations, Ranges, Reserved Numbers

GRAAL Limits, Ranges & Reserved Numbers

These are basic technical limitations to GRAAL 2.

SCREEN GRAPHICS

SCENE AREA: Background pictures must be lowres, 32 or 64 (EHB) colours, and between 320 and 640 pixels wide. (IFF picture files used for clipart must also be the same number of colours.)

COMMAND AND DIALOGUE AREAS: Pictures used must be hires, and 640 pixels wide. (No more than 16 colours.)

BOBS AND BOB IMAGES

Allowed BOB numbers for general use: 1-59

BOB numbers assigned:

60-63 - used for system graphics like the display of messages and names of objects and exits in the scene area

ANIMATION

Animation channels allowed for general use: 2-15

Channel 1 is normally used for the main character animation, although when characters are defined by

CHAR:
statements any channel may be used

for this instead (and is defined by the animation channel set for the object depicting the character).

STATEMENT AND COMMAND LIMITS

Item	Default Limit	Alterable
Max no of DACT lines in room file		Y (graal.main)
Max no of concurrent dialogues	6	Y -"-
Max no of dialogue lines per dialogue	30	Y -"-
Max no of LACT statements per dialogue	90	Y -"-
Max no of ACTION lines in a room		Y -"-
Max no of ACTION lines in a section		Y -"-

```
  Number of flags for each object              6    N
  Number of flags for each room               20    N
  Maximum number of floors in a room          12    N
  Maximum number of paths in a room           12    N
  Max.no of objects in each inventory         50    N
  Number of controllable characters            4    N
  Number of inventories                        4    N
  Number of string varibales                  12    N
  Max no of objects in current room           30    N
   (displayed simultaneously, not counting

               BOBON
               ,
               STATIC:
               ,
               ANIM:
                graphics)
```

1.9 Variables in Text Strings

```
               Special text characters


The following special character strings are replaced with variable
values etc. when used in SAY, THINK, RESP, and similar commands

  \        is replaced with a line break

  #R#n#f#  will be replaced by the value held in flag f for room n

  #O#n#f#  will be replaced by the value held in flag f for object n

  #VARn#    will be replaced by the string held in string variable n

  #OBJ1    will be replaced by the name of OBJ1

  #OBJ2    will be replaced by the name of OBJ2

  #Won     will be replaced by determination word n for object o
           (That is, o should be 1 for OBJ1, or 2 for OBJ2.)

  #TIME    will be replaced by the current game time in the format
           specified in the
                TIME_FORMAT
                 statement..

  #DATE    will be treplaced by the current game date in the format
           specified by the
                DATE_FORMAT
                 statement.

Example: OBJ1 is "apple", and word 1 for the object "apple" has been
defined as "an". The command
```

```
   SAY Just #W11 #OBJ1!

will then cause the character to  say

   Just an apple!
```

1.10 GRAAL Commands

```
                GRAAL Commands:


These are all the commands that can be used in the
                ACTION
                ,
                DACT
                , and

                LACT
                 statements, as well as in cutscene files.

General program flow control

                ~W(ait)~~~~~~~~~
                 Make a pause

                ~EXIT~~~~~~~~~~~
                 Stop searching for commands to execute

                ~REDO~~~~~~~~~~~
                 Re-run current input sentence

                ~CUTSCENE~~~~~~~
                 Execute a cutscene

                ~COMAREA~~~~~~~~
                 Hide or show the command area

                ~MARK~~~~~~~~~~~
                 Mark current position

                ~RESUME~~~~~~~~~
                 Resume marked position

                ~QUIT~~~~~~~~~~~
                 Quit GRAAL

                ~SAVE~~~~~~~~~~~
                 Save game at the last MARK

                ~LOAD~~~~~~~~~~~
                 Load a saved game

                ~EXEC~~~~~~~~~~~
                 Execute a cli command or program
```

```
                ~TRACE~~~~~~~~~~
                 Start or stop single-step trace mode

Timed events

                ~DOAFTER~~~~~~~~
                 Set a timer

                ~CANCEL~~~~~~~~~
                 Cancel a timer

Dialogue control

                ~DSET~~~~~~~~~~~
                 Start / change a dialogue

                ~LINE~~~~~~~~~~~
                 Change dialogue line number

                ~EDLG~~~~~~~~~~~
                 End a dialogue

Sentence control

                ~OBJ1~/~OBJ2~~~~
                 Change object number

                ~VERB~~~~~~~~~~~
                 Change verb number

Room control

                ~GOTO~~~~~~~~~~~
                 Go to a new room

                ~SETRF~~~~~~~~~~
                 Set room flag value

                ~ADDRF~~~~~~~~~~
                 Add to room flag value

                ~DECRF~~~~~~~~~~
                 Decrease room flag value

                ~SHOWEXIT~~~~~~~
                 Show a hideen exit

                ~HIDEEXIT~~~~~~~
                 Hide an exit

"Speech"

                ~SAY~~~~~~~~~~~~
                 Make character speak

                ~THINK~~~~~~~~~~
                 Make character think
```

```
                ~RESP~~~~~~~~~~~
                 Make speaking partner respond

Object manipulation

                ~HANDLE~~~~~~~~~
                 Make character handle object

                ~PICK~~~~~~~~~~~
                 Make character pick up object

                ~GET~~~~~~~~~~~~
                 Add object to inventory

                ~PUT~~~~~~~~~~~~
                 Remove object from inventory

                ~INVENTORY~~~~~~
                 Change the inventory

                ~NAME~~~~~~~~~~~
                 Alter the name of an object

                ~ICON~~~~~~~~~~~
                 Alter the icon image for inventory

                ~PREP~~~~~~~~~~~
                 Alter the preposition for an object

                ~NEWOBJ~~~~~~~~~
                 Create or modify an object

                ~SETOF~~~~~~~~~~
                 Set object flag value

                ~ADDOF~~~~~~~~~~
                 Add to object flag value

                ~DECOF~~~~~~~~~~
                 Decrease object flag value

Object display

                ~OMOVE~~~~~~~~~~
                 Move and animate object

                ~SHOW~~~~~~~~~~~
                 Show object

                ~HIDE~~~~~~~~~~~
                 Hide object

                ~OBJONTOP~~~~~~~
                 Put object on top of other objects

Main character display
```

```
                ~CBOB~~~~~~~~~~~
                 Change character image

                ~CMOVE~~~~~~~~~~
                 Move character

                ~WALK_SPEED~~~~~
                 Main character walking speed

                ~MOBJ~~~~~~~~~~~
                 Move character next to object

                ~MEXIT~~~~~~~~~~
                 Move character to exit

                ~CPOS~~~~~~~~~~~
                 Change character position

                ~CHAR~~~~~~~~~~~
                 Hide / display character

                ~SWITCH~~~~~~~~~
                 Switch controlled character

                ~FOLLOW~~~~~~~~~
                 Make another character follow

Floor control

                ~FLOOR~~~~~~~~~~
                 Define floor

                ~NFLOOR~~~~~~~~~
                 Set number of floors

                ~SETFLOOR~~~~~~~
                 Change character's floor

String manipulation

                ~PROMPT~~~~~~~~~
                 Prompt player for string input

                ~SETVAR~~~~~~~~~
                 Set a string variable

Audio control

                ~TRACK~~~~~~~~~~
                 Soundtracker module control

                ~EFFECT~~~~~~~~~
                 Pre-load a sound effect

                ~SOUND~~~~~~~~~~
                 Play a pre-loaded sound effect
```

```
                ~SAMLOAD~~~~~~~~
                 Load raw or IFF sample

                ~SAM~~~~~~~~~~~~
                 Sample control

Graphics control

                ~CLPART~~~~~~~~~
                 Load a clipart picture file

                ~BOBS~~~~~~~~~~~
                 Grab BOB images from clipart picture

                ~HOTSP~~~~~~~~~~
                 Alter the hotspot of an image

                ~LIGHTS~~~~~~~~~
                 Fade scene area in or out

                ~COLOUR~~~~~~~~~
                 Change a single colour

                ~FADE~~~~~~~~~~~
                 Fade one colour to another

                ~CAMERA~~~~~~~~~
                 Pan the camera to any part of background

                ~TITLE~~~~~~~~~~
                 Display / remove a title screen

                ~TYPE~~~~~~~~~~~
                 Type text on title screen

                ~TEXT~~~~~~~~~~~
                 Display text in scene area

                ~BOBON~~~~~~~~~~
                 Show a BOB

                ~BOBOFF~~~~~~~~~
                 Remove a BOB

                ~PBOB~~~~~~~~~~~
                 Paste a BOB image

                ~COMGR~~~~~~~~~~
                 Paste a BOB image in command area

Time and date manipulation

                ~SETDATE~~~~~~~~
                 Set the date

                ~SETTIME~~~~~~~~
```

```
                 Set the time

                ~ADDTIME~~~~~~~~
                 Advance the time

                ~SAVETIME~~~~~~~
                 Save the current time and date

                ~RESTORETIME~~~~
                 Restore the saved time and date

Special Cutscene commands:

                ~NOBREAK~~~~~~~~
                 Disable [Esc] in cutscene

                ~FINAL~~~~~~~~~~
                 Marks cutscene [Esc] resume point
```

1.11 GRAAL Conditions

```
                GRAAL Conditions:

These are the conditions that can be used in the
                ACTION
                ,
                DACT
                ,
                LACT
                , and

                LINE
                 statements.


                ~IFOBJ~/~IFOBJ2~~~~~
                 Test objects in the input sentence

                ~IFOF~/~IFOF2~~~~~~~
                 Test object flags

                ~IFROOM~~~~~~~~~~~~~
                 Test current room

                ~IFRF~~~~~~~~~~~~~~~
                 Test room flags

                ~IFCARR~/~IFNOTCARR~
                 Test if object is in inventory

                ~IFPICK~~~~~~~~~~~~~
                 Test if object can be picked up

                ~IFTYPE~~~~~~~~~~~~~
                 Test object types
```

```
~IFSPOS~~~~~~~~~~~~~
 Test room starting position

~IFCBOB~~~~~~~~~~~~~
 Test current character image

~IFFLOOR~~~~~~~~~~~~
 Test the current floor

~IFDATE~~~~~~~~~~~~~
 Test the date

~IFTIME~~~~~~~~~~~~~
 Test the time

~IFWEEKDAY~~~~~~~~~~
 Test the day of the week

~IFNOTSAVEDISK~~~~~~
 Test if the saved games disk is present

~IFEXISTS~~~~~~~~~~~
 Test if a certain saved game exists

~IFCHAR~~~~~~~~~~~~~
 Test the currently controlled character

~IFHERE~~~~~~~~~~~~~
 Test if an object is in the current room

~IFVAR~~~~~~~~~~~~~~
 Test the contents of a string variable
```

1.12 animation sequences

```
Basics about GRAAL Animation

Most simple animation sequences used in GRAAL have the following
format:

  A n,(image,time)(image,time)(image,time)...(image,time)

n is a number deciding how many times the animation sequence is played -
in GRAAL, it is set to 0 in most cases, which means the animation will
go on "forever". Forever in this case means "until GRAAL decides to put
a stop to it".

image is an image number.

time is the time the image is displayed before the next one comes on
screen (in 50ths of a second on PAL machines).

Example: A 0,(RBOB1,12)(RBOB2,12)(RBOB3,12)(RBOB4,12) would play the
sequence of four room BOB images over and over again. Note that the
```

commas are not GRAAL parameter separators in this case - they are all
part of the same sequence definition!

1.13 The GRAAL player interface

The GRAAL Player Interface

Although the graphics of the player interface can be changed very much
to your liking, all GRAAL games play in pretty much the same way. It
presents the alternatives to the user in a very clear and precise way to
let them know exactly what objects can be manipulated and what options
are available at any time.

This is an attempt to explain the elements of GRAAL's intuitive control
method. which is really much harder than just doing it:) It also
contains some "style guide" tips...

SENTENCE AND OBJECT DISPLAYS:

OBJECTS IN THE SCENE AREA:

As soon as the mouse cursor moves over an object with a name consisting
of anything but an empty string, its name appears above the cursor.

If there is a default command associated with the object, and neither a
verb or other object has actually been clicked by the player, the
default command for the object beneath the mouse cursor is shown in the
sentence box.

If a verb has been previously clicked, the name of the object also
appears in the sentence box. IF the verb / object combination requries a
second object to beclicked, the appropriate preposition is also
displayed in the sentence box.

OBJECTS IN THE INVENTORY:

Pointing to an object in the inventory works the same way as pointing to
an object in the scene area, with the exception that its name does not
appear above the cursor - instead, it appears in the sentence box IF a
scene area object has not been selected already.

VERBS:

If you move the mouse cursor over a verb in the command area, and no
verb has been clicked, the name of the verb appears in the sentence box.
Once more, IF an object is already there and the verb/object combination
requires a second object, the preposition will also be shown.

EXITS:

If you move the mouse pointer over an exit, and the name of the exit is anything but an empty string, its name apperas above the mouse cursor. Note that object and exit cursor texts in most cases SHOULD be coloured differently, so that players don't leave the room and waste a lot of time by accident.

If you try to use an exit, the message "GO TO exit name" will be shown in the sentence box. If the exit name is blank, no message at all is shown.

KEYBOARD KEYS

PLAYER KEYS

These keys are avaliable regardless of whether the system is in developer or runtime mode (although some of them may be disabled by special statements in graal.main):

S and L	both bring up the same save/load requester. (May be disabled by the DISABLE_SAVE: statement.)
space	puts the game in pause mode. Any key continues.
F	displays the amount of free memory (mainly for debugging purposes.)
M	toggles sound on and off.
V	displays the adventure name and version information.
I	increases the speed of sentence displays.
D	decreases the speed of sentence displays.
Q	quits the game. (May be disabled by the DISABLE_QUIT: statement, if you provide another way to QUIT.)
Esc	jumps to the end of a cutscene
.	ends a pause or sentence display

left mouse button	Select an object, exit, command or dialogue alternative
right mouse button	Select default object action or end a pause

DEVELOPER KEYS

G	brings up the monitor screen.
R	starts a macro recording. Pressing R again stops the macro recording and asks for a name for the new macro.
P	asks for a macro name, then starts playing it.
ctrl+c	aborts GRAAL. Use only in emergencies, as this does not clean up memory, leaves FONTS: assigned to RAM:FONTS, etc.
leftA+m	(A="Amiga") switches between GRAAL and Workbench. Note that programs running concurrently will be sluggish, and only the fonts in your GRAAL FONTS: drawer will be available. (That's why I have included the DPAINT font in the drawer, because I sometimes need to run that in parallell...)

All developer keys are diabled in an encrypted game (registered users

only), unless a DEBUG: statement in graal.main specifically asks for
them to be turned on.

1.14 IFOBJ / IFOBJ2

IFOBJ Condition

Test an object in the current sentence

```
  IFOBJ obj|list
```

This condition is TRUE if the object number of OBJ1 is equal to obj, or
to any one of the objects in a list.

```
  IFOBJ2 object number | list
```

Same thing, but checks OBJ2.

1.15 IFPICK

IFPICK Condition

Test if an object can be picked up

```
  IFPICK [obj]
```

This condition is TRUE if OBJ1 (default) or the specified object can be
picked up.

Example:

```
  IFPICK;MOBJ;HANDLE;PICK;HANDLE -1
```

Move to object and pick it up only if it is defined as "pickable"!

1.16 IFOF

IFOF Condition

Test the value of an object flag

```
  IFOF [obj,]flag<op>value|list
```

This condition is TRUE if the flag of OBJ1, or the specified object,
passes the test (see <op> below).

```
  IFOF2 flag<op>value|list
```

This condition is TRUE if the flag of OBJ2 passes the test. This form is mainly kept for backwards compatibility - specify the object number of OBJ2 in the format above instead, if you have a choice.

<op>

op an be any of the standard logical operators: =, >, <, <>, >=, <=

value

The value can be a fixed integer number or a reference to another flag. The format for flag references is #R#roomnumber#flag# or #O#objectnumber#flag.

NOTE: flag references can only be used when testing a single value. That is, you cannot specify a list of flag references to test.

list

If a list of values is specified, the condition is true if one of the list values makes the condition true.

(There is no point in specifying a list if the operator is <> - c'mon, think about it!!)

Examples:

```
 IFOF 2=3
```

is true if object flag 2 of OBJ1 is 3

```
 IFOF 4,2>3
```

is true if object flag 2 of object 4 is greater than 3

```
 IFOF 4,2=2|4|6|8
```

is true if object flag 2 of object 4 is 2,4,6, or 8.

```
 IFOF 7,4<>#R#3#1#
```

is true if object flag 4 of object 7 is not equal to room flag 1 of room 3.

1.17 IFROOM

IFROOM Condition

Test the current room

IFROOM room|list

This condition is true if the current room matches the room number(s) specified.

1.18 IFRF

IFRF Condition

Test the value of a room flag

IFRF [room,]flag<op>value|list

The operator <op> can be any of the following logical operators:

=, <, >, <>, >=, <=

The IFRF condition works the same way as the IFOF condition, so click that to see a number of examples.

1.19 IFCARR / IFNOTCARR

IFCARR Condition

Test if object is in inventory

IFCARR [obj]

This condition is TRUE if the specified object is in the inventory. If no object number is specified, OBJ1 is assumed.

IFNOTCARR [obj]

TRUE if the object is NOT in the inventory.

1.20 IFTYPE

IFTYPE Condition

Test if an object is of specified type(s)

IFTYPE type|list

This condition is TRUE if the type character matches any of the characters defined in the object type for OBJ1. For example, in standard GRAAL notation, an object defined as DW is (D)ead and made of (W)ood. IFTYPE D would be true, as would IFTYPE S|W (checking if the object is of either stone or wood).

IFTYPE2 type|list

This condition checks OBJ2 according to the same rules as described for OBJ1 above.

1.21 IFSPOS

IFSPOS Condition

Test which starting position was last used

IFSPOS spos|list

This condition is TRUE, if the last GOTO command (or START_ROOM statement) pointed to the specified starting position (= START_POS statement).

Its main use is setting the room up in different ways in DACT statements, depending on which entrance was being used.

1.22 IFCBOB

IFCBOB Condition

Test the currently displayed character image

IFCBOB image|list

This is TRUE if the image used to display the main character matches the number given in the condition. It could, for example, be useful in creating "stall anims" (see
DOAFTER
) and other main character animations
that should look different depending on the main character's current appearance.

1.23 IFCHAR Condition

IFCHAR Condition

Tests which character is currently controlled by the player (in multiple character games)

IFCHAR number

This condition is TRUE if the character currently under player control matches the character number specified. Character numbers can be between 1 and 4. Character 1 is the initial, default character which must always be defined.

See also:
CHAR:
statement,
SWITCH
command

1.24 IFHERE Condition

IFHERE Condition

Tests if an object is in the room

IFHERE obj

This condition is TRUE if the specified object is in the current room.

In most normal cases, you don't need for the presence of an object - the only way it can show up as OBJ1 or OBJ2 in an input sentence from the player is if it is available! However, this condition comes in handy when the presence or absence of something movable influences the gameplay even though the object is not directly used in input sentences.

1.25 IFFLOOR

IFFLOOR Condition

Tests the current floor

IFFLOOR floor|list

This condition is true if the main character is currently on any of the specified floors.

1.26 IFDATE

IFDATE Condition

Test the game-date

IFDATE <op>date

<op>

is one of the logical operators >, <, or =

date

must be in the format year*10000+month*100+date. For example, August 1, 1996 is specified as

19960801

The date can also be specified as a reference to a room or object flag holding a date value (see SETOF and SETRF). The format is #R#roomnumber#flag# or #O#objectnumber#flag#.

Examples:

IFDATE =19960801

is true if GRAAL's calendar is equal to August 1, 1996.

IFDATE <19960801

is true if GRAAL's calendar has not yet reached August 1, 1996

IFDATE >19960801

is true if the date has passed.

IFDATE <#R#4#1#

is true if the date is less than the value held in room flag 1 of room 4.

1.27 IFTIME

IFTIME Condition

Test the game-time

```
  IFTIME <op>time

<op>

is one of the logical operators >, <, or =

time

must be specified as hours*100+minutes (and always in 24-hour format,
regardless of what the TIME_FORMAT: statement says).

So, 2:30 pm (or 14:30 in 24-hour format) is specified as

1430


The time can also be sepcified as a reference to a room or object flag
holding a time value (see SETOF and SETRF). The format is
#R#roomnumber#flag# or #O#objectnumber#flag#.


Examples:


IFTIME =1130

is true if the game clock is 11:30


IFTIME <1130

is true if the clock is less than 11:30


IFTIME >1130

is true if the clock is past 11:30


IFTIME <#O#1#8#

is true if the time is less than the value held in object flag 8 of
object 1.
```

1.28 IFNOTSAVEDISK

```
                IFNOTSAVEDISK Condition

Checks for a saved games disk


  IFNOTSAVEDISK
```

This is a little more than just a condition. If GRAAL comes across it
when a saved games disk is not available, it prompts the player to
insert it. The condition only becomes TRUE if the player cancels the
"insert saved games disk" procedure.

It is only intended to be used in your own save/load rooms. Say that you
have made room 50 your own save/load screen. From any room, you invoke
the save/load room in this way:

MARK;GOTO 50,1

(Note that the MARK is essential for any SAVE to work...)

In the very first DACT: line of your save room, you should place the
following:

DACT: IFNOTSAVEDISK;RESUME

This ensures that the rest of the room actions will only be accessible
if a saved games disk is available for coming SAVE and LOAD commands.

See also: The
SAVE
command for more info on coding personal save/load
routines.

1.29 IFEXISTS

IFEXISTS Condition

Checks if a certain saved game file exists

IFEXISTS game_no

This is TRUE if a saved game file corresponding to game_no exists, and
it is valid for the current version of the adventure and GRAAL driver.

See also: The
SAVE
command for more info on coding your own save/load
routines.

1.30 IFVAR Condition

IFVAR Condition

Tests the contents of a string variable

```
  IFVAR var=string|list

The test is TRUE if variable var is equal to the string

Example:

IFVAR 8=Hello world

is true if variable 8 holds the string "Hello world".

See the
                PROMPT
                 and
                SETVAR
                 commands
```

1.31 IFWEEKDAY

```
IFWEEKDAY Condition

Tests the day of the week


  IFWEEKDAY day_number|list

This condition is true if the weekday, according to the in-game
calender, is matched by the number(s) specified. 1=Monday, 2=Tuesday,
... 7=Sunday.

Are your shops open on Sunday?
```

1.32 W(ait)

```
W Command

Wait nn vertical blanks.


  W frames

This command creates a pause. The time is measured in frames or
"vertical blanks", which occur at the rate of 50 per second for PAL
systems and 60 per second for NTSC systems. On a PAL system,

  W 50

would cause a one second pause.

The W command allows the player to end the pause before the specified
time by pressing the full stop ( . ) or escape key.

(The escape key, when used in a cutscene, also causes a skip to the
```

FINAL section of the cutscene.)

1.33 EXIT

EXIT Command

Ends the execution of commands to handle the current input sentence from the player.

EXIT

is used when all actions for a user sentence has been performed, and you do not wish to search further in the .room, .section, and graal.main files for entries that match the sentence. It is used in ACTION: and DACT: statements, and combined with EDLG to end dialogues.

See also:
EDLG
,
REDO

1.34 REDO

REDO Command

Re-run the scripts after having changed the player's input sentence.

REDO

This command is used after having changed the current sentence contents with the OBJ1, OBJ2 and VERB commands. The whole idea is that sometimes you want exactly the same actions performed for different sentences, and using REDO is easier and less space-consuming than copying all the actions. For example, if you want the same actions taken for USE BOOK and OPEN BOOK, you can replace the verb USE (3) with the verb OPEN (4) and then start over with checking for appropriate actions. Example, assuming the book is object 1:

```
ACTION: 3;IFOBJ 1;VERB 4;REDO
```

The checking will start over with the first ACTION: statement in the same file, but now looking for actions for OPEN BOOK rather than USE BOOK, which was what the player entered. (Not to worry, the player will never see what is going on!)

1.35 CUTSCENE

CUTSCENE Command

Loads and executes contents of a cutscene file

```
  CUTSCENE cutscene_no,S|F|H|N|NF
```

Rather straight forward, this one. You only have to remember that cutscenes can only contain commands and not conditions, and also note the effect the second parameter has on the cutscene indicator (in Olaf's case, the movie camera icon) that is shown instead of the command buttons while a cutscene is being played.

S

Cutscene indicator will be shown as normal and taken away when this cutscene has finished playing.

N

The whole command area will disappear during the cutscene, and return when the cutscene has finished playing.

F

The cutscene indicator will appear as normal but remain on screen when this cutscene finishes. This should be used if several cutscenes are played in sequence, or when cutscenes are "nested" (=called from inside other cutscenes).

NF

The command area will disappear during the cutscene and remain hidden until a cutscene command containing the "N" parameter restores it. (Does the same for "N" as "F" does for "S", if you know what I mean...)

H

The cutscene indicator will not be used at all. Use for short cutscenes and cutscenes with the NOBREAK command, if appropriate.

See also:
NOBREAK

1.36 COMAREA command

COMAREA Command

Turns the command area off and on

```
  COMAREA OFF|ON
```

Use this command to turn the command area off and on. it works exactly like calling a "blank" cutscene with the NF (OFF) or N (ON) parameter.

OFF

The command area disappears. While it is hidden, any cutscenes executed must use the "NF" parameter to keep it hidden. DSET and PROMPT may be used while the command area is hidden.

ON

Sooner or later, you MUST turn the command area back on using this.

Note: Saving and loading of games should nomally only be done while the command area is visible!

1.37 QUIT

QUIT Command

Cleans up and quits GRAAL

```
  QUIT
```

This command simply kills GRAAL. Use it after displaying an end-of-game screen, for example. If you provide your own way to quit the game, you probably also want to put the DISABLE_QUIT: statement in graal.main.

1.38 EXEC

EXEC Command

Executes a cli command or program

```
  EXEC FG|BG,command with parameters
```

This command executes the cli command or program with any parameters that are supplied. GRAAL execution is halted until the command has finished.

If "FG" is specified, GRAAL is switched to the back of the display, showing the workbench screen while executing the command. When execution has finished, control is switched back to GRAAL which is once more put into the foreground.

IF "BG" is specified, the GRAAL screen remains visible while the command is executed invisibly in the background. (But GRAAL execution is still halted until the command has finished.)

You can use this for anything you like, but it’s mainly here to support

intro screens and animations that Amos and GRAAL can't handle.

Note: If executing a command in the foreground (with the FG parameter), the Workbench screen is briefly visible before and after the command does its stuff - this is, sadly, unavoidable.

1.39 SAVE

SAVE Command

Saved the game at the last position set with the MARK command

SAVE game_no

The command creates a saved game file on the saved game disk. You must ensure the saved game disk is present with a
IFNOTSAVEDISK
condition
before attempting this. Using these and related conditions and commands, you can code your own save/load rooms to replace the cheesy built-in GRAAL requesters.

The saved game file will be named game_noSAVE.GRAAL. For example,

SAVE 1

creates the file 1SAVE.GRAAL

See also:
LOAD
,
MARK
and
RESUME
commands,
IFNOTSAVEDISK
and
IFEXISTS
conditions,
DISABLE_SAVE:
statement.

1.40 LOAD

LOAD Command

Loads a saved game

LOAD game_no

This load a previously saved game. Use this to construct your own save/load routines.

See also: The
SAVE
command for a more thorough discussion of the subject.

1.41 DOAFTER

DOAFTER Command

Sets a timer do execute events when a certain time period has elapsed

DOAFTER interval,obj,device

GRAAL has three "timer devices" which can be used to make things happen after a certain period of time or at (almost) regular intervals. Once set up, they continue to operate until a
CANCEL
command defuses them.

Possible uses are machinery where a sequence of actions must be carried out within a certain time limit, rooms where exits close if you take too long, updating of time data, etc.

interval

This is the timer interval in seconds. (Or, alas, in 5/6ths of seconds if you are an NTSC user. That's standards for you!) If you specify an interval like "10-50", this means the interval will be a random number between 10 and 50. However, once the interval has been set, it remains constant as long as the timer operates - unless it is changed by a new DOAFTER command. Also, timer device "0" is a little different - see below.

obj

What happens when the interval has elapsed? GRAAL starts looking for ACTION: statements for the special verb number -1, that's what. And which of those will it use? That depends on the event object, which can be any number. For example, if the event object is set to 3, GRAAL will execute statements beginning with

ACTION: -1;IFOBJ 3;

If you only use the one timer and intend to always run through all the timer action, you may choose not to test on the object number at all, and any number will do nicely here.

Control is returned to the player with an EXIT command, same as always.

Before the EXIT command, you may wish to put one of the following:

 * A new DOAFTER command for the same device, altering its function
   or the time interval
 * A CANCEL command, defusing the timer.

device

This is the timer device, which is a simple number from 0 to 2. Each device is its own little time bomb - however, device 0 works in a slightly different way from the rest.

Device 0 keeps track of how long it has been since the player's last input to the game, rather than counting how long it's been since the DOAFTER command was given. Each mouse click is counted as an action, and thus resets the timer. This timer is also completely disabled during dialogues.

This means that timer 0 can be used for things like "stall anims" - making the character urge you to do something if you take too long, for example. (A classic example of this sort of timer is the one in that good old text adventure, "the Hobbit": "You wait... time passes. And it did.)

Limitations to the timed events:

Because GRAAL only checks for elapsed timers when it waits for player input in dialogue or command mode, the timing is very approximate. For example, if a timer elapses while your hero is making a long speech, the timed event will occur only after the speech stops and when return normally should have returned directly to the player.

In other words, you will have to think about how you set the action in scenes where timing events are critical. Preferably, there shouldn't be long sequences without checks for player input occuring at the same time.

Also note that execution of the timer commands in the ACTION: statements does not multitask, and if two or three timers elapse at the same time, the commands for each timer will be carried out in sequence, starting with timer 0 and ending with timer 2, possibly putting the timing off even further from what was expected.

Finally, NEVER use timers for things which can be achieved with repeating animation patterns - they are much more accurate in the timing and put much less strain on the system!

See also:
CANCEL
command

1.42 CANCEL

CANCEL Command

Cancels the function of a timer

CANCEL device

This command stops a timer started with the
DOAFTER
command.

device

is the device number of the timer (1-3).

See also:
DOAFTER
command

1.43 TCURS command

TCURS Command

Decides whether the cursor is displayed during timer events or not

TCURS ON|OFF

During the execution of timer events, the mouse pointer (cursor) is normally switched of just like when commands input by the player is being executed.

With TCURS ON, you can have the cursor remain on screen. (However, if the timer events take long to execute, this may annoy the player because normal player input is not possible, and there will be no indication of this fact.)

1.44 DSET

DSET Command

Handles dialogue alternatives

DSET dlg[,{dset_command}]

Tells the dialogue dlg how to behave using a number of commands, such as:

+n add line n to the available set of lines

```
  -n       take away line n from the line set temporarily (can be
           restored by another + later on)

  Nn       make line n never appear again in this game, even if
           a DSET+ appears later on.

  Ss       Save the current status (the set of alternatives the
           player sees) before "branching" in the dialogue. s is a set
           of saved lines, and can be 1-3

  Rs       Restore a previously saved dialogue status from set 1-3

  E        Erase all currently displayed dialogue lines, equal to
           giving a "-" command for each line. (This is also done
           automatically by the S (Save) command.
```

If you are not already involved in the dialogue, DSET will put the dialogue control area onto the screen instead of the normal control area.

If no commands are specified, the dialogue is only refreshed. However, even this may alter the set of alternatives the player actually sees - because the availability of the alternatives also depend upon conditions set in the LINE: statements themselves, and those conditions may be changed between DSET commands.

Remember that although dialogues are specified in room and section scripts, their numbers must be unique for the entire game.

Examples:

```
  DSET 4,+1,+3
```

shows lines 1 and 3 of dialogue 4

```
  DSET 4,-1,N3,+4
```

hides line 1 temporarily, line 3 forever, and adds line 4

```
  DSET 4,S1,+12,+13,+14
```

saves the current dialogue status in set 1, clearing all old input alternatives at the same time, and replaces them with lines 12, 13 and 14.

The special command "L" allows you to alter the dialogue status without showing what you are doing in the dialogue control area. This is mainly used to restore the dialogue to its proper status just before ending the dialogue, so that the proper alternatives will be in place when the player starts talking to the same dialogue partner the next time. For example, before we leave dialogue 4 we know that we want to go back to the way things looked before we saved the status in set 1:

```
  DSET 4,L,R1;EDLG;EXIT
```

restores the previously saved dialogue status for dialogue 4 just before

the dialogue is ended, without the user being disturbed by flickering alternatives in the dialogue control area. However, the next time the player engages in this dialogue, the old alternatives will be back to choose from.

See also:
EDLG

1.45 LINE

LINE Command

Alter the line chosen in a dialogue

LINE line

This command, in conjunction with th REDO command, lets you use the same reactions (LACT: statements) for a number of different dialogue alternatives - much easier than copying all commands into a number of LACT:s.

Example: You wish the reactions to line 5 in a dialogue to be exactly the same as those for line 3. Simply specify this:

LACT: 5;LINE 3;REDO

The program now goes through the LACT:s again, now looking for those who are connected to line 3 instead of line 5 which was actually chosen.

See also:
REDO

1.46 EDLG

EDLG Command

Ends a dialogue session

EDLG

This command ends the dialogue. The normal Control Area is put back on screen instead of the Dialogue Control Area. The last set of dialogue lines will be present as default if the dialogue is resumed later on.

Normally, you would not want to evaluate any more line action (LACT:) statements after having decided to end the dialogue. Therefore, you should normally put an EXIT command right behind the EDLG:

EDLG;EXIT

See also:
DSET

1.47 OBJ1 / OBJ2

OBJ1 / OBJ2 Command

Alters the object number for OBJ1 or OBJ2.

```
OBJ1 [obj]
OBJ2 [obj]
```

These commands are used in two main ways:

a) to temporarily put another object number in the place of OBJ1 or OBJ2 in order to manipulate an object using other commands. In this case, you only need a simple OBJ1 or OBJ2 without any parameters to change the object number back to what it originally was when you are through manipulating the object you specified with "obj". For example, imagine you are about to open a can of gasoline in a room with a lit candle. The gasolin has object number 15 and is currently OBJ1, the object number for the candle is 20.

```
...OBJ1 20;MOBJ;HANDLE;SHOW 20, , ,10;SAY I put the light out
first;HANDLE -1;OBJ1 15;...
```

would be an easy way to switch from the gasolin, operate the candle, and then switch back to working with the gasolin.

b) to alter the object handled and then use the
REDO
command to run
through all action statements again.

Note: Exit numbers used to check which exit was clicked is actually a special use of the OBJ1 variable. This must be remembered when coding ACTION: statements for verb 0 (= exit click).

See also:
VERB
,
REDO

1.48 VERB

VERB Command

Alters the current verb

```
VERB verb_no
```

Use this to alter the verb in the current sentence. Mainly used before the
REDO
command to make one action synonym to another.

If no verb number is specified, the verb number before the last VERB command is restored (but why you should want to do that, I don't know at this stage.)

See also: OBJ1/OBJ2, REDO

1.49 ROOM

ROOM Command

Alter the current room

ROOM <room number>

A bit obsolete, this one. You can set flags for rooms other than the current using a special form of the SETRF command, so this one may soon be deleted. Don't use it.

Anyway, specifying ROOM without the parameter brings back the room number that was in effect before the last ROOM <room number> was called, just like OBJ1/OBJ2 can restore the previous object if used without the parameter.

1.50 LINE

LINE Command

Alter the line chosen in a dialogue

LINE line

This command, in conjunction with th REDO command, lets you use the same reactions (LACT: statements) for a number of different dialogue alternatives - much easier than copying all commands into a number of LACT:s.

Example: You wish the reactions to line 5 in a dialogue to be exactly the same as those for line 3. Simply specify this:

LACT: 5;LINE 3;REDO

The program now goes through the LACT:s again, now looking for those who are connected to line 3 instead of line 5 which was actually chosen.

See also:
REDO

1.51 EDLG

```
                EDLG Command

Ends a dialogue session


  EDLG

This command ends the dialogue. The normal Control Area is put back on
screen instead of the Dialogue Control Area. The last set of dialogue
lines will be present as default if the dialogue is resumed later on.

Normally, you would not want to evaluate any more line action (LACT:)
statements after having decided to end the dialogue. Therefore, you
should normally put an EXIT command right behind the EDLG:

  EDLG;EXIT

See also:
                DSET
```

1.52 OBJ1 / OBJ2

```
                OBJ1 / OBJ2 Command

Alters the object number for OBJ1 or OBJ2.

  OBJ1 [obj]
  OBJ2 [obj]

These commands are used in two main ways:

a) to temporarily put another object number in the place of OBJ1 or OBJ2
in order to manipulate an object using other commands. In this case, you
only need a simple OBJ1 or OBJ2 without any parameters  to change the
object number back to what it originally was when you are through
manipulating the object you specified with "obj". For example, imagine
you are about to open a can of gasoline in a room with a lit candle. The
gasolin  has object number 15 and is currently OBJ1, the object number
for the  candle is 20.

  ...OBJ1 20;MOBJ;HANDLE;SHOW 20, , ,10;SAY I put the light out
first;HANDLE -1;OBJ1 15;...

would be an easy way to switch from the gasolin, operate the candle, and
then switch back to working with the gasolin.

b) to alter the object handled and then use the
                REDO
                 command to run
through all action statements again.
```

Note: Exit numbers used to check which exit was clicked is actually a special use of the OBJ1 variable. This must be remembered when coding ACTION: statements for verb 0 (= exit click).

See also:
VERB
,
REDO

1.53 VERB

VERB Command

Alters the current verb

VERB verb_no

Use this to alter the verb in the current sentence. Mainly used before the
REDO
command to make one action synonym to another.

If no verb number is specified, the verb number before the last VERB command is restored (but why you should want to do that, I don't know at this stage.)

See also:
OBJ1/OBJ2
,
REDO

1.54 MARK

MARK Command

Mark the current game position

MARK [number]

This saves the current game state.

number

If number is specified, this command acts as a "save game" to RAM: (which can take some time).

Any number of MARKs can be stored using unique identification numbers to tell them apart. Just keep in mind that each MARK creates a file of about 20K in RAM:, and that the command takes a few seconds to perform. (Older machines can really struggle in comparison with 1200s or accelerated ones, so do not use it excessively.)

If a number is not included, the position is saved to a string of variables in memory, which is somewhat faster. However, only one position can be saved this way.

The position can be re-created later using a RESUME command.

You can use this in a number of ways:

* Implement your own "save to RAM:" commands.

* Provide an "ooops!" function, allowing the player an easy way to get back into a game where something just has gone terribly wrong...

* Cut away to cutscenes using other rooms and restore the game position afterwards, not having to care exactly what the scene looked like when the jump to the cutscene occured.

* Use a MARK as the very first command in the game and provide a "try again" option from an end-of-game screen.

See also:
RESUME command

1.55 RESUME

RESUME Command

Resume the action at the spot saved with the MARK command

```
RESUME [number]
```

number

The optional number must correspond to the one of a previous MARK command. See the MARK command for a detailed description of what is going on.

As with normal "load game" operations, GRAAL searches for and executes any ACTION: statements starting with the special verb number -2 before lighting the scene and returning control to the player. This is to enable you to perform any special actions that need to be taking, for example restoring global BOB images (which are not saved by a "save game" or a MARK command).

See also:
MARK command

1.56 SAY

SAY Command

Makes the main character speak a sentence (or two).

SAY sentence

This command uses the animations in the TALK_MAP statements of the graal.main file to animate the character during the length of the text display. The text display may use some special~characters to perform line breaks, put in variable values, etc.

Note that the SAY command can only be used if the main character is on screen, not if a CHAR~OFF or other command has hidden it!

See also:
THINK

1.57 GOTO

GOTO Command

Move to another room

GOTO room|list,entrance|list

This command automatically moves to a new room - it is not needed when using exits in the normal way, but is handy in cutscenes and the like. As usual, if a list of alternatives is specified, one is choosen at random. Could be used in maze-like surroundings, perhaps?

See also:
MARK
,
RESUME

1.58 THINK

THINK Command

Displays text above the main character

THINK sentence

This command behaves just like SAY, except it doesn't automatically animate the character. Good for "thinking" as well as using special

```
animation sequences shown using MOVE commands instead of the standard
TALK_MAP animations.

See also:
                SAY
```

1.59 RESP

```
                RESP Command

Make a dialogue partner respond


  RESP R|S,partner,sentence

R means after the character has "spoken", it will be displayed using its
default image or animation again. S means the image used just before the
RESP command was called will be used again.

partner

refers to the partner number assigned by the
                DLG:
                 statement in the
graal.main file.

The sentence is constructed just like sentences used in, for example,
the
                SAY
                 and
                THINK
                 commands.

See also:
                SAY
                 and
                THINK
                 commands,
                DLG
                 statement
```

1.60 HANDLE

```
HANDLE Command

Make the main character handle an object

  HANDLE [obj|LOW|MID|HIGH|-1]

HANDLE on its own uses the HANDLE_MAP animations specified in the
graal.main file to make the main character "operate" OBJ1.
```

HANDLE obj makes the character operate the specified object.

HANDLE LOW|MID|HIGH uses the "handle animation" - a general, convenient way to stretch out a hand without having to resort to CBOB or OMOVE 0,... commands.

HANDLE -1 resets the main character to what he/she looked like just before the previous HANDLE command took effect.

1.61 PICK

PICK Command

Pick up an object

PICK [obj]

Adds the specified object (or OBJ1, if no object number is specified) to the inventory and erases it from the scene area. This command is often preceded by a MOBJ and a
HANDLE
command to show on screen what's going on.

There is not a DROP command that automatically does the opposite. The most comfortable way to dispose of objects is letting the main character do so automatically when they have filled their purpose, using the

REMOVE
command.

See also:
GET
,
REMOVE

1.62 GET

GET Command

Add an object to the inventory

GET obj,U|N

The object is added to the inventory. Use "U" if the inventory display should be updated (which is the normal procedure), "N" if the inventory should be left unaffected, for example if GET is used during a dialogue, or you make a number of consecutive GETs letting only the last one update the display.

The difference between GET and
PICK

is that GET does not take any notice
of the object's previous whereabouts.

See also:
PICK
,
REMOVE

1.63 REMOVE

PUT Command

Relocates an object

PUT obj|OBJ1,U|N,room|Iinventory

This command can be used to put any object in any room or inventory,
except the current room and inventory, at any time.

If it was previously in the current inventory, it will be removed. In
this case, and if the inventory is displayed, you should use the "U"
parameter to update the inventory display.

To put an object in the current room, use the
SHOW
command.

To put an object in the current inventory, use the
GET
command.

obj|OBJ1

The number of the object to be relocated. If OBJ1 is specified, the
first object in the sentence is the one that is removed or relocated.

U|N

If U is specified, the inventory display is updated.

room|Iinventory

The object is placed in the specified room. If the room is specified as
0, the object "disappears" from the game!

I followed by an inventory number is only used when you have more than
one inventory in the game, and wish to place the object in an inventory
other than the one currently on screen. The default inventory is always
number one, but inventories can be swicthed with the
INVENTORY
command.

REMOVE 12,U,I2

would remove object 12 from the displayed inventory and put it in
inventory 2 instead. (It should thus become visible when you switch to
display inventory 2.)

Note:

See also:
GET
,
PICK

1.64 INVENTORY

INVENTORY Command

Changes the currently used inventory

```
INVENTORY number,U|N
```

This command allows the use of multiple inventories in GRAAL - up to six
inventories can be used, and the default inventory is number 1.

number

The new inventory number to use. All subsequent
GET
,
REMOVE
, and
PICK
commands will use this inventory.

U|N

Specify U if the inventory display should be updated immediately to show
the new inventory. Use N only when the inventory display is currently
unavailable and another command will refresh it before control is
returned to the user next time.

1.65 NAME

NAME Command

Alter the name of an object

```
NAME new_name[,word1,word2,word3]
```

Very often in an adventure, an object is "transformed" - that is, one object appears while another disappears at the same time. (For example, a parcel is opened to reveal a book - the parcel is gone, the book exists.) To save memory, it makes sense to use the old object number for the new object also, since there is no risk of confusion - the two objects never appear at the same time.

The command NAME alters the name of the current OBJ1 (see the syntax conventions). In addition, the determination words (see the OBJECT statement) may be altered to suit the new object description.

See also:
OBJ1 command,
OBJECT statement.

1.66 ICON

ICON Command

Alter the icon used for inventory display

```
ICON [obj,]image
```

This command does a bit of what the NAME command does for a text inventory, but you should probably use both the NAME and the ICON command when changing properties for objects in an icon display - NAME still determines the text shown when the object is referred to in the game, although it doesn't appear in the inventory list...

1.67 PREP

PREP Command

Alter the preposition of OBJ1

```
PREP [preposition]
```

By specifying a preposition for an object, the verb USE will assume this object must be used in conjunction with something else, and therefore awaits the input of a second object before checking for actions.

Specifying PREP without a preposition will do the opposite, allowing an object to be used on its own again.

1.68 NEWOBJ

```
                NEWOBJ Command

Creates or modifies an object


  NEWOBJ: object_parameters

This command acts exactly like the OBJECT: statement - only here, the
parameters are separated by commas instead of semi-colons, so you can
not place an animation sequence containing commas as the default object
image. Also note that this command resets all object flags to 0, and
shouldn't be used unless you experience some sort of emergency I haven't
been able to anticipate.

See also:
                OBJECT
                 statement
```

1.69 SETOF

```
                SETOF Command

Assigns a value to an object flag

  SETOF [obj,]flag=value|list

If no object number is specified, OBJ1 is assumed. The flag is set to
the value.

If the value is specified as #DATE, the value is the current in-game
date in the format year*10000+month*100+date, e.g. 19960801 for Augist
1, 1996

If the value is specified as #TIME, the value is the current in-game
time in the format hour*100+minutes, e.g. 2355 for 11:55 pm.

The value can also be a reference to a room or object flag. The format
is #R#roomnumber#flag# or #O#objectnumber#flag#

For example,

 SETOF 3,6=#R#3#4#

would set flag 6 for object 3 to the value held in room flag 4 of room
3.

If a list of values is specified, one of the values is chosen at random.
For example,

 SETOF 2=3|7|9

would set object flag 2 for object 1 to either 3, 7, or 9. A maximum of
```

12 values may be specified in a list.

See also:
ADDOF
,
DECOF
,
IFOF

1.70 ADDOF

ADDOF Command

Adds to or subtracts from a flag value

```
ADDOF [obj,]flag[,value]
ADDOF2 flag[,value]
```

ADDOF affects a flag for an object. If no value is specified, 1 is added. Negative values may be used, thus subtracting from the flag value.

ADDOF2 affects OBJ2 - this form is kept mainly for backwards compatibility, you can achieve exactly the same by specifying object 2's object number in most cases...

DECOF is used for special "countdown" purposes.

The value can be a reference to a room or object flag. The format is #R#roomnumber#flag# or #O#objectnumber#flag#.

For example,

```
ADDOF 1,#O#4#2#
```

would add the value held in object flag 2 for object 4 to object flag 1 for OBJ1.

Another example:

```
ADDOF2 4
```

would add 1 to object flag 4 of OBJ2.

See also:
SETOF
,
DECOF
,
IFOF

1.71 DECOF

```
                DECOF Command

Counts down the flag value to zero

  DECOF flag
  DECOF2 flag

DECOF is used for OBJ1, DECOF2 for OBJ2. The flag value is decreased by
1 until it reaches zero, then it stays there.

See also:
                SETOF
                ,
                ADDOF
                ,
                IFOF
```

1.72 SETRF

```
                SETRF Command

Assigns a value to a room flag

  SETRF [room,]flag=value|list

If no room number is specified, the current room is assumed. The flag is
set to the value.

If the value is specified as #DATE, the value will be the current
in-game date in the format year*10000+month*100+date, e.g. 19960801 for
August 1, 1996.

If the value is specified as #TIME, the value will be the current
in-game time in the format hours*100 + minutes, e.g. 2355 for 11:55 pm.

For example,

 SETRF 2,1=5

would set room flag 1 for room 2 to 5

The value can be a reference to a room or object flag. The format is
#R#roomnumber#flag# or #O#objectnumber#flag#

For example,

 SETRF 2=#R#2#1#

would set room flag 2 for the current room to the value of room flag 1
for room 2.

If a list of values is specified, one of the values is choosen at
```

random. For example,

```
 SETRF 2=3|7|9
```

would set room flag 2 to either 3, 7, or 9. A maximum of 12 values may be specified in a list.

```
 SETRF 1,3=#TIME
```

would set room flag 3 for room 1 to the current game time.

```
 SETRF 0,1=#DATE
```

would set room flag 1 for room 0 to the current game date. Room 0 is a "global" room never used as a normal room. Nevertheless, it exists flag-wise, so its 20 flags can be used to hold "global game values".

See also:

ADDRF
,
DECRF
,
IFRF

1.73 ADDRF

ADDRF Command

Adds to or subtracts from a room flag value

```
  ADDRF [room,]flag[,value]
```

ADDRF affects a flag for the current room. The number is added to the flag value. If the number is negative, a subtraction is performed.

The value may be a reference to a room or object flag. The format is #R#roomnumber#flag# or #O#objectnumber#flag#.

For example,

```
 ADDRF 3,1,#O#5#2#
```

adds the value held in object flag 2 of object 5 to room flag 1 of room 3.

```
 ADDRF 2
```

would add 1 to room flag 2 of the current room

```
 ADDRF 2,3
```

would add 3 to room flag 2 of the current room

```
 ADDRF 5,2,3

would add 3 to room flag 2 of room 5


See also:
                SETRF
                ,
                DECRF
                ,
                IFRF
```

1.74 PROMPT Command

```
                PROMPT Command

Prompts the user for input of a string variable


  PROMPT var,text

This command uses the dialogue area to display the prompt and
accompanying text. It can not be used while in dialogue mode.

var

The string variable (1-12) where the player's input will be held. This
can then be displayed using a text
                variable
                 or tested  with the
                IFVAR
                condition.

text

An explanatory text shown in the first line of the dialogue area

Example:

PROMPT 1,Please enter your name:

Whatever the player types will be stored in string variable 1.
```

1.75 SETVAR Command

```
                SETVAR Command

Sets a string variable
```

```
  SETVAR var,string

GRAAL has 12 string variables which can be set by the player (see

                PROMPT
                ) or with this command.

var

Variable number (1-12)

string

An arbitrary text string

Example:

SETVAR 1,Hello world!

will store the text "Hello world!" in string variable 1.
```

1.76 SHOWEXIT

```
                SHOWEXIT Command

Shows a previously hidden exit


  SHOWEXIT exit_no

This restores a previously hidden exit on screen

See also:
                HIDEEXIT
                 command
```

1.77 HIDEEXIT

```
                HIDEEXIT Command

Hides an exit


  HIDEEXIT exit_no

When entering a room, all exit defined by EXIT: statements are always
visible and usable. This command hides the exit. It can be restored
later by the SHOWEXIT command.

See also:
                SHOWEXIT
                 command
```

1.78 DECRF

```
                DECRF Command

Counts down the room flag value to zero

  DECRF flag

DECRF decreases the value of the flag until it reaches zero, then it
stays there.

See also:
                SETRF
                ,
                ADDRF
                ,
                IFRF
```

1.79 CBOB

```
                CBOB Command

Alter the image for the main character


  CBOB image

The main character changes to the specified image. The screen (hotspot)
position is not altered.

See also:
                CPOS
                ,
                CMOVE
                ,
                OMOVE
```

1.80 CMOVE

```
                CMOVE Command

Moves the character to a new screen position using the default
                WALK_...
                animations.

  CMOVE x,y,C|P

x and y are the screen (hotspot) coordinates. Use P to end the CMOVE
with an appropriate PAUSE_... image, C" link "ST_PAUSE" 0} image, C to
end with a
                STILL_...
                 image. C is mainly used when another CMOVE follows
```

immediately.

See also:
CBOB
,
CPOS
,
OMOVE

1.81 WALK_SPEED command

WALK_SPEED Command

Sets the pace of the main character animation

WALK_SPEED speed

See the description of the
WALK_SPEED
statement.

1.82 MOBJ

MOBJ Command

Move the main character next to an object

MOBJ [obj]

The main character is moved to the position indicated by the character offset parameters of the OBJECT statement or command. If no object number is given, OBJ1 is assumed.

1.83 MEXIT

MEXIT Command

Move character to exit

MEXIT

This command can only be used in an ACTION: 0;... statement, and moves the character to the exit point for the clicked exit, as specified in the corresponding
EXIT:
statement

1.84 CPOS

```
                CPOS Command

Alter the character's screen position


  CPOS x,y

Immediately alters the main character's screen position to x,y (without
walking there like
                CMOVE
                .) The image is not altered:
                CBOB
                 may be used
for that.

See also:
                CBOB
                ,
                CMOVE
                ,
                OMOVE
```

1.85 CHAR

```
                CHAR Command

Turn main character display on or off

  CHAR ON|OFF

CHAR OFF means the main character is not on screen - use for cutscenes,
animated intros and the like. CHAR ON restores the main character to the
position before CHAR OFF.

See also:
                CPOS
                ,
                CBOB
```

1.86 SWITCH

```
                SWITCH Command

Switches control between alternate controllable characters


  SWITCH character

Using
                CHAR:
```

Statements in graal.main, up to four characters can be defined that can be put under player control. This command switches control between them.

Note that inventorys are not automatically connected to characters, so an
INVENTORY
command should normally be given immediately before the SWITCH command.

Note: You cannot use SWITCH while a
CHAR~OFF
is in effect.

1.87 FOLLOW Command

FOLLOW Command

Makes an alternate controllable character follow the current character

```
FOLLOW character,startx,starty,followx,followy,delay
FOLLOW OFF
```

Any player action causing the controlled character to walk around, and any
CMOVE
command, will cause the following character to follow.

This "automatic following" procedure will be in effect until the next

SWITCH
, or FOLLOW OFF command. You may also alter the FOLLOW parameters at any time by giving a new FOLLOW command.

character

This is the number of the character that should follow the player-controlled character around.

startx,starty

If the controlled character is transported to a new room, the following character will also appear there. Its image will be offset from the controlled character by the amounts specified here. startx should always be a positive number: It is the Left/Right/Middle position in the

START_POS:
statements that determines whether the following character will be placed left or right of the controlled character. starty should probably be a small, negative number most of the time - because further up the picture is also further away in the scene, this will make the following character appear further away than the controlled character.

followx,followy

These are the offset amounts used when the CMOVE command or a player mouse click causes the controlled character to move. Both numbers should always be positive: In most cases, the following character will end up followy pixels further up the picture than the controlled character. The only exception is when the controlled character stops after "walking away" and ends up with its back towards the player - then, the following character stops followy pixels below the controlled character. This gives the illusion of the following character still keeping behind the controlled character.

delay

This specifies the time before the following character starts to move. A delay of 50 equals a one second wait.

Notes:

Unfortunately, there is no feasible way for me to make the controlled and following character - or indeed, multiple characters in the same room in any situation - keep out of each other's ways. This means the characters will sometimes seem to walk through each other. You can minimize the problem by selecting the possible movement paths carefully.

1.88 FLOOR

FLOOR Command

(Re-)defines a floor

FLOOR number,x1,y1,x2,y2,floormap1/.../floormapn

This command works exactly like the
FLOOR:
statement, and allows you to
re-arrange floors for a room any way you like. You can make previously unreachable areas accessible, or quite the opposite.

You must make sure that all floormaps are valid - changing a single floor may mean you have to use FLOOR commands for other floors to just to change the floor maps. And if you alter the number of floors, you must also use the
NFLOOR
command to set the new number of floors.

See also:
NFLOOR
and
SETFLOOR
commands,
FLOOR:

statement

1.89 NFLOOR

NFLOOR command

Changes the number of floors in a room

NFLOOR n_floors

Only use this command when you have changes the floor structure, and number of floors, in a room with the
FLOOR
command.

See also:
FLOOR
and
SETFLOOR
commands.

1.90 SETFLOOR

SETFLOOR command

Informs the system about the main character's whereabouts in the floor system.

SETFLOOR [character,]floor

Normally, GRAAL automatically keeps track of on which floor your character is currently positioned. There are, however, a few commands that may leave the system unaware about your hero's whereabouts.

These are the
FLOOR
,
OMOVE
, and
FOLLOW
commands. If one of these commands places your character on another floor previously, you should specify the new floor number with this command. Otherwise, strange things may happen when the character tries to move next.

character

Only use this parameter in multiple-character games. If character is not specified, it is always the current floor for the character currently under player control that is set.

See also:
FLOOR
and
OMOVE
commands, and the
FLOOR:
statement.

1.91 OMOVE

OMOVE Command

Move an object (or the main character) according to specified animation sequence

```
OMOVE obj_no,x,y,speed,FLIP| ,WAIT| ,anim|ptrn
```

The object's hot spot is moved to the x,y screen co-ordinates. During the movement, the
animation~sequence
specified in the last parameter is
used.

The x and y positions can be set relative to the main character's current position using CX+offset and CY+offset. Example:

```
OMOVE 2,CX+20,CY+0,1,FLIP, ,A
0,(SBOB1,12)(SBOB2,12)(SBOB3,12)(SBOB2,12)
```

moves object 2 to a position 20 pixels to the right of the main character, at the normal WALK_SPEED speed, using an animation consisting of four different images.

If the speed adjustment factor is 1, the speed will be the speed set with the WALK_SPEED parameter in the graal.main file. A lower number gives faster" link "ST_WALK_SPEED" 0} parameter in the graal.main file. A lower number gives faster movement, a higher number gives slower movement.

If FLIP is specified, and movement is from left to right, the images are used as supplied, but if the movement is from right to left, all images in the animation sequence are automatically flipped first. Specifying any other value (such as a blank) means the images are always used as specified.

If WAIT is specified, the entire animation sequence is carried out before GRAAL continues with the next command. Otherwise, GRAAL will not check if the animation has been concluded until the next OMOVE command for the same object. If you put several OMOVE commands for the same object next to each other, you should specify a blank space instead of WAIT - this eliminates the brief pauses between OMOVE commands that will otherwise occur. On the other hand, if the command following an OMOVE is

something like a SHOW command for the same object, always specify WAIT - otherwise the SHOW would be affecting the object before the animation sequence had a chance to finish.
  OMOVE can be used to move and animate the main character using other animations sequences than the default. Just specify object number 0 to point to the main character.

If x and y are left blank, the object is animated using the animation string at its curent position. Normally, the animation is automatically stopped when the object reaches the x,y position, and the first BOB image in the animation sequence will the be used as the still image. When no new x,y position is given, the animation goes on until another image-manipulating command for the object is encountered, for example SHOW or CBOB. Example:

```
  OMOVE 0, , ,1,A 0,(11,24)(12,24)
```

would animate the main character alternating between BOB images 11 and 12 indefinitely. (Well, until the BOB image for the main character is altered using some other command, anyway.)

See also:
SHOW
,
CMOVE
,
CPOS
,
CBOB
,
HIDE

1.92 SHOW

SHOW Command

Show an object

```
  SHOW obj,x,y,image|anim|ptrn
```

If the object number is 0, the commands manipulates the graphics of the main character.

The image can be a BOB image number, an animation string, or even a pattern (PTRN) specification.

If x and y are left blank, the position of the object will not be altered, only the image.

If image is left blank, the object is moved to the new co-ordinates retaining the previous image.

Note: If the object was in the inventory before the SHOW, it is removed and the inventory is updated.

Example:

```
  SHOW 3, , ,PTRN 1
```

would show object 3 in its previous position using the animation sequence stored in the 1.ptrn file. This only works if the object was previously visible.

Another example:

```
  SHOW ROBJ1,30,70,
```

would place room object 1 at the new co-ordinates 30,70.

A third example:

```
  SHOW 0, , ,
```

refreshes the graphics of the main character. This is useful if you have loaded new global images, for instance.

See also:

HIDE
,
OMOVE
,
CPOS
,
CBOB
,
CMOVE

1.93 HIDE

HIDE Command

Hides an object

```
  HIDE obj
```

hides the specified object from view (that is, removes it from the current room). This may often be used in room DACT statements, using room flags to decide what objects are being shown and not in a particular situation or phase of the game.

See also:

CHAR
,
SHOW

1.94 OBJONTOP

OBJONTOP Command

Puts the object on top of all other displayed objects

```
OBJONTOP obj
```

Sometimes it happens that two objects partly occupy the same space on screen - one object being displayed on top op the other.

However, the topmost object being displayed properly is not, in itself, a guarantee that the mouse cursor will actually register it when you move the cursor across it. If the underlying object is further up GRAAL's internal list of objects shown in the room, it is that object's name that will be shown, which is probably not what you want. And that list was unavailable to you before GRAAL 2.

The simple remedy is this command. If, when testing your adventure, you find an object which is unavailable in the manner described above, simply give an OBJONTOP command for it once it's been placed in the room - in a DACT: statement if it is a ROOMOBJect, or right after a SHOW command, for example.

1.95 TRACK

TRACK Command

Handles soundtracker music modules

```
TRACK file|list,tempo,FILTER|NOFILTER
```

file|list

If the file name is different than the last sound tracker file name used, or no tracker file is currently in memory, the file is loaded and the module starts playing.

tempo

If tempo is set to 0, vertical blank timing will be used just as in previous versions of GRAAL.

If tempo is set to another value, it indicates the tempo in BPM (beats per minute), which gives easier and more accurate timing in most cases.

(This parameter replaces the old LOOP parameter which never did work, anyway.)

FILTER|NOFILTER

Specify FILTER if you want the Amiga's low-pass audio filter to be on while the module is playing (takes away some high frequencies and hissing noises.)

TRACK OFF

Stop the module playing temporarily.

TRACK ON

Resume playing a stopped module.

TRACK NO

Stop playing and erase the module from memory (thus freeing memory space).

TRACK CHANNELS=channels

This command determines which channels will play tracker music, and which will be kept free for sound effects. Specify a list of channels:

TRACK CHANNELS=123	will use channels 1-3 (or 0-2, depending on how you refer to them) for tracker music and keep channel 4 free for SAMPLAY etc.
TRACK CHANNELS=13	will use channels 1 and 3 for tracker, and 2 and 4 for samples.
TRACK CHANNELS=1234	will use all channels for tracker music. This is the default, and it also means samples are also set to all four channels, thus interrupting the tracker module if you try to use the sound channels for both things simultaneously.

The channel allocation will remain in effect until another TRACK CHANNELS=... command is given.

See also:
SAMLOAD
,
SAMPLAY

1.96 EFFECT:

EFFECT: Command

Pre-loads a frequently used sound effect

```
  EFFECT number;file;frequency|DEFAULT


number

This is the effect number to be used in the
                SOUND
                 command when playing
the sound effect. A maximum of 2 simultaneous effects can be loaded with
EFFECT commands, so this number must be either 1 or 2.


file

This is the file name of the file containing the IFF or raw sound sample
to be used for the effect.


frequency|DEFAULT

Specifying a frequency means the SOUND command will use this frequency.
DEFAULT means the default frequency stored in the IFF sample will be
used. In the case of a raw sample, DEFAULT means the frequency will be
8363 Hz.
```

1.97 SOUND

```
                SOUND Command

Plays a sound effect pre-loaded with the EFFECT: statement


  SOUND effect_number

Up to two sound effects can be loaded with
                EFFECT
                   commands for instant
access with this SOUND command.
```

1.98 SAMLOAD

```
                SAMLOAD Command

Loads a raw or IFF sample into memory


  SAMLOAD file|list>

This command loads a raw or IFF sample into memory for later use with
the SAM command.
```

See also:
SAM

1.99 SAMPLAY

SAM Command

Plays a previously loaded sample

SAM DEF|channel,DEF|frequency

Play a sample loaded with a previous SAMLOAD command. Note that from version 2.1, samples can't be looped. If you need a continuously looped sample, you'll have to put it in a tracker module and use the TRACK command.

DEF|channel

DEF will play the sample over the sound channels left over by the

TRACK~CHANNELS=
command. That is, if TRACK CHANNELS=24 has been used, samples will play on channels 1 and 3. If TRACK CHANNELS= has not been used, the sample will play on all channels, temporarily interrupting any music that may be playing.

channel will play the sample on a single channel as specified, regardless of whether it has been designated a sample channel or not.

DEF|frequency

DEF will make the sample play with the default frequency. To raise or lower the pitch, specify a frequency instead.

SAM OFF

Stop playing a sample.

SAM NO

Stop playing a sample and erase it from memory (thus freeing memory space).

See also: SAMLOAD

1.100 CLPART

CLPART Command

Load a clipart IFF file

```
  CLPART file

The specified picture file is loaded into memory, where it is used for
grabbing images with the BOBS command.

  CLPART OFF

Get rid off a previously loaded clipart file when it is no longer
needed.

See also:
                BOBS
```

1.101 BOBS

```
                BOBS Command

Loads BOB images into the image bank


  BOBS no,bob,x1,y1,w,h,x-offset,hotspot

The parameters for this command are exactly the same as for the
                BOBS:
                statement, except the parameters should be separated by commas (,)
instead of semi-colons (;). There is one slight difference in how you
specify the the starting BOB image number (which is the second
parameter). This should be specified as n, SBOBn, or RBOBn, depending on
the type  of images you intend to add / replace in the image bank.

This command must be preceeded by a CLPART command.

See also:
                CLPART
```

1.102 MAKE3D command

```
                MAKE3D Command

Re-creates the scaled images for a character


  MAKE3D character

Normally, the
                3D:
                 statement handles all scaling of character images
automatically.

However, if you use the
                CLPART
```

and
BOBS
commands to replace the graphics
for a character, you have to issue a MAKE3D command afterwards to create
the scaled images for the new character images.

1.103 HOTSP

HOTSP Command

Alters the hotspot of an image

```
HOTSP image,position
```

This command is used when you need to make changes to the "3D order" in which objects and images are displayed on the screen.

A hotspot position of 0 defines the default hotspot at the middle of the bottom of the image. Any other value defines another hotspot in the y direction of the image. The y direction is the important one, because it is the relative position of hotspots in the y direction that determines which image goes in front of another on the screen.

An unfortunate side-effect of altering the y hotspot is that the x hotspot position "jumps" from the middle of the image to the left edge - there is no convenient way for me to avoid this. This means that you have to redisplay the image on screen with a new SHOW, OMOVE, BOBON or other such command, and in that command adjust the x position to cancel out the effect of the hotspot having moved in the x direction as well as in the y direction.

See also:
BOBS
statement

1.104 LIGHTS

LIGHTS Command

Fade scene area out or in

```
LIGHTS ON|OFF
```

ON makes the scene area visible. OFF fades the scene area to black. A LIGHTS ON must always be present in a DACT: statement for a room, otherwise the screen will stay black and nobody will be able to do very much!

1.105 COLOUR

```
                COLOUR Command

Change a colour


  COLOUR [DLY,]colour,colour_value

The colour is changed to the new value. If you want to manipulate
colours in DACT: statements before the
                LIGHTS~ON
                 command has been
issued, begin the command with the DLY (delay) parameter. This will
cause the new colour to faded in together with the rest when the LIGHTS
ON take effect.

See also:
                FADE
```

1.106 FADE

```
                FADE Command

Fade one colour to another


  FADE colour,speed,colour_value,WAIT|NOWAIT|STACK

Fades the specified colour to the new colour value with a certain speed.
Use the STACK parameter if several colours should be faded
simultaneously - GRAAL will wait until a FADE command with WAIT or
NOWAIT specified and then also fade all STACKed colours at the same
time.

WAIT causes the action to be suspended during the colour fade. NOWAIT
means action will continue while the colours are being faded.

See also:
                COLOUR
```

1.107 CAMERA

```
CAMERA Command

Pan the camera to any part of the background picture in scene area


  CAMERA xpos_center

xpos_center is the horizontal position GRAAL tries to put in the center
of the scene area. Of course, the pan stops whenever one of the edges of
```

the background picture comes into view.

Use this command in cutscenes and the like, when you need to move the camera away from or independently of the main character.

1.108 TITLE

TITLE Command

Show a title screen

```
TITLE file,effect|CUT|FADE
```

The file is an ordinary iff picture file. The effect can be one of the following:

effect

A previous title picture is gradually dissolved into a new one using a bit pattern that depends on the number given. Odd numbers, and prime numbers in particular, are recommended. Some numbers don’t work at all!

FADE

The old picture is faded to black, then the new one is faded in.

CUT

Pictures are just swapped without any special effects. HAM screens should be handled this way.

See also:
TYPE

1.109 TYPE

TYPE Command

Type text on a title screen.

```
TYPE font,colour,x,y,SHADOW|SHADOW2|BORDER|NONE,text
```

This command is used to type text on title background screens.

font is 1 or 2, corresponding to the
TITLEFONT:
statements in the
graal.main file.

colour is the colour number

x,y is the printing position. x=-1 means the text will be centered.

SHADOW, SHADOW2 or BORDER, surrounds the text with different kinds of
shading for greater legibility. If no effect is desired, use NONE.

See also:
TITLE

1.110 TEXT

TEXT command

Display text in scene area

```
TEXT x,y,colour,text
```

This command uses the same font and pause lengths as the
SAY
,
THINK
, and

RESP
commands, but any text can be used and it is not connected to the
main character or a certain dialogue.

x,y

The text is placed centered around these co-ordinates.

If x is set to -1, the text is be centered vertically on screen, no
matter how the background is currently scrolled.

See also:
SAY
,
THINK
, and
RESP
commands

1.111 BOBON

BOBON Command

Places a BOB that is not a GRAAL object on screen.

```
BOBON bob,x,y,image
```

If you are putting a new image on the screen, first make sure the BOB number is not already in use for any object in the room.

If the BOB is already placed on screen, and x and y are left empty, only the image is changed and not the position.

If the BOB is already placed on screen, and the image number is left blank, only the BOB position changes.

See also:
BOBOFF

1.112 BOBOFF

BOBOFF Command

Take away a BOB that is not an object from the screen

```
BOBOFF bob
```

Used to take away BOBs from display that have been put there by the BOBON command.

See also:
BOBON

1.113 PBOB

PBOB command

Pastes a BOB image

```
PBOB ulx,uly,image
```

The image is pasted into the picture without anyway of removing it afterwards (unlike the BOBON / BOBOFF commands, which actually use a BOB to display an image).

Note that the coordinates in this case are the upper left corner of the image and not the hotspot position.

1.114 COMGR command

COMGR Command

Pastes a BOB image onto the command or dialogue area

```
  COMGR ulx,uly,image,S|

This is exactly like a
                PBOB
                 command for the command or dialogue area,
depending on which happens to be on screen at the time.

ulx,uly

Upper left corner of image

S

If the last parameter is S, the command areas current look will be saved
to memory. Otherwise, the pasted image is lost when switching back and
forth between dialogue and command modes.

Note: Saving the new version of the command area takes time!
```

1.115 SETDATE

```
                SETDATE Command

Sets the (game) date


  SETDATE year,month,date,weekday

Note that we are talking about the "internal game time", not the system
real time clock...

If any of the weekday, date, or month parameters are left blank, they
retain their old values.

year

Anything you wish, preferably 2 or 4 digits

month

1-12, 1 being January...

date

1-31. The GRAAL calendar can handle the normal lengths of the months,
but does not consider leap-years.

weekday

1-7, 1 being Monday and 7 Sunday. The GRAAL calendar does not check the
historical accuracy of weekday versus date, though (see above).

See also:
                SETTIME
```

and
ADDTIME
commands

1.116 SETTIME

SETTIME command

Sets the time

SETTIME hours,minutes

The time must be set in 24-hour format, regardless of whether it is presented that way or not (see
TIME_FORMAT
).

Note that we talk about the "in-game clock", not the real-time system clock here.

See also:
ADDTIME
command

1.117 ADDTIME

ADDTIME command

Advances the clock

ADDTIME hours,minutes

Added time also alters the calendar if needed. Note that this command is only meant to be used for adding minutes, hours or possibly a day or two - when jumping further in time, use the
SETDATE
command.

If
TIME_LAYOUT
or
DATE_LAYOUT
is active, the command also updates the time and/or date displays.

See also:
SETTIME
command

1.118 SAVETIME

```
                SAVETIME Command

Saves the current in-game time and date


  SAVETIME


This command is mainly here to make it a little easier to perform
operations on dates and times. Doing "maths" on dates and times manually
is not very fun, so this command lets you use the
                ADDTIME
                 command
without loosing the current time and date forever: Using
                RESTORETIME
                brings back the saved time and date.

Example: You wish to store the date and time twelve hours from "now" in
room flag 1 for room 1. This sequence of commands ought to do it.

SAVETIME;ADDTIME 12,0;SETRF 1,1=#TIME;RESTORETIME
```

1.119 RESTORETIME

```
                RESTORETIME Command

Restores a previously saved time and date


  RESTORETIME

Use this command to restore the date and time to that saved with

                SAVETIME
                .
```

1.120 NOBREAK

```
NOBREAK Cutscene Command

Disables [Esc] key in cutscenes


  NOBREAK

This can only appear as the very first statement in a cutscene, and
tells GRAAL that the [Esc] key cannot be used to skip this cutscene.
```

1.121 FINAL

FINAL Cutscene Command

Indicates resume point in cutscene

```
  FINAL
```

This can only be use in a cutscene. All commands below FINAL will be executed is the rest of the cutscene was skipped with the [Esc] key.

1.122 TRACE Command

TRACE Command

Starts or stops the single-step trace mode

```
  TRACE ON|OFF
```

This is a development function, which can also be activated and de-activated from the on-line monitor.

TRACE ON opens a trace console window at the top of the display, where each condition and command that GRAAL is about to execute is shown in green. Nothing will happen until you press a key - then the text switches to red and the condition or command is executed in the normal way. An empty console window means GRAAL is waiting for player input.

1.123 graal.main file

graal.main Statements =DEMO=>

When it comes to the order in which the statements should appear, look to the graal.main file of the "Olaf" demo and the ones in The GRAAL Herald diskmags for guidance. The most important thing is to make sure all statements requiring graphics to be present have something to work with - so the statements specifying the graphics should come before those actually using the graphics!

("Number" below: ONE means statement occurs only once. ANY means zero to any number of times.)

Statement	Number	Description
~NAME~	one	Name of the adventure
~VERSION~	one	Version number of the adventure

```
~MAX_CACHE~
        one    Maximum number of files in memory cache


~DEBUG~
            0-1    Switches on debug stuff in encrypted version


~MAX_ROOM~
         one    Maximum room number used

~MAX_SECTION~
      one    Maximum section number used

~MAX_DLG~
          one    Maximum number of dialogues

~N_DIALOGUES~
      0-1    Sets the limits for dialogues

~MAX_DACT~
         one    Maximum number of DACT statements per room

~MAX_ACTION~
       one    Maximum number of ACTION statements per room


~GLOBALOBJS~
       one    Number of global objects

~SECTIONOBJS~
      one    Number of section objects

~ROOMOBJS~
         one    Number of room objects


~N_GLOBALBOBS~
     one    Number of global BOB images

~N_SECTIONBOBS~
    one    Number of section BOB images

~N_ROOMBOBS~
       one    Number of room BOB images


~CLPART~
           any    Name of picture containing clipart graphics

~BOBS~
             any    Grab global BOB images from clipart picture


~MODE_SWITCH~
      0-1    Command/dialogue switching style
```

```
~AREA_SIZES~
       0-1    Heights of scene and command areas

~COMMAND_AREA~
     one    Name of picture with command area graphics

~DLG_AREA~
         one    Name of picture with dialogue area graphics

~RESOURCE~
         one    Name of interface resource bank


~N_VERBS~
          0-1    Number of verbs, default is 9

~VERB_ZONE~
        any    position and size of each verb "button"

~VERB_TEXT~
        any    Message when pointing to a verb


~ARROW_CURSOR~
     0-1    Image to use for arrow mouse pointer

~CROSSHAIR_CURSOR~
 0-1    Image to use for crosshair mouse pointer

~CURSOR_PALETTE~
   0-1    Colours for mouse pointer


~INV_LAYOUT~
       0-1    position and size of inventory list

~INV_UP~
           0-1    properties of inventory scroll arrow

~INV_DOWN~
         0-1    properties of inventory scroll arrow


~CUTSCENE_LAYOUT~
  0-1    position and size of inventory list


~DLG_LAYOUT~
       0-1    position and size of dialogue lines

~DLG_UP~
           0-1    properties of dialogue scroll arrow

~DLG_DOWN~
         0-1    properties of dialogue scroll arrow
```

```
~SENTENCE_LAYOUT~
  0-1    position and size of sentence display


~TIME_FORMAT~
      0-1    format of time display

~TIME_LAYOUT~
      0-1    layout of time display

~DATE_FORMAT~
      0-1    format of date display

~DATE_LAYOUT~
      0-1    layout of date display

~MONTH_TEXT~
       0-1    change names of all months

~DAY_TEXT~
         0-1    change names of all the days of the week


~SYSTEM_TEXT~
      any    change system message texts


~WALK_BUTTON~
      one    Left or right button used for walking?

~DISABLE_QUIT~
     0-1    Disables the "q" quit key

~DISABLE_SAVE~
     0-1    Disables the "s" and "l" save/load keys


~EXIT_COL~
         one    Text color of exit names

~OBJ_COL~
          one    Text color of object names


~START_ROOM~
       one    Adventure starting position


~MSGFONT~
          one    Scene area text font and size

~COMFONT~
          one    Command and dialogue area text font and size

~TITLEFONT1~
       one    Titlepage text font and size (1)
```

```
                ~TITLEFONT2~
                       one    Titlepage text font and size (2)


                ~LINE_LENGTH~
                      one    Line length for SAY, RESP, etc.

                ~NORMAL_WAIT~
                      0-1    Normal wait period for texts


                ~CHAR~
                             0-4    Specify data for alternate characters


                ~CHARACTER_HEIGHT~
                 0-1    Estimated average height of main character

                ~CHARACTER_WIDTH~
                  0-1    Estimated average width of main character


                ~CHARACTER_COL~
                    0-1    Text color of main character "speech"

                ~STILL_RIGHT~
                      one    Main character right profile image

                ~STILL_LEFT~
                       one    Main character left profile image

                ~STILL_BACK~
                       one    Main character backside image

                ~STILL_FRONT~
                      one    Main character front image

                ~PAUSE_RIGHT~
                      one    Main character pause image having walked
right

                ~PAUSE_LEFT~
                       one    Main character pause image having walked
left

                ~PAUSE_BACK~
                       one    Main character pause image having walked
away

                ~PAUSE_FRONT~
                      one    Main character pause image having walked
toward

                ~WALK_RIGHT~
                       one    Main character animation for walking right
```

```
                ~WALK_LEFT~
                        one    Main character animation for walking left

                ~WALK_AWAY~
                        one    Main character animation for walking away

                ~WALK_TOWARD~
                      one    Main character animation for walking toward

                ~WALK_SPEED~
                       0-1    Main character walking speed adjustment


                ~TALK_MAP~
                         1-8    Speech animations mapped to pause/still
images

                ~HANDLE_MAP~
                       1-8    Object manipulation animations mapped to -"-


                ~OBJECT~
                           any    Definitions of global objects


                ~DLG~
                              any    Definitions of dialogue partners *NEW*


                ~ACTION~
                           any    Actions taken for input relevant to entire
game
```

1.124 .section files

```
                n.section Statements =DEMO=>

Follow the statement order presented here in your .section files to
avoid any unnecessary trouble.

  Statement      Number Description


                ~CLPART~
                       any    Name of picture file containing clipart

                ~SECTIONBOBS~
                  any    Grab section BOB images from clipart picture


                ~SECTIONOBJ~
                   any    Define section objects
```

```
~LINE~
         any    Define dialogue lines main character can choose
        from *NEW*


~LACT~
         any    Define responses to dialogue lines *NEW*


~DACT~
         any    Actions executed directly when the section file
        is first used.


~ACTION~
       any    Actions taken for player input relevant to
        section
```

1.125 .room files

```
n.room Statements =DEMO=>
```

Please follow the order indicated here in your .room files to avoid unnecessary errors and trouble.

```
  Statement    Number Description


~UPDATE~
     one    Frame update rate *NEW*

~SECTION~
    one    Section to which room belongs

~BACKDROP~
   one    Name of background picture for room


~START_POS~
  any    Starting positions for main character


~FLOOR~
      1-12   Areas where the main character can "put its feet"


~PATH~
       0-12   Path used for navigating between floors


~EXIT~
       1-10   Exits
```

```
~CLPART~
     any    Name of picture file containing clipart

~ROOMBOBS~
   any    Grab room BOB images from clipart picture


~STATIC~
     any    Place static graphic elements on background
      picture

~ANIM~
       any    Place animated graphic elements on background
      picture


~ROOMOBJ~
    any    Define room objects


~DACT~
       any    Actions to be taken directly upon entering the
      room


~LINE~
       any    Define dialogue lines main character can choose
      from

~LACT~
       any    Define responses to dialogue alternatives


~ACTION~
     any    Actions to take for player input relevant to room
```

1.126 NAME

NAME Statement (main)

Gives the adventure name

```
  NAME: game_title
```

It’s always nice to know what you are playing, isn’t it? This is shown when the player presses "V" and also identifies saved game files.

1.127 VERSION

VERSION Statement (main)

Gives the adventure version

VERSION: version_no

This is used to make sure saved game files are compatible with the current status of your adventure - always update this when you do ANYTHING with the adventure that affects the number of rooms, objects, sections, object definitions, or any flag usage! One of the most common sources of problems when debugging is using old saved game files.

1.128 MAX_CACHE

MAX_CACHE Statement (main)

Sets the maximum number of files in the memory cache

MAX_CACHE: no_of_files

For normal use: Set to 0 when creating a game (especially if you are using the on-line debugger to reload altered scripts).

Set to 100 once the game is ready to be played to eliminate disk swaps and make use of any extra memory you may have.

When GRAAL detects that extra memory is available, it calculates how many files it will be able to fit into RAM, thus reducing disk access during gameplay. GRAAL calculates an average of 50K per file - if this is totally wrong (and don't ask me how, you will probably never have to bother), you may have to set this to a very low number or even to zero.

If you are creating a game and have MAX_CACHE set to anything but 0, graal will sometimes read old scripts from memory when you really want it to read the changes you just saved to disk. That is why you should set this to 0 during development.

1.129 DEBUG

DEBUG: Statement (main)

Switches on debugging tools & stuff in encrypted version of the game

DEBUG:

Note: This is only of use to registered users.

Sometimes, you have to debug problems to do with the organisation of files on the delivery diskettes, and you need to do it AFTER having used the encrypt/compress option in GPRO. Just have this statement in graal.main (BEFORE running it through GPRO, mind you!). This makes it

possible to call up the monitor, and it also leave s the "looking for
file ..." message in the diskette switching requester. Once everything
is OK, take away DEBUG: in the source graal.main, and make another set
of diskettes by running GPRO and GDC again...

1.130 NTSC_TIMING: Statement

NTSC_TIMING: Statement

Makes pauses equally long on NTSC and PAL systems

```
NTSC_TIMING: YES|NO
```

While most European countries use the PAL standard for T.V. broadcasts,
U.S.A. and other countries use NTSC. NTSC has a lower resolution but a
higher refresh rate, making for a more stable picture. This means that
depending on the system used, timing operations that depend on waiting
for vertical blank periods (that is, the periods between drawing a
complete T.V. picture) will have different lengths depending on the
system. A pause for 1/50th of a second on a PAL system will be a 1/60th
second wait on an NTSC system.

This command makes all pauses handled by the GRAAL pause function
equally long on NTSC and PAL systems. That is, if NTSC_TIMING: is set to
yes, the length of each pause will be multiplied by 1.2.

Among the affected commands are W(ait), SAY, THINK, RESP and TEXT.

Note 1: Animations are NOT affected by this statement: they will still
run faster under NTSC.

Note 2: To make the playback speed of tracker modules equal in PAL and
NTSC, use the new tempo parameter of the
TRACK
 command.

1.131 ARROW_CURSOR:

ARROW_CURSOR / CROSSHAIR_CURSOR Statements (main)

Changes the image of the mouse pointer

```
ARROW_CURSOR: image;hotspotx;hotspoty
CROSSHAIR_CURSOR: image;hotspotx;hotspoty
```

image

a normal image number - you must grab the image to be used using a BOBS:
statement first.

Note that the images to be used as mouse pointer shapes must be drawn i
lowres, and in four colours (2 bitplanes) only. Also, the image must be
exactly 16 pixels wide. (Actually, the BOBS: statement should read 17
pixels, which will actually pick up 16 - one of life's little
mysteries.)

hotspotx;hotspoty

This sets the "sensitive point" of the cursor image, counted in pixels
from the upper left corner of the image.

See also:
CURSOR_PALETTE
statement

1.132 CURSOR_PALETTE:

CURSOR_PALETTE Statement (main)

Sets the colours to use for the mouse pointer in the command area

```
CURSOR_PALETTE: rgb;rgb;rgb
```

The mouse pointer uses three colours, except for its first colour (0),
which is regarded as transparent.

The colours specified here are used in the command and dialogue area.
Each colour value is given as a a red, green, and blue component value
in hexadecimal.

FFF means white 000 means black 888 means grey 550 means dark yellow
(some red + some blue) 0FF means bright cyan (all green + all blue), and
so on...

In the scene area, colours 1, 2, and 3 of the cursor will be translated
into colours 17, 18, and 19 of the backdrop palette.

1.133 INV_LAYOUT

INV_LAYOUT Statement (main)

Controls the layout of the inventory list

```
INV_LAYOUT: x1;y1;x2;y2;rows;cols;TEXT|ICONS;
            VERTICAL|HORIZONTAL;ink/image_no;bg
```

The first four parameters determines the size and position of the box

containing the inventory list. "rows" and "columns" determines how many rows and columsn there are.

TEXT|ICONS

determines whether text or icons will be used for the objects in the inventory list

VERTICAL|HORIZONTAL

determines whether the list scrolls vertically (top to bottom) or horizontally (left to right).

ink/image_no

If the inventory display is TEXT, this is the ink colour.

If the display is ICONS, this is the image number to be used for an "empty space" in the inventory display. For example, if all your inventory icons have a border, this image should be a border with nothing in it - it kind of helps fill out the display...

When a text inventory is specified, it is assumed there may be a border (1 pixel high, 2 pixels wide) around each "cell" in the inventory display: This means GRAAL does not erase the edges of the "cell".

bg

specifies the colour to use for the background colour.

If no INV_LAYOUT statement is given, the following is assumed:

INV_LAYOUT: 284;19;634;60;3;2;TEXT;VERTICAL;7;8

which corresponds to the GRAAL built-in command area (the one used if COMMAND_AREA: DEFAULT is specified).

1.134 INV_UP

INV_UP / INV_DOWN / DLG_UP / DLG_DOWN Statements (main)

Sets the properties of the arrows used to scroll the inventory and the dialogue lines

```
INV_UP: x;y;image1;image2
INV_DOWN: x;y;image1;image2
DLG_UP: x;y;image1;image2
DLG_DOWN: x;y;image1;image2
```

x;y

is the top left hand corner of the aroow of other symbol used to
indicate the list can be scrolled (up or down, depending on which
statement we're talking about)

image1

is the image used when the function is available

image2

is the image used when the function is unavailable

The images must be global. The default statements are as follows:

```
 INV_UP: 265;18;12;10;3;5
 INV_DOWN: 265;48;12;10;4;5

 DLG_UP: 8;8;12;10;3;5
 DLG_DOWN: 8;38;12;10;4;5
```

As you see, by default the inventory and dialogue displays use the same
symbols (up and down arrows), and all statements use the same image2.
This is possible because the "not available" symbol is just a piece of
background, erasing the unavailable arrow(s) completely.

1.135 DLG_LAYOUT

DLG_LAYOUT Statement (main)

Determines the layout of the dialogue area

```
  DLG_LAYOUT: x1;y1;x2;y2;rows;ink;bg;inkhi
```

x1;y1;x2;y2

defines the "box" containing the dialogue lines.

rows

determines the numer of lines shown at the same time. The height of the
box is divided into this many "cells"

ink;bg;inkhi

sets the text and background colours. inkhi is the highlighting colour
used when the mouse cursor passes over a sentence.

When handling the dialogue lines, GRAAL assumes there may be a border 1
pxel high and 2 pixels wide around each "cell" in the list, and

therefore does not erase the edges of the cell.

1.136 CUTSCENE_LAYOUT

CUTSCENE_LAYOUT Statement (main)

Determines size, position and image for cutscene indicator

CUTSCENE_LAYOUT: x1;y1;x2;y2;bg;ulx;uly;image

This is used by the
CUTSCENE
command (with the "S" or "F" parameter)
to place the cutscene indicator on the command area.

x1;y1;x2;y2;bg

defines the area in the command area which should be "blanked out"
before placing the indicator itself, an its colour. (A larger area
"eats" some memory, because the overlaid graphics have to be stored
elsewhere for the duration...

bg

is the background colour

ulx;uly

is the top left corner of the indicator image: note that x1;y1;x2;y2 can
define a larger or different area to erase - the area does not have to
be exactly that which is covered by the actual cutscene indicator
image.

image

is the bob image containing the actual indicator.

If this statement is not in the graal.main file, the following is used:

CUTSCENE: 5;18;255;62;8;122;24;6

1.137 SENTENCE_LAYOUT

SENTENCE_LAYOUT Statement (main)

Determines the size and position of the sentence display area

```
  SENTENCE_LAYOUT: x1;y1;x2;y2;ink;inkhi;bg
```

This statement sets the box where the constructed sentence is built and displayed.

x1;y1;x2;y2

are the corners of the area of the sentence box.

It is assumed a border is included in the area (1 pixel high, 2 pixels wide), which means the edges of the specified area will not be erased by GRAAL. The text will be centered at the top of the area.

ink;inkhi;bg

sets the text, highlighted text, and background colours.

1.138 TIME_FORMAT

TIME_FORMAT Statement (main)

Determines how the time is shown

```
   TIME_FORMAT: format_string;am_text;pm_text
```

The time can be shown either permanently in the command area (using

TIME_LAYOUT
), or in a text in the scene area (using the special variable #TIME in a TEXT, SAY, THINK, or RESP command).

This statement determines how it is shown (excluding the ANALOGUE display possible with TIME_LAYOUT - see that for more info.)

format-string

This is a string of characters. The following special characters may appear:

#12

the hours will be placed here in 12-hour format

#24

the hours will be placed here in 24-hour format

#MM

```
the minutes will be placed here

#AM

the am/pm text will be placed here


Examples:

TIME_FORMAT: #12:#MM #AM;am;pm

may give results such as "3:35 am" and "6:00 pm"

TIME_FORMAT: #24:#MM

may give results such as "1:30" or "15:37"
```

1.139 TIME_LAYOUT

```
                TIME_LAYOUT Statement (main)

tells GRAAL to show the time in the command area


  TIME_LAYOUT:
DIGITAL|ANALOGUE;x1;y1;x2;y2;ink;bg[[;ax1;ay1;ax2;ay2;ink;bg];font]

If this statement is present in the graal.main script, the time will be
permanently shown in the command area. It will be automatically updated
when needed.


DIGITAL|ANALOGUE

 "DIGITAL"  will show the time in figures and text using the format
            specified in
                TIME_FORMAT
                 "ANALOGUE" will draw the hands of an analogue clock


x1;y1;x2;y2

This is the area where either the text or the clock hands are drawn. If
ANALOGUE is chosen, the width of the rectangle should be double the
height to achieve a circular clock face on a hires command area.

In DIGITAL mode, the text will be centered at the top of the rectangle.

In ANALOGUE mode, the area is NOT erased when the hands are redrawn -
only the old positions of the hands are erased. This means you can draw
the rest of the clock face around the hands in the command area backdrop
picture - just make sure the hands do not pass over any of your
graphics.
```

ink;bg

sets colour and background for the "digital text" or the clock face.

ax1;ay1;ax2;ay2

This can only be used together with ANALOGUE and defines a second rectangle in the command area where the "am/pm" texts are shown to complement the information in the analogue clock. If you do not want to display this information, just leave the last 7 parameters out. (That is, the last "font" parameter is also left out, because you do not need to specify a font for the display of the clock face...)

ink;bg

sets the text and background colours for the "am/pm" text display for an analogue clock (if this is included)..

font

This is the font used for the "DIGITAL" time display, or the "am/pm" display complementing the "ANALOGUE" clock. A number between 1 and 4 is expected:

```
 1 is TITLEFONT1
 2 is TITLEFONT2
 3 is MSGFONT
 4 is COMFONT
```

Examples:

TIME_LAYOUT: DIGITAL;10;8;90;24;1;0;4

This prints the time in text format with ink 1 and bg colour 0 in the area 10;8;90;24. The font is font 4 (COMFONT).

TIME_LAYOUT: ANALOGUE;500;10;580;50;3;6

This prints the hands of a clock in the area 500;10;580;50 with ink 3 and bg colour 6.

TIME_LAYOUT: ANALOGUE;500;10;580;50;3;6;500;56;580;70;1;0;3

As above, but goes on to print the am/pm text in the area 500;56;580;70 with ink colour 1, bg colour 0, and font 3 (MSGFONT).

1.140 DATE_FORMAT

DATE_FORMAT Statement (main)

Sets the date display format

```
  DATE_FORMAT: format_string

The date can be displayed either permanently in the command area (using
the
                DATE_LAYOUT
                 statement) or in the sentence area using the special
variable #DATE in a SAY, THINK, TEXT or RESP statement.

This statement determines how the date will be presented.

format_string

In this string, the following special characters will be replaced by
"date data"

#Y

is replaced with the year

#M

is replaced by the number of the month without a leading zero

#0M

is replaced by the number of the month with a leading zero

#N

is replaced by the name of the month (is seldom used in the same string
as "M", obviously)

#D

is replaced by the date without a leading zero

#0D

is replaced by the date with a leading zero

#W

is replaced by the weekday


All other characters in the string is kept as is.

Examples:

DATE_FORMAT: #Y-#0M-#0D

may give "1996-08-01"

DATE_FORMAT: #M/#D/#Y
```

```
may give: "8/1/96" (two or four digits in the year simply depends on
what you set the year to - see SETDATE command)

DATE_FORMAT: #N #D, #Y

may give "August 1, 1996"

DATE_FORMAT: #W, #N #D

may give "Saturday, August 1"



Note that the names of months and weekdays can be changed using the

                MONTH_TEXT:
                 and
                DAY_TEXT:
                 statements.
```

1.141 DATE_LAYOUT

```
                DATE_LAYOUT Statement (main)

tells GRAAL to show the date in the command area


  DATE_LAYOUT: x1;y1;x2;y2;ink;bg;font

If this statement is present in the graal.main script, the date will be
permanently shown in the  command area. It will be automatically updated
when needed. The display format is decided by the
                DATE_FORMAT
                statement.


x1;y1;x2;y2

This is the rectangle containing the date. The text will be centered at
the top of the rectangle.


ink;bg

sets the text and background colours


font

is a number between 1 and 4:

 1 is TITLEFONT1
 2 is TITLEFONT2
 3 is MSGFONT
```

4 is COMFONT

1.142 WALK_BUTTON

WALK_BUTTON Statement (main)

Sets the mouse button used for walking.

WALK_BUTTON: LEFT|RIGHT

The setting determines if you can use the left or right mouse button to command the main character to walk to any spot in the room (that is, click anywhere that isn't an object or an exit).

1.143 DISABLE_QUIT

DISABLE_QUIT Statement (main)

disables the standard "q" quit key and function.

DISABLE_QUIT:

This statement has no parameters. When found in the graal.main file, it disables the use of the "q" quit key to quit the game. You shouldn't do this until you have implemented and tested your own quit function, which probably uses a direct verb (see the VERB_TEXT: statement) and the
QUIT
command.

1.144 DISABLE_SAVE

DISABLE_SAVE Statement (main)

Disables the "s" and "l" keyboard keys

DISABLE_SAVE:

Use this if you have coded your own save/load routines and wish to disable the built-in save/load requester - but not before you have tested your own routines!

See also: The
SAVE
command for info on coding your own save/load routines.

1.145 N_VERBS

```
                N_VERBS Statement (main)

Sets the number of commands (verbs) used in the player interface


  N_VERBS: no_of_verbs

The default number of verbs is 9, not counting verb 0 (go to), which is
always present but not shown to the player.

If you do use this statement, there must also be a
                VERB_ZONE:
                 statement
for each and every verb you plan to use - even if some of the verbs
match the default positions, GRAAL will not know where to place them
otherwise!
```

1.146 VERB_ZONE

```
                VERB_ZONE Statement (main)

Sets the position and size of a command (=verb) "button".


  VERB_ZONE: verb number;x1;y1;x2;y2;image1;image2;quickkey


If VERB_ZONE statements are to be used at all, or
                N_VERBS
                 changed from
the default 9, one
                VERB_ZONE:
                 and one
                VERB_TEXT:
                 statement must be
present for each verb you want to make available in the command area.


verb number

is the number of the verb this zone belongs to


x1;y1;x2;y2

defines a rectangle (top left and bottom right corner, as always when
specifying areas this way).



If you don't want to use graphics to highlight the verb zones, set both
the following parameters to 0.
```

image1

This is a BOB image that will overlay the VERB_ZONE area as soon as the cursor moves over it.

image2

This is a BOB image that will overlay the VERB_ZONE area when the command has been clicked.

The images used must be cut out to be exactly the same size as the corresponding verb zone - an easy task if you use the <Area> button in the GRAAL Editor.

quickkey

A letter or number corresponding to a keyboard key (typed in uppercase). When the player presses the key, GRAAL acts as if the command had been clicked with the mouse. If you don't want a keyboard key ahortcut for the command, leave this parameter empty.

1.147 VERB_TEXT

VERB_TEXT Statement (graal.main)

Contains the message shown when using a verb, and also whether a verb is a direct verb or not.

```
VERB_TEXT: verb_no;[$]text
```

This is the text that appears in the sentence area when you use a command or an exit. Texts 0-9 and 999 have default values in English, but all of them can be translated into any language using this statement, and all of them (except a few) can also be changed to a completely different verb.

If the text is prefixed with a dollar sign ($), the verb is treated as a direct verb. A direct verb does not use objects - it is executed immediately when the player clicks it. This can be useful for making special functions available in the command area, such as speical forms of QUIT, or a RESTART command - although the possibilities do no end there.

These are the default values:

```
 0 - Go to            (Meaning can not be changed)
 1 - Give             (Meaning can not be changed)
 2 - Pick up
```

```
 3 - Use              (Meaning can not be changed)
 4 - Open
 5 - Talk to
 6 - Push
 7 - Close
 8 - Look at
 9 - Pull


 999 - to  (This is the preposition text for command 1, "Give")
```

And here's why you cant change the meaning of 0,1, and 3:

```
0 - This is not a command shown in the command area, but rather what
    happens when you click an exit in the scene area - thus, it must
    always have the meaning "go to" (although you can translate THAT
    into any language using this statement!)

1 - This command is always suffixed with the preposition " to ".
    To alter the " to ", use VERB_TEXT: 999;newtext

3 - This command makes use (no pun intended!) of the preposition
    defined for an object, which is what makes it possible to make
objects
    interact with each other (or not).
```

Of course, some of the other commands are also hard to think of a better replacement for - how are you going to engage in conversations without a "Talk to" command, for instance? Or perhaps you are not planning to - that's also up to you!

1.148 MONTH_TEXT

```
                MONTH_TEXT Statement (main)
```

sets the names of all the months

```
  MONTH_TEXT: name1;name2; ... ;name12
```

This is the name of the month that can be used when displaying a date. The default names are "January","February", ... , "December"

See also:

```
                DATE_FORMAT
                 and
                DATE_LAYOUT
                 statements
```

1.149 DAY_TEXT

```
                DAY_TEXT Statement (main)

sets the names of the days in the week


  DAY_TEXT: name1;name2; ... ;name7

These are the names that can be used when displaying a date. The default
names are "Monday","Tuesday", ... , "Sunday"

See also:
                TIME_FORMAT
                 and
                TIME_LAYOUT
                 statements
```

1.150 SYSTEM_TEXT

```
SYSTEM_TEXT: Statement (main)

changes a system message text


  SYSTEM_TEXT: message_no;text

This is of use for translators to foreign languages only (like my own,
for instance).

Try to keep the text about the same length as the English original.
These are the default texts (the quotes must be included in the
statement):

  1  "Please insert disk "
  2  "Select a saved game slot"
  3  "Saved game description"
  4  "Load"
  5  "Save"
  6  "Back"
  7  "OK"
  8  "Cancel"
  9  "Change"
 10  "   There are no saved games on this disk."
 11  "Do you want to use this disk for saved games?"
 12  "****** GAME PAUSED ******\Press any key to continue"
 13  "You are playing"
 14  "running under "
 15  "Please insert save disk into DF0:"
 16  "Please write-enable the disk!"
 17  " Yes"
 18  "Music & Sounds off"
 19  "Music & Sounds on"
 20  "  Speech speed:"
 21  "Slow         Fast"
```

1.151 EXIT_COL

EXIT_COL Statement (main)

Specifies color of exit names shown in scene area

```
  EXIT_COL: ink
```

ink is the number of the colour.

Make this a fairly bright colour - the text will be surrounded by a
black outline.

1.152 OBJ_COL

OBJ_COL Statement (main)

Specifies the colour of object names displayed in the scene area

```
  OBJ_COL: ink
```

ink is the number of the colour.

Make this a fairly bright colour - it will be surrounded by a black
outline.

1.153 START_ROOM

START_ROOM Statement (main)

Specifies the starting position for the adventure

```
  START_ROOM: room;entrance
```

The action will commence in this room and at this entrance - also see
the .room
START_POS
statement.

1.154 MAX_ROOM

MAX_ROOM Statement (main)

Highest room number used in this adventure

```
   MAX_ROOM: n_rooms
```

All rooms in an adventure are numbered from 1 and upwards - try not to leave "holes" in the room sequence, since each room number, used or not, takes up valuable memory space! If the adventure is split up onto several disks, there is no way for GRAAL to automatically know how many rooms there actually is, so this statement must be updated continually as more rooms are added to the game.

Tip: If you delete a sequence of rooms in the middle of the game, re-use those vacant room numbers if you add more rooms later on - rather than increasing the highest room number.

1.155 MAX_SECTION

MAX_SECTION Statement (main)

The highest section number used in the adventure

```
  MAX_SECTION: n_sections
```

Like MAX_ROOM, this must be manually updated with the highest section number used so far during development. Always start with section 1 and continue upwards without leaving "holes" in the numbering sequence if you can avoid it.

1.156 MAX_DACT

MAX_DACT Statement (main)

Sets the maximum number of DACT statements that can be used in a room file

```
  MAX_DACT: n_DACTs
```

This is the maximum number of DACT statements used in any room script. Set to 50 or so initially.

1.157 MAX_DLG:

MAX_DLG Statement (main)

Sets the highest dialogue number that can be used in the game.

```
  MAX_DLG: dlg_no
```

There has been a radical change in dialogue set-up since the 2.0 beta

version:

Before, the DLG: statement containing the highest dialogue number set the space needed for dialogue data. This was possible (but not very well thought out) because each dialogue required its own

```
DLG:
```

statement in the graal.main script.

From 2.0 and onwards, DLG: statements merely define "speaking partners" - that is, the partners referred to in RESP commands.

Thus, the MAX_DLG: statement is now needed to set the highest number of any dialogue used in the game. If it is not present in the graal.main script, space for 30 dialogues is reserved by default.

What does this mean for pre-2.0 scripts? Well, if you have more than 30 dialogues, you must add a MAX_DLG: statement. If you have 30 dialogues or less, you don't have to do anything! There is still no law against having one partner defined for each dialogue, it's just that with the new possibilities of using dialogues for other things than person-to-person conversations, it's a bit unnecessary at times.

See also:

```
DLG:
 and
N_DIALOGUES:
 statements.
```

1.158 N_DIALOGUES

```
N_DIALOGUES Statement (main)
```

Sets the limits for dialogues

```
N_DIALOGUES number;lines;actions
```

number

is the number of simultaneously available dialogues - that is, the maximum number referred to at any time by the current section and the current room scripts. For example, if a section script defines 2 dialogues and a room script for that section contains 5 dialogues, number would have to be set to 7, since all these dialogues must be handled by GRAAL simultaneously.

lines

is the maximum number of LINE: statements used in a dialogue

actions

is the maximum number of LACT: statements used in a dialogue

A recent addition to the setup of space for dialogues is the
MAX_DLG:
statement, which sets the total number of dialogues for the entire ↩
game.

1.159 MSGFONT

MSGFONT COMFONT TITLEFONT1 TITLEFONT2 Statements (main)

Defines fonts and sizes for various uses

```
MSGFONT: fontname;fontsize
COMFONT: fontname;fontsize
TITLEFONT1: fontname;fontsize
TITLEFONT2: fontname;fontsize
```

The fonts (in the proper sizes) must be available in a FONTS: drawer in your development directory. Furthermore, FIXFONTS must have been used on the drawer for all the fonts to be OK.

MSGFONT is the font used for all text displayed in the scene area.

COMFONT is the font used for the input sentence display and dialogue lines.

TITLEFONT1 and TITLEFONT2 are primarily used with the
TYPE
command to
print text on title screens.

1.160 LINE_LENGTH

LINE_LENGTH Statement (main)

Determines the line length of displayed sentences (SAY, RESP, and other commands.)

```
LINE_LENGTH: line_length
```

GRAAL does automatic line breaks in long sentences - this is the length it aims for for each line in SAY, RESP, and other similar commands.

GRAAL will only break lines in between words, and does not hyphenate.

You may control line breaks manually in any sentence - just insert backslash (\ ) characters where you want line breaks to appear. This will override the setting of the LINE_LENGTH: statement.

1.161 NORMAL_WAIT

NORMAL_WAIT Statement (main)

Sets the default display time for text and sentences in scene area

```
  NORMAL_WAIT: time
```

This value is used in a formula calculating for how long a text or spoken sentence should be displayed in the scene area. The formular also incorporates the overall text length and the number of lines it is broken into. Also, character sentences are shown for a slightly shorter period of time than other characters speech, because the contents of a "SAY" sentence uttered by the main character is often known wholly or partly by the player beforehand.

The bigger the number, the longer the pause. Default is 100. If this is not enough, try other nice, round numbers like 200 or 400..

1.162 MODE_SWITCH

MODE_SWITCH Statement (main)

Decides how a switch between command and dialogue mode is performed.

```
  MODE_SWITCH: ROLL|INSTANT
```

ROLL

The command area rolls off the bottom of the screen, and the dialogue area rolls in from where the command area disappeared. (And vice versa.)

This is the default, and what GRAAL 1 used.

INSTANT

The command area disappears instantly, and the dialogue area is created "on the spot" to replace it. (And vice versa.)

("Instantly" does not mean "instantly" on slower machines - the screens are compressed in memory and takes a few moments to decompress when used.)

1.163 SPLIT_LINE

AREA_SIZES: Statement (main)

Sets the height of the scene area and the command area.

```
AREA_SIZES: scenearea;comarea
```

This statement allows you to vary the size of the screen set aside to the scene area and the command/dialogue area. The numbers are expressed in pixels; default is 120 for the scene area and 80 for the command area.

NOTE: If you want NTSC machine users to be able to run your adventures, you should make sure the sum of the values is not higher than 200. "Pal-only" adventures should not use a sum bigger then 256.

1.164 COMMAND_AREA

COMMAND_AREA / DLG_AREA Statements (main)

Graphic files containing the graphics for the command area and dialogue area

```
COMMAND_AREA: file|DEFAULT;BUFFERED|NORMAL
DLG_AREA: file|DEFAULT
```

These files contain the graphics for the command area and its replacement during dialogues, the dialogue control area.

file|DEFAULT

If DEFAULT is specified as filename, a new file will not be loaded - the default graphics present in GRAALs memory will be used instead, which speeds up the loading time.

BUFFERED|NORMAL

When you alter the graphics in the command area, it may flicker a little. If you specify the BUFFERED parameter, the flicker will be eliminated. However, screen updates will be slower, and it takes more memory.

Make sure the height of your images match the values set in the

AREA_SIZES:
statement.

1.165 RESOURCE

RESOURCE Statement (main)

Name of interface resource bank

RESOURCE filename

This must be an Amos Pro resource bank especially designed for GRAAL. It controls the graphic appearance of the disk swapping, save/load and quit dialogue boxes, among other things. Experienced Amos Pro users can also easily use the Amos resource editor to make their own banks, but this will not be explained here.

Registered users get several different ready-made resource banks with different looks: High-tech, Stone-age, Medeival...

Consider customizing the quit, save and load routines - see the
SAVE
and

QUIT
commands for more information. The only requesters you cannot hide from the player are the ones connected to disk swapping when playing from diskettes.

1.166 GLOBALOBJS

GLOBALOBJS / SECTIONOBJS / ROOMOBJS Statements (main)

Sets the number of objects that can be used in the game

GLOBALOBJS: n_objects
SECTIONOBJS: n_SOBJs
ROOMOBJS: n_ROBJs

These statements decide how many objects of each time GRAAL will make room for. You can alter any of the values any time during the development, so there is no need for accurate estimates right away.

Objects are all the objects that have a name within the adventure and thus can be manipulated by the user.

It would be an unnecesary waste of memory to have the data for all objects in memory all the time, which is why they are divided into three categories:

GLOBAL OBJECTS are indeed available all the time. Everything that can be carried in the inventory over more than one section of the game, and all characters that Olaf can have conversations with must be in this category. Numbering of global objects start with 1 and proceeds upwards.

SECTION OBJECTS can only exist within one particular section of the game. Note that if a player leaves the section and re-enters it at a later date, all information in object flags and the like has been lost - all object will be re-initialised with the status and position defined in the OBJECT: stetements in the .section file. Therefore, use this with caution! To refer to a section object, use SOBJn, where n is the number of the section object.

ROOM OBJECTS are restricted to the current room only, and should preferrably be objects which can not be "seriously" manipulated by the user - usually, they are only there to add a bit of atmosphere and to act as red herrings. The torch in the bar in Olaf 1 is a perfect example of such an object. To refer to a room object, use ROBJn, where n is the number of the room object. Like section objects, the flags for room objects are reset each time the player leaves and re-enters the room, so if anything concerning room objects needs to be saved, it must be stored in room flags instead.

1.167 GLOBALBOBS

GLOBALBOBS / SECTIONBOBS / ROOMBOBS Statements (main)

Sets the number of BOB image bank slots available for each BOB image category

```
N_GLOBALBOBS: n_bobs
N_SECTIONBOBS: n_SBOBs
N_ROOMBOBS: n_RBOBs
```

Object images are referred to and treated according to which of the three above categories they belong:

GLOBAL BOBS are images that are always in memory for instant access anywhere in the game - for instance, the images used for the animation of the main character. Global BOB images are grabbed by the BOBS: statement in the graal.main file and referred to by their true number. Since BOB images 1-10 are reserved for system use, the ones you normally refer to in the game start with number 11.

SECTION BOBS are grabbed with the SECTIONBOBS: statement in a .section file and remain in memory as long as the player stays in the section. They are referred to using SBOBn, where n is the number of the section BOB image.

ROOM BOBS are grabbed with the ROOMBOBS: statement in a .room file and remain in memory as long as the player stays in the room. They are referred to using RBOBn, where n is the number of the room BOB image.

The numbers set in these statements may be altered at any time during development, so don’t panic.

Note: If you need some "dynamic" image replacing, any kind of image may also be grabbed/replaced by the

BOBS

command, which has the same
parameter as these statements.

1.168 CLPART

CLPART Statement (main, section, room)

Loads an IFF picture file containing clipart into memory

```
  CLPART: file
```

GRAAL doesn't mess around with complicated picture storage formats - all graphics used in the game are "grabbed" from ordinary IFF files. This statement selects the IFF file to be using for subsequent "grabbing" with the BOBS:, SECTIONBOBS: and ROOMBOBS: statements.

1.169 BOBS

BOBS Statement (main)

SECTIONBOBS Statement (section)

ROOMBOBS Statement (room)

Grabs BOB images into the BOB image bank

```
  BOBS: number;start_bob;x1;y1;w;h;x-offset;hotspot
  SECTIONBOBS: number;start_bob;x1;y1;w;h;x-offset;hotspot
  ROOMBOBS: number;start_bob;x1;y1;w;h;x-offset;hotspot
```

This command can grab a single image or a row of images, provided they are aligned horisontally and equally sized and spaced. All three versions of the command have the same syntax. The only difference is the BOB image category they grab. BOBS are later referred to by their proper image number, SECTIONBOBS by SOBJn and ROOMBOBS by ROBJn.

number

The number of images to grab with this statement.

start_bob

The first image to grab will get this number. If more than one image is grabbed, the number will be assigned in increasing sequence. E.g.,

BOBS: 4;11;... would grab the global BOB images 11, 12, 13 and 14.

ROOMBOBS: 1;5;... would grab ROBJ5.

SECTIONBOBS: 3;2;... would grab SBOB3 and SBOB4.

x1;y1

Imagine the clipart being cut out of the picture (previously loaded with the CLPART: statement) by placing a rectangular frame over the picture and cutting along the edges, x1;y1 is the co-ordinate in the upper left corner of the frame...

w;h

...and this is the width and the height of the frame. Everything that is cut out and is of colour 0 will be transparent when the image is used.

x-offset

If more than one image is grabbed with the statement, this number tells how many pixels to the right the "frame" should be moved before cutting out the next image.

hotspot

The hotspot decides which point of an image is actually placed at the co-ordinates of a command using the image. For example,

```
  BOBON 10,30,70,SBOB3
```

is a GRAAL command placing image SBOB3 at the co-ordinates 30,70. Great, but which point of the image is actually placed at 30,70? That is decided by the hotspot. The default hotspot in graal (choosen by setting the hotspot parameter to 0) is in the middle at the bottom of the frame - which is where a character grabbed as an image usually should have its feet. Setting the hotspot parameter to another value retains the x-position of the hotspot but alters the y-value. This has to do with getting objects in 3D scenes in the correct order and is explained in more detail in the GRAAL tutorial.

1.170 CHAR

CHAR: Statement (main)

Defines alternate characters

```
  CHAR:
number;obj;stimg;eimg;floor;ink;height;text_offset;width;walkspeed
```

This statement is used to define the characters in games that allows the player to switch control between multiple characters.

Whenever the character switching features are to be used, a CHAR: statement for character number 1 MUST be present, because character 1 is always the character which defines the graphics routines for all character animation, and it is always the character GRAAL puts under player control at the start of the game.

number

A number between 1 and 4, thus allowing up to 4 controllable characters in the game.

obj

Each controllable character must be connected with an object (even character 1). The starting room of the object becomes the starting room of the character, because all characters are just normal objects as long as control isn't switched to them.

All character objects must

* be global objects
* be VIS (visible)
* be NPICK (non-pickable).
* have a unique BOB number not used for anything else in the sections of the game where the character may appear
* have a unique animation channel (3-15) not used for anything else in the sections of the game where the character can appear

Obviously, it's nice if the object number matches the character number, but this is not at all necessary.

stimg;eimg

This defines a range of images used by this character for its default graphics.

All controllable characters must use the same animation strings as the default character, and the images must be stored in the image bank with the same relationships between each other.

For example, imagine all graphics for the default character that are used in STILL_...:, WALK_...:, HANDLE_MAP: and TALK_MAP: statements occupy global BOB images 11-50 (40 images):

```
  CHAR: 1;3;11;50;...
```

(Character 1 is object 3, and uses images 11-50 for default graphics. 50-11+1 = 40 images.)

There must be an identical set of images for character 2, the alternate character, only stored in a different area iof the image bank, of course:

```
  CHAR: 2;5;51;90;...
```

(Character 2 is object 5, and uses images 51-90 for default graphics. 90-51+1 = also 40 images!)

Now, image 51 should show character 2 in the same pose as image 11 shows

```
character 1, image 12 should show an identical pose to image 52, and so
on.

NOTE: The images for character 1 must always have lower numbers than the
images for any other character(s).


floor

Most data about each character is fetched from the object definition.
However, there is nothing about the initial floor number in the object
definition, so this must be specified here. (Compare to the
                START_POS:
                statement in .room scripts, which also must give the initial floor
number...)


ink

Specifies the colour of the character's text display for SAY and THINK
commands


height;text_offset;width

These parameters correspond to the
                CHAR_HEIGHT:
                 and
                CHAR_WIDTH:
                statements.


walk_speed

This parameter corresponds to the
                WALK_SPEED:
                 statement.


MEMORY-SAVING TIP:

If you have various alternate characters that don't appear in the same
sections of the game, you can preserve valuable memory by letting them
use the same image numbers - then load the correct set of images for the
current "buddy" character with CLPART and BOBS commands in the section
files!
```

1.171 SELECT_CHAR: statement

```
                SELECT_CHAR: Statement

Decides whether controlled character is registered by the cursor.


  SELECT_CHAR: YES|NO
```

```
This statement is only relevant if you use
                CHAR:
                 statements.

If it is set to NO, the character under the players control will not be
regarded as an object by the cursor. This allows you to move the
character more precisely because you can clikc inside the character
image to move it.

Default is YES, which means the currently controlled character can be
handled just like any other object when constructing sentences.
```

1.172 CHARACTER_HEIGHT

```
CHARACTER_HEIGHT / CHARACTER_WIDTH Statements (main)

Gives the average size of the main character


  CHARACTER_HEIGHT: pixels[,text_offset]
  CHARACTER_WIDTH: pixels

This states how big you main character normally is, and is used to
calculate how the main character may move on the screen.

When the player clicks a point in the scene area, GRAAL normally takes
the character height into account and tries to place the centre of the
character at that point. If you wish the feet to be placed at the
clicked point instead, you must specify the character height as 0
pixels. However, this also affects the position where the text in SAY
and THINK commands is placed. To compensate, you must give a number in
the text_offset parameter.

CHARACTER_HEIGHT: 50

The character is about 50 pixels high. Text will be placed immediately
above its head, when ordered to move GRAAL will place the centre of the
character image at the clicked point (if possible).

CHARACTER_HEIGHT: 0;50

The characters feet wil end up at the clicked position (or as close as
possible), and the text will be displayed at the same height above the
character as in the first example.

CHARACTER_HEIGHT: 50;10

This works like the first example, but we want some extra space between
the character's head and the messages.
```

1.173 CHARACTER_BOB

CHARACTER_BOB Statement (main)

Sets the BOB number used for the main character

```
  CHARACTER_BOB: number
```

Do not alter this! (But some time in the future there may be a good reason to be able to customise it.)

1.174 CHARACTER_COL

CHARACTER_COL Statement (main)

Sets the colour to use for main character "speech"

```
  CHARACTER_COL: ink
```

Make this a fairly bright colour, since it will be surrounded by a black outline. It should also a colour that is the same for all graphics in the adventure, because it does not look good if the main characters speech keeps shifting its colour from dialogue to dialogue.

1.175 PAUSE_RIGHT

PAUSE_RIGHT / PAUSE_LEFT / PAUSE_BACK / PAUSE_FRONT Statements (main)

Sets the images used when the character pauses in one of the four main directions

```
  PAUSE_RIGHT: image
  PAUSE_LEFT: image
  PAUSE_BACK: image
  PAUSE_FRONT: image
```

These images are used when the main character pauses waiting for player input. You may choose to use exactly the same images as for the corresponding STILL_ statements.

1.176 STILL_RIGHT

STILL_RIGHT / STILL_LEFT / STILL_BACK / STILL_FRONT Statements (main)

Sets the still images used for the main character and the four main directions.

```
  STILL_RIGHT: image
  STILL_LEFT: image
  STILL_BACK: image
  STILL_FRONT: image

These images will be used in automatic main character movement.
```

1.177 WALK_RIGHT

```
                WALK_RIGHT / WALK_LEFT / WALK_AWAY / WALK_TOWARD Statements (main)

Defines the animation sequence used for movement in the four
directions.


  WALK_RIGHT: anim
  WALK_LEFT: anim
  WALK_AWAY: anim
  WALK_TOWARD: anim

These four
                animation~sequences
                 are used by GRAAL for all automatic
movement of the main character.
```

1.178 WALK_SPEED

```
WALK_SPEED Statement (main)

Adjust walking speed


  WALK_SPEED: character_speed

This statement adjusts the speed of the automatic main character
movement so that the speed matches the WALK_xxxxx animation squences -
the objective is to make it look like the character actually uses his
feet to walk, rather than glide around on a slippery surface. Simply
experiment with the value until the movement (especially sideways) looks
good!
```

1.179 TALK_MAP

```
                TALK_MAP Statement (main)

Defines animation sequences used for automatic main character speech"
link "GG_Animation" 0} used for automatic main character speech


  TALK_MAP: still_image;anim
```

When a
SAY
command is given, GRAAL checks which image is currently used for the main character. (This should normally be one of the
STILL_...
or

PAUSE_...
images.) The image is checked against the TALK_MAP statements (there may be up to 8 of them) and the one where the previous image matches the main character's current image is used for the animation of the speech.

1.180 HANDLE_MAP

HANDLE_MAP Statement (main)

Defines the animation~sequences used when main character manipulates objects" link "GG_Animation" 0} used when main character manipulates objects

```
HANDLE_MAP: still_iimage;anim_lo;anim_mid;anim_hi
```

When a HANDLE command is encountered, GRAAL checks which image is currently being used for the main character, and the "handle position" for the object being manipulated. The proper animation sequence foom the HANDLE_MAP statement matching the current main character image is then used. Note that the animation sequences only show the main character reaching out for something, and that the last image in each sequence should be specified as lasting for only one frame.

1.181 OBJECT

OBJECT Statement (main)

SECTIONOBJ Statement (section)

ROOMOBJ Statement (room)

Defines an object

```
OBJECT: obj;name;room;VIS|NVIS;bob;image;x;y;cx;cy;cimage;prep;
        PICK|NPICK;anim_ch;verb;icon;LOW|MID|HIGH;types;w1;w2;w3

SECTIONOBJ: ...ditto...

ROOMOBJ: ...ditto... same syntax for all three...
```

Note that all parameters should be on the same line in the GRAAL file -

a bit difficult to show in ordinary guide format here, though. Anyway, you really should use GRAALs own object editor to be able to edit the parameters in a more user-friendly format. (See the editor documentation for this one.)

Yes, this is the most complex statement there is, but let's run through it one parameter at a time:

obj

For global objects specified by the OBJECT statement, the object is later referred to by this very number.

Objects defined by SECTIONOBJ will be referred to as SOBJn, where n is the number.

Objects defined by ROOMOBJ will be referred to as ROBJn, where n is the number.

name

The object name shown when the cursor hits the object, and in the inventory. A backslash in the name will cause a line break in that position when it is displayed in the scene area. The
NAME
command can
alter this at any time.

room

The room number where the object is initially positioned. 0 is used for objects that are "nowhere". To specify an inventory, specify "I"+ the inventory number - for example I1 means inventory 1 (the default inventory).

VIS|NVIS

VIS if the object is visible, NVIS if it is hidden.
SHOW
and
HIDE
commands can alter this as you wish later on.

bob

the bob number to use for this object. two objects (including characters and static and animated objects) cannot use the same bob number at the same time.

image

image number, animation string or pattern definition initially used for the object.

x;y

The object's position on the screen.

cx;cy

The main character's position relative to the object when manipulating it, looking at it, or plain walking up to it (with the MOBJ command).

If cx is a normal number, the offset is the character's hotspot relative to the object's hotspot. If cx is prefixed with a W (for example, "W9") the offset is the closest edge of the character relative to the object's hotspot. The edge of the character is calculated using the

CHARACTER_WIDTH: statement (or the corresponding parameter in a CHAR: statement, if you have multiple controllable character's in the game.)

This means different controllable characters will end up at the same distance from the object when using MOBJ. It is primarily when the MOBJ ends with the character facing the object with a side view.

cimage

The image used for the main character after having walked up to the object.

prep

A preposition that indicates that the object can not be used on its own, but must be combined with a second object.

More technically put, verb 3 (default USE) will look at this and if it is not a blank, will wait for a second object to be clicked, combining the two objects with this preposition.

PICK|NPICK

PICK means the object can be picked up and added to the inventory. (NPICK for churches, planet systems and other things hard to carry around.)

anim_ch

The animation channel used for the object. (Only if the object is animated, of course.) Make sure two animated objects in the same room don't try to use the same channel!

verb

This is the command verb directly executed if the player points to an object in the scene area and clicks the right mouse button

icon

If the inventory is displayed as icons, this is the image number used for the inventory display. Can be a global, room or section bob image - naturally, the image must be available at all times when there is a chance of the oject being in the inventory!

LOW|MID|HIGH

Decides whether the object is manipulated using the LOW, MID or HIGH animation (
HANDLE
command).

types

A character string, where each character specifies some property for the object. The following are suggested, but using
IFTYPE
, you can use this feature for almost anything you like!

M = Male character
F = Female character
A = Animal
G = Group
V = Alive
D = Dead
C = Container
W = Wood
T = Metal
S = Stone
L = Liquid
E = Food ... and so on ...

w1;w2;w3

When you construct sentences referring to an object, sometimes you would like to use the proper article or other word connected to the object. These parameters give you a chance to specify such words, which can the be put into your

sentences
. These can also be altered with the
NAME
command during the game.

1.182 DLG

DLG Statement (main)

Defines the dialogue partners.

DLG: partner;object;speech colour;speech offset;speech animation sequence

For each dialogue partner referred to by a RESP command,a DLG statement must be present. (This is a slight change from the rules for this statement in pre-2.0 versions. See
MAX_DLG:
for more info.)

dlg

Dialogue partner number. Must be unique for each DLG: statement. Should start from 1 and proceed upwards. This is the number used in the
RESP
command.

obj

The dialogue "partner" is defined by this object. A partner can be a global, section, or room object.

ink

The colour used for the partner's speech.

y_offset

This number determines where the partner's speech is printed. lower numbers=higher above the partner's head.

anim

The speech animation sequence used with the
RESP
command.

See also:
RESP
command.

1.183 ACTION

```
                ACTION Statement (main, section, room)

Contains conditions and commands checked when player inputs a sentence.


  ACTION: verb_no;{cond|comm}

verb_no

is the command verb number in the input sentence being checked

{cond|comm}

is any number of conditions and/or commands, separated by ";"
characters.


When the player inputs a sentence, the ACTION: statements are checked
for the currently used script files, in this order:

 room file, top to bottom
 section file, top to bottom
 graal.main file, top to bottom

When an ACTION statement with the proper verb number is found, its
parameters - the conditions and commands - are checked and executed from
left to right. As soon as a condition is FALSE, the rest of that ACTION
statement is skipped, and GRAAL looks at the next one. (There is a
special verb number, -1, for "timer events" - see the
                DOAFTER
                 command.)

The search for further ACTION statements is stopped as soon as a valid

                EXIT
                 command has been found. If a
                REDO
                 statement is encountered, the
process is restarted from the first ACTION statement in the file
currently being processed.
```

1.184 DACT

```
                DACT Statement (section, room)

Immediate actions upon entering section / room


  DACT: {cond|comm}


{cond|comm}
```

is any number of conditions and/or commands, separated by ";"
characters.

DACT:s in room scripts

Immediately after a room script has been loaded, all DACT statements in
the script are scanned for commands that should be executed before
control is returned to the player. As soon as an EXIT command is found,
any remaining DACT:s in the script are ignored.

DACT:s in section scripts

DACT:s in sections are executed immediately before the room DACT:s for
every room belonging to that section. Note that an EXIT command in a
section DACT only skips the rest of the section DACT:s - then the
execution of the room DACT:s begins.

The structure and function of DACT statements are the same as for

ACTION:
statements, except there is no verb number as first parameter here.

1.185 MAX_ACTION: statement

MAX_ACTION: Statement

Sets the maximum number of ACTION: statements in room and section
scripts

MAX_ACTION: actions

Nothing much more to say, really. Mandatory from release 2.2. Try a
value of between 100 and 200 to begin with.

1.186 UPDATE

UPDATE Statement (room)

Set screen update rate

UPDATE: scroll_frames[;normal_frames]

When there is a lot of graphics being updated simultaneously, the
animation(s) may become jerky. This is because all elements can't be
updated within 1/50th of a second - the time gap available before GRAAL
normally refreshes the screen. The problem is most noticeable during

background scrolls, because of the added amount of graphics moving about.

To make the animations smoother, we sometimes need to slow down the updating rate and update the screen perhaps every 2/50ths or 3/50ths, allowing GRAAL more time to do its graphics work. The UPDATE: statement allows you to set specific rates for each room, optimising the performance, and also to provide different values depending on whether the background is currently scrolling or not.

Default values are 6 for background scrolling and 1 for "normal" displays.

1.187 SECTION

SECTION Statement (room)

Defines to which section this room belongs

```
SECTION: section_no|SAME
```

When you enter a new room and the section number is not the same as the one for the previous room, the approprita section file (n.section) is loaded and its contents executed.

Use SECTION: SAME in rooms which are jumped to from different sections, for example load/save rooms or "current score" screens. This makes the room use the previous section number, which prevents messing up the section data...

1.188 3D: statement

3D: Statement (room)

Sets the scaling points when scaling characters

```
3D: max;mid;min;adjspeed
```

3D: enables automatic scaling of controllable characters. max, mid, and min refer to vertical (y) coordinates in the scene area (backdrop picture).

max

below this line, the characters will be shown in their original size. Above it, they are shown at 5/6 of the original size.

mid

Above this line, the characters are shown at 2/3 of the original size.

min

Above this line, the characters are shown at 1/2 of the original size.

adjspeed

This value allows you to adjust the speed factor, so that the smaller
the character becomes, the slower it moves. A value of 0 means no change
to the speed will be made. A value of 0.4 means the speed of the
smallest character size will be half of that in the original character
size. Experiment to find a value giving walking speeds that seem
"natural" in the scene.

By placing some of the lines completely above or below the visible part
of the scene area, you can force GRAAL to use only some of the character
sizes in a particular room.

3D: relies on the data in the
CHAR:
statements to find the images that
must be scaled. This means that even in single-character games, you have
to define a global object for the character, and also create a CHAR:
statement for it

Character scaling affects the following commands:

CPOS
,
CBOB
,
CMOVE
,
MOBJ
,
HANDLE
,
SAY
This means in a "scaled" room, you should not animate characters ↩
using
OMOVE or SHOW. IF two or more scalable characters are in the same room,
and they should interact with each other, you should do this using the

SWITCH
command followed by CMOVE etc. for the respective characters.

1.189 BG_IFF

BACKDROP Statement (room)

Name of background graphics file

```
  BACKDROP: file
```

This statement loads the background graphics for the room. The backdrop should be 120 pixels high, and at least 320 pixels wide.

(Until release 2.2, this statement was called BG_IFF:)

1.190 START_POS

START_POS Statement (room)

Sets possible starting positions

```
  START_POS: startpos_no;image;x;y;L|R|M;floor_no
```

startpos_no

Number of this entrance to the room, should range from 1 and upwards.

image

The image used for the main character when placed in the starting position.

x;y

Position of the main character

L|R|M

Decides whether the Left, Right, or Middle of the backdrop is initially shown.

floor_no

A floor containing the x;y point. If this is not properly set, the main character may do a strange walkabout the first time he is commanded to move somewhere else...

1.191 FLOOR

FLOOR Statement (room)

Defines the "path" where the main character can walk and how they are connected

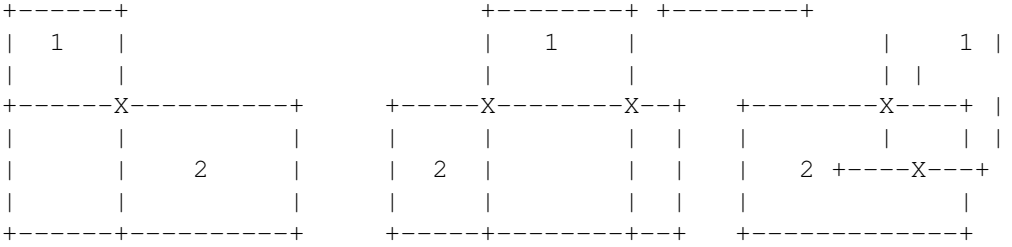
```
  FLOOR: floor_no;x1;y1;x2;y2;floormap{[/floormap]}
```

Each floor defines a rectangular area where the main character can "place its feet". Up to 12 such rectangles can be defined in a room, and

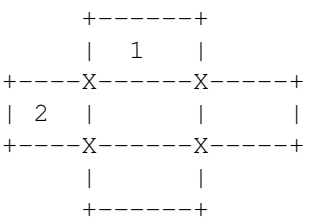
they should be connected in such a way that no floor becomes an "island" without sharing any space with any other floor. In fact, floors shoull overlap as much as possible in order for the character to be able to move about.

OVERLAPPING OF FLOORS

There are some vital rules for floor overlapping. Two floors may have one or two intersection points, but not four. In the following diagrams, X marks the intersection points:



are all OK, but the following is ILLEGAL:



WALKING AROUND

The floormap parameters decide which floors the main character uses to move from one spot to another.

For each floor defined, there must be as many floormaps defined as there are floors in the room. Each floormap has the following format:

  finishfloor-nextfloor

and answers the question: "If my final destination is finishfloor, which floor should I go to from where I currently am?"

Example: For floor 2, there is a floormap

  3-4

This floormap says that to go to floor 3 from floor 2, you should go via floor 4.

Through logic follows that for each floor defined, there is always a floormap for that floor pointing to itself. For example, one of the floormaps for floor one is always

  1-1

because, if you want to get to floor 1, and you already are there, you

should not go anywhere else to get there. That's logic!

If there is more than one floormap, they are separated by a slash (/).

PATHS

From GRAAL 2, there is a way of navigating between floors that do not require the floors to overlap: A path can be defined between them using the
PATH:
statement. Paths can also twist and turn a number of times, which may come in handy in its own right.

To use a path in a floormap, prefix the (path) number with a "P". Click PATH: above for a more complete description.

EXAMPLE

```
FLOOR: 1;10;40;210;60;1-1/2-2/3-2/4-P1
```

This is floor 1, which extends from 10;40 to 210;60. To go to floor 1 (itself), you go on to floor 1 (as discussed above). To go to floor 2, you go directly on to floor 2. To go to floor 3, you go via floor 2. To go to floor 4, you go via path 1, which thus must have one of its end points within floor 1.

1.192 PATH

PATH Statement (room)

defines a path to walk between two floors

```
PATH: path_no;floor1;floor2;px1;py1;px2;py2[{more_points}]
```

floor1

The number of the floor containing the first co-ordinate of the path.

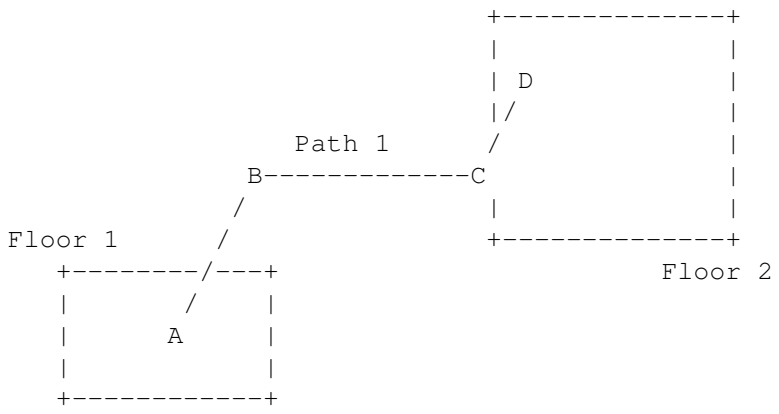
floor2

The number of the floor containing the last co-ordinate of the path.

px1;py1;px2;py2;...

Co-ordinates specifying points along the path.

This is an alternative way to move between two floors. If two floors do not overlap, a path can be defined between them:



As shown in the diagram, a path can alter direction a number of times. This path has a starting point (A), two "knees" (B and C), and an ending point (D). Up to six point may be defined. Two is the natural minimum.

Example:

PATH: 1;2;4;120;40;160;30;230;60

Path 1 goes between floors 2 and 4. The starting point is 120;40 (within floor 2), it turns at 160;30 and ends at 230;60 (within floor 4).

These are the rules:

* A path must start inside one floor and end inside another.

* A when a path should be used, it is specified in a floormap (se the FLOOR: statement) prefixed by a "P". For example, the floormap

  3-P2

  means "if the destination floor is 3, now use path 2"

* Paths cannot be changed by commands, and if the floor settings are changed by the SETFLOOR command, the starting and ending points of all paths must still remain within the original floors. (Just a longwinded way to say "avoid using paths in rooms where you use SETFLOOR a lot...)

* When the player clicks a position on the screen, GRAAL tries to calculate the closest point to which the character can walk. This does NOT include points along the paths, but only the nearest point within a floor. For example, is the player clicks point B in the diagram above, the character will actually only move to the point in floor 1 directly below and closest to it. If the player clicks point C, GRAAL thinks "Aha, you want to go to floor 2, and moves the character to the paths ending point (D), then to the point

```
   within that floor that is closest to point C.
```

1.193 EXIT

```
                EXIT Statement (room)

Defines an exit from the room


  EXIT: exit_no;x1;y1;x2;y2;x;y;description

Up to 12 exits may be defined in each room.

exit_no

is from 1 to 12 and must be unique within the room.

x1;y1;x2;y2

Defines an area which the player can click to exit. x1;y1 = upper left
corner, x2;y2 = lower right corner.

x;y

The co-ordinate the main character will walk to when exiting.

description

Name of exit displayed in the scene area when the cursor moves over it.


When the exit is clicked, GRAAL will execute any ACTION statements that
begin

ACTION: 0;IFOBJ exit_no;...

This should usually be followed by
                MEXIT
                 and
                GOTO
                 commands, if all you
want to do is a straightforward, uncomplicated switch to the next room.
However, you may do anything you like in these ACTION statements that
you can do when taking care of "normal" player input. Just remember, the

                VERB
                 is 0, and
                OBJ1
                 is the exit number.
```

1.194 STATIC

```
                STATIC Statement (room)

Display a static image that is not an object


  STATIC: bob;image;x;y

This statement is particularly useful to insert foreground objects into
a scene, which you never want to manipulate in any way.

As always, make sure the bob number used isn't used for any other image
displayed in this room.

See also:
                ANIM
```

1.195 ANIM

```
                ANIM Statement (room)

Display an animated image that is not an object


  ANIM: bob;image;anim_ch;
                anim
                |ptrn;x;y

This statement is particularly useful for animated foreground images
that you do not wish to manipulate in any way in the game.

See also:
                STATIC
```

1.196 LINE

```
                LINE Statement (section, room)

Define a dialogue alternative


  LINE: dlg;line_no;sentence1;sentence2; |{cond}

Each LINE statement defines a sentence to be shown in the dialogue
control area during a dialogue. It is the
                DSET
                 command that decides
which alternatives actually appears at a certain time.

See the bottom of this text for special notes on use in .section files.

dlg
```

Dialogue number (defined by the DSET statement in graal.main).

line_no

A number from 1-30, must be unique within the dialogue

sentence1

The sentence the main character will "speak" the first time this line is used. If the first character of the sentence is a dollar sign ($), the main character will NOT speak the sentence. This means you can use the dialogues for player multiple choice input, or having dialogues going on when the main character is not shown on screen.

sentence2

The sentence the main character will "speak" if the line has been used before. This enables you to rephrase alternatives in a way that is more natural than having to repeat the first-time line. For example, in the sentence is

  Who are you?

and you would like to repeat that further on, the alternative should be something like

  Who did you say you are?

because repeating the first version would seem rather stupid.

 |{cond}

Special conditions deciding whether the sentence is available to the player or not. Put a blank space if no special conditions exist.

There are two factors deciding whether a line appears in the dialogue control area or not. The first one is that a DSET command must have given it permission to appear. The second one is that all conditions specified here must be fulfilled.

About section dialogues

As of GRAAL 2, dialogues may be common to an entire section of the game.

Needless to say, this takes some care in the planning stage: Just keep in mind that everywhere the dialogue can be invoked, the objects, characters and room flags needed to carry out all possible LACT:s are in place. For example, it is wise to make specific references to room flags (IFRF room,flag=value) rather than test for "current room", because the room may change from call to call.

At any time, the number of active dialogues loaded from the .room and .section scripts must not exceed the number of dialogues set in the

N_DIALOGUES:
statement (default=6). For example, if there are two dialogues in use by the section, no room belonging to that section must load more than N_DIALOGUES-2 dialogues.

1.197 LACT

LACT Statement (section, room)

Contains actions in response to a certain dialogue alternative

```
LACT: dlg;line_no;{cond|comm}
```

Important note: All LACT: statements for a certain dialogue line must be placed immediately below that precise DLG: statement in the script file.

The
LINE:
statement description holds some information about using dialogues in section files.

dlg

This is the number of the dialogue.

line_no

This is the number of the line the player selected.

{cond|comm}

These are any ordinary GRAAL conditions and/or commands, separated by ";" characters.

LACT statements often end with a DSET;EXIT combination to refresh the dialogue status, or an EDLG;EXIT combination to end the dialogue and return to normal input mode.

See also:
LINE
and
DLG
statements, and
DSET
and
EDLG
commands.

1.198 Trouble-shooting

TROUBLE-SHOOTING

An ad-hoc creation like GRAAL is bound to have some niggles, many of which I am probably not aware - because I invented the whole thing to suit my needs and no-one else's. However, there are some common mistakes easily made, which will get you into trouble. Here's how to deal with some of them:

1.199 My command / statement doesn't work

Trouble-shooting:

MY COMMAND / STATEMENT DOESN'T WORK

You have probably got the number of parameters and / or delimiters wrong. In most cases, GRAAL should detect this - in other cases, it doesn't.

Check the syntax with the example files and this reference, and above all, use the parameter editor and syntax checker facilities of the GRAAL editor! Pay particular attention to the delimiters used: Semi-colons for statements and commas for conditions and commands.

Be extra careful with those statements and commands where the last parameter in itself may contain a variable number of other conditions or commands, like the LINE and FLOOR statements. If you want to leave the last parameter of a statement or command blank at the very end of a line, there must be a blank space after the last delimiter. The GRAAL Editor inserts this space automatically, but watch out if you use another editor for some reason.

Also be extra careful when checking the syntax of animation strings!

1.200 My iff pictures look awful / crash the system

Trouble-shooting

MY GRAPHICS CAUSE TROUBLE

A lot of the trouble that may arise is caused by the graphics capabilities of Amos Pro. Remember that only ECS graphics can be used, and not AGA modes.

If you use DPAINT as a paint package, you MUST make sure that stencils and fixed background modes are turned OFF before saving the graphics - otherwise, horrible things will occur when Amos encounter that information which is included in the IFF file.

If you use a paint package with AGA capabilities, also remember that there are more and subtler colour shades available: The colours in the picture, and especially gradients, may look different when loaded into GRAAL. In DPAINT IV (at least), there is a SCALE button in the palette requester that scales all colours to "GRAAL-compatible" shades directly, so you may see what the picture will actually look like.

Keeping the palette colours in order is pretty much up to you - just remember,

* colour 0 is always transparent

* colour 1 should be white

* colour 2 should be black

In the scene area it’s colours 16, 17, and 18 that show up in the mouse pointer, so you may wish to keep these the same in all your backdrop graphics.

1.201 "

Trouble-shooting:

Illegal Function Calls

GRAAL should no longer give the "illegal function" or similar message - instead, you may encounter an "unknown error" message from GRAAL. Here are some possible reasons:

* You tried to
  GOTO
  or
  EXIT
  to a room that has no .room file.

* A
  RESP

command pointed to a character that is
actually not displayed on screen - most likely, you got the dialogue

number wrong in the command.

Alternatively, the
OBJECT
defining the character
hasn't been made visible or was placed in the wrong room in the
OBJECT statement.

* Check that all objects / graphics that you have ordered to be present in the scene have actually been loaded into the BOB image bank first using
CLPART
,
BOBS
,
ROOMBOBS
, etc.

* You tried to use a room flag number higher than 20 or an object flag number higher than 6.

1.202 Mouse cursor does not register visible object

Trouble-shooting

Although an object is visible on screen, the mouse cursor does not pick it up.

This is due to the fact that if more than one object occupies the space where the mouse cursor is, GRAAL only picks up the one that was first added to the list of objects available in the room.

Use the
OBJONTOP
command to remedy the problem if and when it occurs.

1.203 My exits do not appear

Trouble-shooting

My exits do not show up on screen

1. You have not provided an exit name in the EXIT: statement. This means no "Go to..." message is shown in the sentence box, and no name is shown

above the cursor in the scene area. However, this can also be put to good use, because it allows you to use exits for defining any kind of clickable area other than true exits and objects, for use as menu selections and what have you.

2. An exit is not detected by GRAAL if it is covered by an object. The remedy is often to make the graphics that are to be shown when the exit is available the backdrop picture. A common example would be a door opening. The only time the exit is available is when the door is open. Therefore, the open doorway should be the backdrop picture, the closed door should be an object or a BOB put over the doorway when the door is closed and the exit is hidden.

(Completely hiding an exit with an object this way actually means you do not have do issue the HIDE EXIT command. However, I think it is good coding practice to include it just the same. The code makes a lot more sense that way, showing clearly what is going on in the room.)

1.204 GRAAL

Trouble-shooting

GRAAL looses the exit number

Before you are finished going through the commands for an EXIT click (ACTION: 0;IFOBJ....), the exit number seems to have changed or disappeared.

Remember that the exit number is actually a special use of OBJ1, so any commands put in these ACTION: statements that alter
OBJ1
 must put the
original back before it is time to check the exit number next time!

1.205 Index

Index of database 002e4fd0-0

Documents

" link
" link

~CUTSCENE_LAYOUT~

.room files

.section files

A Very Short Introduction

DOAFTER

DSET

EDLG

EXEC

EXIT

FADE

FINAL

FLOOR

GET

GOTO

graal.main file

HANDLE

HIDE

HIDEEXIT

HOTSP

ICON

IFCARR / IFNOTCARR

IFCBOB

IFDATE

IFEXISTS

IFFLOOR

IFNOTSAVEDISK

IFOBJ / IFOBJ2

IFOF

IFPICK

IFRF

IFROOM

IFSPOS

IFTIME

PICK

PREP

QUIT

REDO

REMOVE

RESP

RESTORETIME

RESUME

ROOM

SAMLOAD

SAMPLAY

SAVE

SAVETIME

SAY

SENTENCE_LAYOUT

SETDATE

SETFLOOR

SETOF

SETRF

SETTIME

SHOW

SHOWEXIT

SOUND

SPLIT_LINE

Syntax Conventions

SYSTEM_TEXT

TEXT

The GRAAL player interface

The Structure of a GRAAL Game

NAME

NTSC_TIMING: Statement

OBJECT

OBJ_COL

PAUSE_RIGHT

PROMPT Command

RESOURCE

SECTION

SELECT_CHAR: statement

SETVAR Command

START_POS

START_ROOM

STATIC

STILL_RIGHT

SWITCH

TALK_MAP

TCURS command

TRACE Command

UPDATE

VERSION

WALK_RIGHT

WALK_SPEED

WALK_SPEED command
Buttons

~A~Very~Short~Introduction~~~~~~~~~~~~~

~About~this~Reference~~~~~~~~~~~~~~~~~~

~ACTION~

~ADDOF~~~~~~~~~~

~ADDRF~~~~~~~~~~

~ADDTIME~~~~~~~~

~ANIM~

~BACKDROP~

~BOBOFF~~~~~~~~~

~BOBON~~~~~~~~~~

~BOBS~~~~~~~~~~~

~CAMERA~~~~~~~~~

~CANCEL~~~~~~~~~

~CBOB~~~~~~~~~~~

~CHAR~~~~~~~~~~~

~CLPART~

~CLPART~

~CLPART~~~~~~~~~

~CMOVE~~~~~~~~~~

~COLOUR~~~~~~~~~

~COMAREA~~~~~~~~

~COMGR~~~~~~~~~~

~Commands~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

~Conditions~~~~~~~~~~~~~~~~~~~~~~~~~~~~

~CPOS~~~~~~~~~~~

~CUTSCENE~~~~~~~

~DACT~

~DEBUG~

~DECOF~~~~~~~~~~

~DECRF~~~~~~~~~~

~DOAFTER~~~~~~~~

~DSET~~~~~~~~~~~

~EDLG~~~~~~~~~~~

~EFFECT~~~~~~~~~~

~EXEC~~~~~~~~~~~~

~EXIT~

~EXIT~~~~~~~~~~~~

~FADE~~~~~~~~~~~~

~FINAL~~~~~~~~~~~

~FLOOR~

~FLOOR~~~~~~~~~~

~FOLLOW~~~~~~~~~

~GET~~~~~~~~~~~~

~GLOBALOBJS~

~GOTO~~~~~~~~~~~

~HANDLE~~~~~~~~~

~HIDE~~~~~~~~~~~

~HIDEEXIT~~~~~~~

~HOTSP~~~~~~~~~~

~ICON~~~~~~~~~~~

~IFCARR~/~IFNOTCARR~

~IFCBOB~~~~~~~~~~~~~

~IFCHAR~~~~~~~~~~~~~

~IFDATE~~~~~~~~~~~~~

~IFEXISTS~~~~~~~~~~~

~IFFLOOR~~~~~~~~~~~~

~IFHERE~~~~~~~~~~~~~

~IFNOTSAVEDISK~~~~~~

~IFOBJ~/~IFOBJ2~~~~~

~IFOF~/~IFOF2~~~~~~~

~IFPICK~~~~~~~~~~~~

~IFRF~~~~~~~~~~~~~~~

~IFROOM~~~~~~~~~~~~~

~IFSPOS~~~~~~~~~~~~~

~IFTIME~~~~~~~~~~~~~

~IFTYPE~~~~~~~~~~~~~

~IFVAR~~~~~~~~~~~~~~

~IFWEEKDAY~~~~~~~~~~

~INVENTORY~~~~~~

~LACT~

~LIGHTS~~~~~~~~~

~Limitations,~Ranges,~Reserved~Numbers~

~LINE~

~LINE~~~~~~~~~~~

~LOAD~~~~~~~~~~~

~Machine~Requirements~~~~~~~~~~~~~~~~~~

~MARK~~~~~~~~~~~

~MAX_CACHE~

~MAX_DACT~

~MAX_DLG~

~MAX_ROOM~

~MAX_SECTION~

~MEXIT~~~~~~~~~~

~MOBJ~~~~~~~~~~~

~NAME~

~NAME~~~~~~~~~~~

~NEWOBJ~~~~~~~~~

~NFLOOR~~~~~~~~~

~NOBREAK~~~~~~~~

~N_DIALOGUES~

~N_GLOBALBOBS~

~N_ROOMBOBS~

~N_SECTIONBOBS~

~OBJ1~/~OBJ2~~~~

~OBJONTOP~~~~~~~

~OMOVE~~~~~~~~~~

~PATH~

~PBOB~~~~~~~~~~

~PICK~~~~~~~~~~

~Player~interface~and~shortcut~keys~~~~

~PREP~~~~~~~~~~

~PROMPT~~~~~~~~

~PUT~~~~~~~~~~

~QUIT~~~~~~~~~~

~REDO~~~~~~~~~~

~RESP~~~~~~~~~~

~RESTORETIME~~~~

~RESUME~~~~~~~~

~ROOMBOBS~

~ROOMOBJ~

~ROOMOBJS~

~SAM~~~~~~~~~~

~SAMLOAD~~~~~~~~

~SAVE~~~~~~~~~~

~SAVETIME~~~~~~~

~SAY~~~~~~~~~~

~SECTION~

~SECTIONBOBS~

```
                ~VERSION~

                ~W(ait)~~~~~~~~~

                ~WALK_SPEED~~~~~
                .ptrn
.scene

                3D:

                3D:~statement
                =DEMO=>
=DEMO=>
=DEMO=>

                >>Unknown~error<<~messages~~~~~~~~~~~~~~~~~~~

                ACTION

                ACTION:

                Additions~to~the~cx~parameter~of~OBJECT~definitions

                ADDOF

                ADDRF

                ADDTIME

                ANIM

                anim

                ANIM:

                animation~sequence

                animation~sequences

                AREA_SIZES:

                BACKDROP:~statement~replacing~BG_IFF:

                BOBOFF

                BOBON

                BOBS

                BOBS

                BOBS:

                CANCEL

                CBOB
```

```
CHAR

CHAR~OFF

CHAR:

CHAR:~statement

CHARACTER_WIDTH:

CHAR_HEIGHT:

CHAR_WIDTH:

CLPART

CLPART

CMOVE

COLOUR

COMAREA~command

COMGR~command

COMMAND_AREA:~Statement

CPOS

CURSOR_PALETTE

CUTSCENE

DACT

DATE_FORMAT

DATE_LAYOUT

DAY_TEXT:

DECOF

DECRF

DISABLE_SAVE:

DLG

DLG:

DLG_LAYOUT:~statement

DOAFTER
```

```
DSET

EDLG

EFFECT

EXIT

EXIT

EXIT:

FADE

FLOOR

FLOOR:

FOLLOW

GET

GOTO

GRAAL~'looses'~the~exit~number~~~~~~~~~~~~~~~~

graal.main

HANDLE

HIDE

HIDEEXIT

IFEXISTS

IFNOTSAVEDISK

IFOF

IFRF

IFTYPE

IFVAR

INVENTORY

LACT

LACT

LIGHTS~ON

LINE

LINE:
```

LOAD

MAKE3D~command

MARK

MAX_ACTION: statement

MAX_DLG:

MEXIT

MOBJ

MONTH_TEXT:

Mouse~cursor~does~not~register~visible~object~

My~command~/~statement~doesn't~work~~~~~~~~~~

My~exits~do~not~show~up~on~screen~~~~~~~~~~~~~

My~iff~pictures~look~awful~/~crash~the~system~

n.room

n.section

NAME

NFLOOR

NOBREAK

N_DIALOGUES:

N_VERBS

OBJ1

OBJ1

OBJ1/OBJ2

OBJECT

OBJECT

OBJONTOP

OMOVE

PATH:

PAUSE_...

```
PBOB

PICK

PROMPT

PUT~command

QUIT

Read~the~News!!!

REDO

REMOVE

RESP

RESTORETIME

RESUME

ROOMBOBS

SAM

SAMLOAD

SAMPLAY

SAVE

SAVE

SAVETIME

SAY

SELECT_CHAR:~statement

sentences

SETDATE

SETFLOOR

SETOF

SETRF

SETTIME

SETVAR

SHOW

SHOWEXIT
```

SOUND

special~characters

START_POS

START_POS:

STATIC

STATIC:

STILL_...

SWITCH

syntax

TCURS~command

THINK

TIME_FORMAT

TIME_LAYOUT

TITLE

TITLEFONT:

TRACK

TRACK~CHANNELS=

TYPE

variable

VERB

VERB
VERB_TEXT:

VERB_TEXT:

VERB_ZONE:

VERB_ZONE:~Statement

WALK_...

WALK_SPEED

WALK_SPEED~command

```
                WALK_SPEED:
```