

3D GameStudio



World Definition Language

Reference manual

ACKNEX version 3.9 - Mai. 31, 1998



CONITEC DATENSYSTEME

64807 Dieburg • Dieselstr. 11c • fon 06071-92520 • fax 06071-925233

Copyright © Conitec GmbH 1996, 1997, 1998

WDL interpreter	Volker Kleipa
ACKNEX engine	Johann Christian Lotter
Sound/music	Wladimir Stolipin
World editor	Wolfgang Schulz / Wladimir Stolipin (<i>original code: B. Wyber & R. Quinet</i>)
World compiler	Christoph Kirst

Latest news, demos, and tools are available from our site
<http://www.conitec.com>

As a commercial or professional version owner you are entitled to **technical support** via email. Please mail your question to support@conitec.com, and don't forget to give your version, purchase date and serial resp. customer number (on disk one resp. on your invoice).

Even if you own only a lite version, you're not on your own: You'll find hints, a FAQ section, and the international ACKNEX forum in the **ACKNEX Users Magazine** on **<http://www.conitec.com/aum>**.

Manual and Software are protected under the copyright laws of Germany. Any reproduction of the material and artwork printed herein without the written permission of conitec is prohibited. We undertake no guarantee for the accuracy of the informations in this manual. conitec reserves the right to make alterations or updates without further announcement.

Contents

Introduction	1
World Editor	4
Vertex mode	12
Wall mode	14
Region mode	14
Thing and actor mode	15
Way mode	16
WMP file format	17
The ACKNEX 3D engine	19
World Definition Language	22
Introduction	22
Keywords	23
Files	27
Predefined Keywords	29
Palettes	33
Textures	36
Walls	50
Regions	56
Things, Actors, Ways	63
Actions	74
Synonyms	98
Skills & role playing games	100
Changing levels, saving the game, and multiplayer mode	122
User interface: Views, Panels, Texts and Overlays	124
Publishing your game	133
Problems & Hints	137
Index	140

Introduction

Congratulations! You have purchased the conitec 3D GameStudio. This toolkit now allows you to create 3-D demos, role playing, action, adventure or racing games *without programming knowledge* and to publish them subsequently without having to pay royalties.

In order to construct a game you have to write a script file (**WDL**) which will contain the 'source-code' for the game world with specifications concerning textures, regions, things, and actors. The level topography is contained in a second file (**WMP**) with the coordinates (vertices) of all objects. All files contain plain ASCII text, so you may edit them with any ASCII editor like WORDPAD or EDIT. The WMP file, however, is normally created with the topography editor **WED**.

WED contains the **ACKNEX** 3-D engine for running and testing the game. When your game is finished, WED can create a **runtime module** for it, a separate .EXE file, which you can then freely distribute or publish. There will be two different runtime modules created for each game, one for DOS, the other for Windows 95 / DirectX.

There are currently three versions of ACKNEX, the lite, commercial and professional version. They differ in the features supported, and , of course, in price. In the last chapter of this manual you'll find the features and royalty conditions listed.

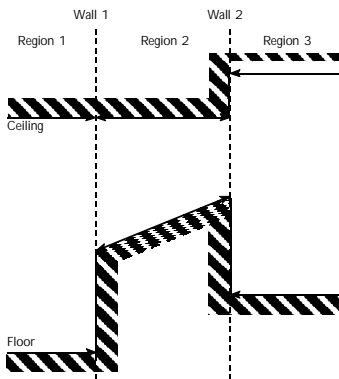
This handbook is supposed to serve as a reference. In order to familiarize yourself with the many features ACKNEX offers we would suggest to thoroughly use the tutorial also included.

The virtual world which you can create with 3D GameStudio is composed of the following elements:

A **level** is a part of a game, and can be compared to a two-dimensional 'map'. The graphics and other components of a level will be in the RAM or the virtual memory in their entirety. After loading a level the world will be precalculated and pre-rendered, which may take several seconds.

The level is split into **regions**. A region is defined by the walls surrounding it, and by its floor and ceiling. You can consider it as a solid column of infinite height, with a 'gap' between the floor and the ceiling. It may have the shape of a complex polygon. Each region may contain any number of further regions. The heights and slopes of floor and ceiling of a given region are freely definable; this way you can build staircases, gorges etc. (one step of a staircase equalling one region!). One source of light may be defined per region. You can stack regions vertically in order to insert more 'gaps' into the column, to construct bridges, suspended objects, vehicles or multi-floor buildings. Floors and ceilings have textures and ambient light values, or may be given a backdrop (sky) texture. The same region may appear several times within a level.

The regions are bordered by vertical **walls** of any height, length and angle. Thus on each side of a wall there has to be a region. Walls are represented by lines connecting two **vertices** on the two-dimensional map. They can be at any angle to each other. Their visible heights - the parts of a wall you see - are determined by the *difference* of floor and ceiling heights of the regions of both sides. If both regions have the same height of floor and ceiling, the wall is hidden within the floor and ceiling, and thus invisible.



Side view of three neighboring regions

The figure shows you a side view like a vertical cut through three adjacent regions of different heights. The middle region represents a slant 'wall'. Wall textures are only visible along the continuous lines. Steps, windows or portals result from different heights of floors or ceilings.

Floors and ceilings may be inclined by the positions of the Z-coordinates of their vertices (represented by wall 2 in above figure).

If the ceiling of a region is given a sky texture, the open sky will be seen over this region.

In the level you can place objects (**things**) or living beings (**actors**) at any position whatsoever. The texture of such an object may change with the perspective, thus the object - although only two-dimensional - may appear to be spatial. Actors are independently moving, 'intelligent' objects. The behaviour and properties of an actor may be defined by appropriate use of the WDL language. With the WED you can give actors a "**way**" which they will follow in the level. Each actor is a state machine with determined **actions**, triggered by **events**.

Textures are bitmaps of any desired size, which can cover walls, floors, ceilings or objects. Textures may be animated. They can have a sound and they can change during gameplay, e.g. by being hit. Through the 'attach' feature you can cover a wall with as many 'layers' of textures as you desire, eg in order to display inscriptions or bullet holes. Textures are shaded in real time, depending upon the ambient light, their own reflectivity and their distance to the player. 256 colors are available, including a transparent color for holes or windows.

In the following chapters the commands and functions of the editor and the World Definition Language WDL are described as a reference listing. If you want to start at once with the developing a game level, first read the **tutorial** at the end of this book. There you will find a step by step explanation of how to build a minimal level.

If you haven't done it yet, now it's a good time to insert the Disk labeled #1, open it and double click onto the setup icon.

World Editor

The world editor WED.EXE (WEDS.EXE in the lite, WEDC.EXE in the commercial version) is the integrated environment to create and edit the world topography. WED scans the WDL and - if any exists - the WMP file; it can generate a runtime module and (with the professional version) compile all your game files together into a single compressed resource file.

WED is a DOS application, but runs under Windows 95 as well. Start the world editor with the following DOS command:

WED [*WED options*] *name*[.*WDL*] [*name.WMP*] [*ACKNEX options*]

If you omit the WMP name, WED loads the WMP file which is declared in the WDL script. Should this file not exist, it will be created.

You can give the following WED command line options (the ACKNEX options are discussed in the next chapter):

- S** SVGA resolution 800x600
- I** SVGA resolution 1024x768

- VESA** Use VESA BIOS for the graphic display (slower!). Start with this option if your video card has a nonsupported chipset and the SVGA display seems distorted (e.g. Matrox Mystique).

- VGA** Start with this option if you don't have a SVGA card. Floor and wall textures, however, cannot be displayed in this mode.

- RUN** Run the game directly without switching to WED. Can also be used to run a compiled game resource (.WRS) file.

Batch options (professional version only):

WED -C *name[.WDL] [name[.WDL]...]*

Compiles all level files *name.WDL* given to the resource *name.WRS*.

WED -CT *name[.WDL] [name[.WDL]...]*

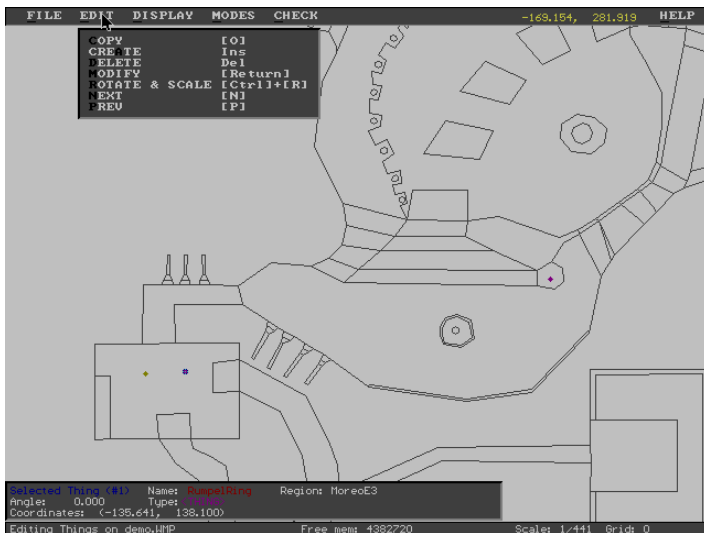
Like **-C**; but all files are compiled, even such ones which are skipped in the WDL file by **IFDEF** or **IFDEF** (see below). This way several versions of a level can be put together into the same resource file.

WED -X *name[.WDL] directory*

Copies only those files you need to run *name.WDL* into the given *directory*. This is useful to separate your necessary game files from unneeded ones.

WED -XT *name[.WDL] directory*

Like **-X**; but all files are copied, even such ones which are skipped in the WDL file by **IFDEF** or **IFDEF**.



After start-up WED shows the WMP file as 'map' representing walls as white lines, things as green crosses, actors as red crosses, giving a blue cross for the

starting point of the player. The scale is adjusted to allow the complete level to fit on the screen.

You can work with WED like a with CAD program: Objects are *selected* with a touch of the mouse. In order to *mark* it, click with the left mouse button. The info panel on the lower screen gives you information about the current mode and the parameters of the object selected or marked most recently.

WED menu functions:

FILE	[Alt-F]	File submenu
OPEN WMP	[F3], [Ctrl-O]	Loads a new WMP file with definitions based on the actual WDL file. If no such file exists, it will be created.
SAVE WMP	[F2], [Ctrl-S]	Saves the current WMP file with backup (.BAK) file creation.
SAVE AS	Shift-[F2]	Saves the current WMP file under a new name.
PUBLISH	[Ctrl-P]	Generates the DOS and the Windows 95/98 freely distributable runtime module (WRUN.EXE and WWRUN.EXE) for the current level. Use the first level of the game to create the runtime modules. If you are going to distribute your game royalty free, please consider the restrictions for royalty free publication (see last chapter). WWRUN.EXE needs the definition files WWRUN.MDF, WWRUN.WDF in its directory in order to run.

COMPILE & PUBLISH	[F9]	Like PUBLISH, but also invokes the world compiler to create a world resource file (.WRS) out of all files belonging to the level (professional version only). Also displays the 'Magic Key' number you'll need if you have to pay royalties for the publication. If you want all files to be included into the resource, even such ones which are skipped in the WDL file by IFDEF or IFNDEF , keep the All IFDEFs switch in the dialogue enabled. The runtime modules for a resource file are named VRUN.EXE (DOS) and WVRUN.EXE (Windows 95/98).
EXIT	[Alt-F4], [Ctrl-Q], [F10]	Exits WED.
MODE	[Alt-M]	Mode submenu
VERTICES	[V]	Activates vertex mode.
WALLS	[W]	Activates wall mode.
REGIONS	[R]	Activates region mode.
THINGS	[T]	Activates thing/actor mode.
WAYS	[Y]	Activates way mode.

WALK- THROUGH	[Shift-^] [Shift-~]	Starts the game, or switches between game and editor while the game is running. This allows you to test your topography. If the player position is changed within WED, the new position will instantaneously be taken over into the game. Changes of walls, things, and actors will also simultaneously appear in the game.
------------------	-------------------------	---

EDIT	[Alt-E]	Edit submenu
-------------	----------------	---------------------

UNDO	[BkSp], [Ctrl-Z]	Cancels successively the last operations.
UNDO ALL	[Ctrl-U]	Cancels all changes after the last saving.
COPY	[Ctrl-V]	Creates a copy of the selected object slightly displaced to the lower right.
CREATE	[Ins], Left Click	Creates a new object, dependent on the actual mode.
DELETE	[Del]	Deletes the selected object.
MODIFY	[Return], Right Click	Changes the attributes (name etc.) of the selected object or object(s) marked.
IMPORT	[Ctrl-I]	Adds vertices and objects from an external .WMP file to the map. The WMP file may be selected from the current directory using a file scroll box. All names of the new objects must be defined in the actual .WDL file!

EXPORT	[Ctrl-E]	Saves all marked objects including all belonging vertices to a new .WMP file in the actual directory.
ROTATE & SCALE	[Ctrl-T]	Rotates and scales the selected or marked object(s) by a given angle and percentage value.
NEXT	[Tab]	Selects next object.
PREV	[Shift-Tab]	Selects previous object.
WALL	[Alt-W]	Wall submenu (wall mode only)
SPLIT	[S]	Splits the selected wall and inserts a new vertex in the middle.
JOIN	[J]	Connects several adjacent walls marked and erases the joint vertices. The properties of the first of the old walls are assigned to the new wall.
FLIP	[F]	Alters the orientation of the selected or marked walls by exchanging the start and end vertex as well as the left and right region. It mirrors the wall texture and is necessary for functions which treating the left and right sides of a given marked wall differently.
RESET	[Z]	Resets the horizontal and vertical alignments (X and Y offset) of all marked walls to zero.

ALIGN HOR	[A]	Adjusts the textures of the right sides of several connected walls horizontally so that their edges are seamless. This is done by changing the X offsets. You must select a connected chain of walls. If necessary, change the orientation of single walls with FLIP. Starting with the first wall of the chain, all X offsets of the adjacent walls are adjusted so as to start any texture at the end of the preceding texture.
ALIGN CEILING	[C]	Adjusts the textures of several marked walls vertically to the ceiling by changing the Y offsets. After this operation the upper borders of the textures match exactly with the ceiling of the right hand region of the first marked wall.
ALIGN FLOOR	[L]	Adjusts the textures of one or more several marked walls vertically to the floor by changing the Y offsets. After this operation the upper borders of the textures lie exactly at floor level of the right hand region of the first marked wall.
LENGTH	[Ctrl-L]	Assigns a defined length to a selected wall by moving the second vertex of this wall. When calling this function only one wall can be selected. A numeric window allows you to define the length of the wall in steps.
RECTANGLE	[Ctrl-R]	Creates a quadrilateral of four walls at cursor position. A numeric window allows you to define the desired length in steps.

POLYGON	[Ctrl-Y]	Creates an n-sided polygon of n walls at cursor position. You can specify the number and length of the sides in a numeric window.
---------	----------	---

DISPLAY	[Alt-D]	Display submenu
----------------	----------------	------------------------

GRID	[G]	Switches the blue grid on and off. If the grid is enabled you can place vertices, things and actors only at the grid points.
------	-----	--

GRID+	[+]	Doubles the scale of the grid.
-------	-----	--------------------------------

GRID-	[-]	Halves the scale of the grid.
-------	-----	-------------------------------

ZOOM+	[Num*]	Increases the scale of the map (also with keys [0]..[9]).
-------	--------	---

ZOOM-	[Num/]	Reduces the scale of map.
-------	--------	---------------------------

SCROLL	[Scroll Lock]	Switches the autoscroll feature on or off. In autoscroll off mode the visible section of the map can only be moved with the cursor keys.
--------	---------------	--

CHECK	[Alt-C]	Check submenu
--------------	----------------	----------------------

KEYWORDS	[Ctrl-K]	Checks the actual WMP names against the definitions in the WDL file.
----------	----------	--

STATISTICS	[Ctrl-H]	Shows the number of objects of the current level.
------------	----------	---

SCAN REGIONS	[O]	Checks all regions and switches to region mode. Necessary, if new regions were created e.g. by splitting a region with a wall.
--------------	-----	--

CHECK WALLS	[Ctrl-W]	Checks the consistency of the assignment of regions to walls. All suspicious Walls that are wrongly assigned are automatically marked.
DUPLICATES	[Ctrl-D]	Marks all regions, walls, things, actors, or regions which have the same name as the selected object.
HELP	[Alt-H]	Help submenu
ABOUT		Shows the version number.

If the mouse pointer is positioned above an object, it will automatically be selected with a yellow frame. Clicking the left mousebutton marks the selected object with a green frame. You can draw a frame to mark several objects, or click on them with pressed **[Shift]** key. You can drag and drop the selected or marked object(s) with pressed left button.

In all modes you can zoom the section of the map containing the mouse pointer by the keys [1]..[9] in fixed steps. With the **[TAB]** key you jump the selection mark from one object to the next.

1. Vertex mode

You can select this mode with the **[V]** key or by menu. You may place, move or erase vertices and connect them with lines which then will become walls. As a result of restricted mathematical accuracy the distance between a vertex and an independent wall must be at least 0.25 steps. If the respective wall should also be visible from farther off, the distance must increased correspondingly. When tilting regions by use of the vertex Z coordinates be careful not to 'bend' floor and ceiling areas, and to hold the angle of inclination below around 75°. Tilted areas will not be shown very precisely, therefore diffuse textures are more desirable in this case.

In vertex mode the following mouse and keyboard functions are active:

Left click on background places a vertex. The vertex is shown as green cross.

Left click on vertex marks the Vertex. If **[Shift]** or **[Ctrl]** is pressed simultaneously with the click, all previously marked vertices remain marked - otherwise they will be unmarked. A marked vertex is identified by a green frame.

Left pull on background creates a frame for marking several vertices.

Left pull on vertex marks and shifts the vertex. If **[Shift]** or **[Ctrl]** is pressed, all other marked Vertices are moved; otherwise they will be unmarked.

Right click on vertex allows editing of the Z coordinate (height) of a vertex, in order to create tilted regions.

[Del] erases all marked vertices.

[Ins] connects all marked vertices with wall lines in the sequence they were marked in and switches into the wall mode. If there are already defined regions to the left or right of the the new walls, they will automatically be assigned to the walls.

[Shift]-[Ins] connects all marked vertices - if there are more than two - with wall lines in the sequence they were marked in, forming a closed polygon of wall lines. as above.

2. Wall mode

This mode is switched on using the **[W]** key or by menu. It allows you to place, erase, shift, rotate or scale groups of lines. Every wall **MUST** have been given a name and a left and right region. The right side of each wall is marked with a small perpendicular 'nose'. Regions must be enclosed by walls on all sides. Regions touching without being separated by walls will cause visible errors in the picture!

Left click on wall marks the wall. If you press **[Shift]** or **[Ctrl]** simultaneously, all marked walls stay marked; otherwise they will be unmarked. A green frame appears around the marked wall.

Left pull on background creates a frame for marking walls.

Left pull on wall marks and shifts the wall together with its vertices. Simultaneously pressing the **[Shift]** or **[Ctrl]** shifts all additionally marked walls; otherwise they are unmarked.

Right click on wall causes a scrollbox to appear. It allows you to assign this wall a wall name from the WDL file.

[Del] erases all marked walls including their vertices.

3. Region mode

In this mode you can assign your previously defined regions to areas surrounded by walls, and alter the floor or ceiling heights. If you have created new regions by placing walls, switch into region mode with **[O] (SCAN REGIONS)**, otherwise **[R]** is sufficient.

Left click on region marks the region. When simultaneously pressing either **[Shift]** or **[Ctrl]** all regions marked afterwards will remain marked; else they will be unmarked. A green frame appears around the marked regions.

Left pull on background creates a frame for marking several regions.

Left pull on region marks and shifts the region together with its walls and vertices. When simultaneously pressing either **[Shift]** or **[Ctrl]** all regions marked afterwards will also be shifted; else they will be unmarked.

Right click on region causes a scrollbox to appear. It allows you to assign the region (or all marked regions) a new region-keyword from the WDL file. You can either change the region heights or using the default heights from the region definition.

[Del] erases all marked regions including their walls and vertices.

4. Thing and actor mode

Here you can place Things and Actors, assign starting angles and mark positions - for example the starting position of the player.

Left click on background creates an object (Thing or Actor). The object appears as a circle with a green cross.

Left click on object marks the object. When simultaneously pressing either **[Shift]** or **[Ctrl]** all objects marked afterwards will remain marked; else they will be unmarked. Around the marked object a green frame will appear.

Left pull on background creates a frame for marking several objects.

Left pull on object marks and shifts the object. When simultaneously pressing either **[Shift]** or **[Ctrl]** all objects marked afterwards will remain marked; else they will be unmarked.

Right click on object causes a scrollbox to appear. It allows you to assign the object marked the player starting position or a thing/actor keyword.

[Ins] creates an object (same as left click).

[Del] erases all marked objects.

5. Way mode

This mode is activated with the **[Y]**-key or through the menu. You may define a closed path of any number of waypoints for an actor. The path is displayed on screen as a light-blue dotted line connecting the waypoints. The following mouse and keyboard functions are active in this mode:

Left click on background creates a new waypoint at the cursor position in the actual way. This waypoint appears as blue cross and will automatically be inserted between the two closest waypoints connected with each other or be connected to the last marked waypoint.

Left click on waypoint marks the waypoint. When simultaneously pressing either **[Shift]** or **[Ctrl]** all waypoints marked afterwards will remain marked; else they will be unmarked. A blue frame will appear around the waypoint marked.

Left pull on background creates a frame for marking several waypoints.

Left pull on waypoint marks and shifts a waypoint. When simultaneously pressing either **[Shift]** or **[Ctrl]** all waypoints marked afterwards will remain marked; else they will be unmarked.

Right click on Waypoint assigns a way name. A pop-up list of all defined way keywords appears, from which you can choose one.

[Ins] creates a new way and sets the first waypoint at the actual cursor position.

[Del] erases all marked waypoints from the way. The remaining waypoints will stay connected.

6. WMP file format

The WMP file generated from WED contains plain ASCII text which can be edited with any text editor (but we suggest better not to try it!). It includes a list of all coordinates and objects in declared sequence in the following format:

VERTEX *x y z*;

Wall vertex definition. *x*, *y* and *z* are fixed point numbers and correspond to the X and Y coordinate and the height of the Vertex in steps.

REGION *name f c*;

Region definition; *name* is the keyword for the region from the WDL file, *f* is the floor and *c* the ceiling height.

WALL *name v1 v2 rr rl sx sy*;

Wall position. *name* is the keyword for the assigned wall from the WDL file. *v1* and *v2* are the consecutive numbers of the vertices of both corner points. *rr* and *rl* are the consecutive numbers of the left and right regions (relative to the view from *v1* to *v2*). *sx* and *sy* are the horizontal and vertical texture offsets in pixels, which may be changed by the ALIGN function.

THING *name x y a r*;

Thing position. *name* is the keyword from the WDL thing definition, *x* and *y* the coordinates of the thing, *a* the angle of the thing in units from 0...360 and *r* the region the thing inhabits.

ACTOR *name x y a r*;

Actor position. *name* is the keyword from the WDL actor definition, *x* and *y* the coordinates of the actor's starting position in steps, *a* the angle of the actor in units from 0...360 and *r* the region the actor inhabits.

PLAYER_START *x y a r*;

Starting position, angle and region of the player.

WAY *name* *x1 y1* *x2 y2*... *xn yn*;

Way definition. *name* is the assigned keyword of the way, followed by the pairs of coordinates for the waypoints.

The ACKNEX 3D engine

This is the core of the software - the 3D rendering engine which runs the game. It is integrated in WED, as well as in the runtime modules for DOS or for Windows 95. You can start it by running WED with option **-RUN** or by selecting **WALKTHROUGH** or pressing **[F11]** from the WED window.

Apart from the file name the following WED command line options are available:

- WMP *name*** Uses the topography from the file *name.WMP*, even if a different file was defined with MAPFILE.
- Dir *name*** Employs the directory defined for the game score, even if a different directory was defined in the WDL.
- D *name*** Defines the token *name* for evaluation in the WDL file by **IFDEF** (see below).
- NODE *n*** Number of the network node (0...1) with multi-player games (commercial or professional version only)
- COM *n*** Port number of the serial link (1...4) used for communication with multi-player games. The serial interface may not be used by the mouse and must have a free interrupt available. COM1 and COM3 are assigned to INT4, COM2 and COM4 are assigned to INT3.
- IPX** Use a local area network (LAN) with IPX protocol, instead of a serial link, for multi-player communication (professional windows version only).
- OS** Run without sound.
- OM** Run without music.
- OCD** Run without activation of audio CDs.
- OT** Test run without sound, music and timer.
- RO** Use - instead of the FM sound - the MPU401, if available, for wavetable music.

- NJ Disable the joystick; useful for avoiding joystick port problems with the DOS engine under a multitasking operating system like Win95.
- NM Disable the mouse.

- C Syntax check mode; checks only the syntax of the WDL and WMP files.
- E Numer of error messages. Usually up to 6 error messages will be returned after which the pressing of any button is required.
- W1 Extended syntax check with warnings for potential errors in the WDL file.
- W2 Like **-W1**, but also shows unused objects and superfluous definitions.

- SST Single step modus; you have to press the **[S]**-Key after each frame.
- GOD God mode: The player can now pass massive walls and objects within the level (be carefully - may crash!).
- NC No **CLIP_DIST** check (see below).

- WND Will start in window mode instead of fullscreen mode.
- 3D_SIMEYE Will start in 3D stereoscopic mode with SimulEyes® shutter glasses support (professional windows version only; won't be supported by all video cards / directx drivers).

With the game running you can toggle special debugging modes by pressing certain key combinations. These modes are disabled within the runtime modules.

- [Ctrl]-[Alt]-[G]** Toggles God mode.
- [Ctrl]-[Alt]-[C]** Toggles **CLIP_DIST** check.
- [Ctrl]-[Alt]-[S]** Toggles single step mode. The game 'freezes' and can be continued frame by frame by pressing the **[S]** key.
- [Ctrl]-[Alt]-[End]** Quits the game.
- [Alt]-[Return]** Toggles between fullscreen and window mode.
- [F11]** Switches back to WED.

If not redefined by WDL, the following keys are active within the game:

- [F2]** Saves the game into the file "TEST_0.SAV".
- [F3]** Loads the last saved game.
- [F5]** Toggles motion blur.
- [F6]** Takes screenshots into the file "SHOT_n.PCX".
- [F10]** Quits the game.
- [F12]** Toggles music and sound.
- [TAB]** Toggles the automap.
- [0]** Activates a default player movement which allows the player to fly and move vertically (disabled within runtime module).

World Definition Language

1. Introduction

The elements and objects of your virtual world - walls, regions, actors and so on - are described in a WDL script (WDL = World Definition Language). If the game consists of several levels, a separate WDL script can be written for each level.

There are different types of definitions, e.g. for regions, walls, actors, things, textures, bitmaps and sounds. Objects of a higher order like walls may contain lower order elements like textures which themselves may contain bitmaps or sounds. You can describe the **properties** of an object by predefined parameters. You may assign **actions** to objects, triggered through **events** that act upon the object. These actions themselves may change the properties of a given object.

Space and time units of the virtual world are the *step* and the *tick*. One step is equivalent to a screen diameter (around 40 cm). One tick is equivalent to the time between two frame cycles on a 486 PC - this means around 1/16 second. Angles are stated in radians (0 to 6.28) and counted counter-clockwise, while 0 is equivalent to the positive direction of the X axis coordinate.

The following special characters are valid within WDL:

...;	Semicolon terminates assignment
...,...	Commata separate parameters
{...}	Parameter or instruction lists are between winged brackets
"..."	Text is between quotation marks
< ...>	File names (without path) are between pointed brackets
#...	Comment until end of line
//...	Comment until end of line
/*...*/	Comment block

2. Keywords

All elements of the world and their properties are addressed by **keywords**. In WDL keywords have a similar role as variables in a programming language. Some keywords are predefined. The keywords for files, objects, textures and so on may be defined by the user.

Keywords are generally defined and assigned a value by a line in the WDL script which looks like this:

TYPE Keyword *value*;

or

TYPE Keyword { ... *more keywords* ... }

Keyword is any name consisting of up to 30 letters. Names must not begin with numbers nor contain any special characters except the underscore `_`. Nor are you allowed to allot the same keyword to different objects. TYPE stands for the keyword type, *value* is for the assigned content, possibly a number or more keywords, respectively a complete list of keywords between winged brackets `{...}`. The keyword definition may stretch over some lines and is closed with a semicolon. These are the basic types for user-defined keywords (in alphabetical order):

ACTION for a real time action - a listing of instructions that may be triggered during game through an event, and may change the properties of walls, things, actors or regions,

ACTOR for an animated object, a state machine with certain behaviour primitives and action and reaction capabilities,

BMAP for a bitmap (graphics file) in formats .PCX, .WEX, .LBM or .BBM or a rectangular section of such a bitmap,

FLIC	for an animation file in .FLI, .WEX or .FLC format (professional version only),
FONT	for a bitmap which is interpreted as a character set to represent numbers or text,
MODEL	for a textured 3D model file in format .MDL,
MUSIC	for a song file in format .CMF or .MID,
MSPRITE	for the mouse pointer overlay,
OVERLAY	for a screen overlay in order to show cockpits, weapons or tools over the rendered scene in the 3-D window,
OVLY	for a bitmap assigned to an overlay,
PANEL	for a display panel with bar graph or numeric displays,
REGION	for a region in the map bordered by walls,
SKILL	for a numeric property,
SOUND	for a sound file in .WAV or .WEX format,
STRING	for raw text, e.g. for descriptions, menus or dialogues,
SYNONYM	for a blank 'template' to fill in,
TEXT	for formatted text,
TEXTURE	for an animated texture, which may be assigned to floors, ceilings, walls, things or actors,

THING	for a tool, a weapon or other inanimate object,
WALL	for a wall separating regions,
WAY	for the way an actor covers within a level,
VIEW	for a 3D window into the virtual world.

The *value* to be assigned to the defined keyword dependent on the type of the keyword. The following values may be assigned:

Number Assigns a number to the keyword. Numbers are integers of up to five digits (e.g. 12345) or fixed point numbers of up to three digits following the decimal (e.g. -12345.678).

Number,... Number
Assigns a list of several numbers, separated by comma, to a keyword.

Keyword Assigns another previously defined keyword to a keyword.

Keyword, ... Keyword
Assigns a list of several previously defined keywords to a keyword.

"Text" Assigns a text string to a keyword. Line feeds are written in C notation as `"\n"`.

- < *Filename*> Assigns the content of a file to a keyword. The kind of file is apparent from the extension. Valid extensions are < .LBM> , < .BBM> , < .PCX> for bitmaps or palettes, < .FLI> , < .FLC> for animations, < .MDL> for 3D model files, < .MID> for songs, < .IBK> for instruments, < .WAV> for sound effects, and < .WEX> for graphics, sound or animation files to be read from outside the resource. All files in pointed brackets will be bound compiled into a world resource by the compiler.
- {...} Assigns a complex description consisting of a listing of parameter assignments to the keyword. This description can have any number of lines and is closed by brackets.

Subsequently the predefined keywords are written in CAPITALS; user defined keywords, parameters or numbers are written in *italics*. Keywords whose values may be changed during gameplay through actions are marked by an arrow (►) Keywords whose values automatically change after each frame and can be evaluated by actions and may not be predefined are indicated by a rectangle (#).

3. Files

Keywords may be assigned to files to define bitmaps, animations, sounds or songs. Files that belong to the game and have to be compiled must be stated without path in pointed brackets < ...> . The following file keywords exist:

BMAP *Keyword*,< *Filename*> ;

BMAP *Keyword*,< *Filename*> ,*x,y,dx,dy*;

Assigns a 256 color bitmap - a graphics file - to the keyword, in the format .PCX, .LBM or .BBM. A .PCX file can be renamed to .WEX; in this case the file is first looked for outside the resource. The second form assigns a rectangular section from a bitmap to the keyword. The coordinates *x,y* mark the upper left corner of the section, *dx,dy* give the height and width in pixels. Omitting the coordinates causes the entire bitmap to be assigned to the keyword.

OVLY *Keyword*,< *Filename*> ;

OVLY *Keyword*,< *Filename*> ,*x,y,dx,dy*;

Like BMAP; except that it assigns an overlay to the keyword. Overlays are for showing cockpits, weapons, mouse pointers or similar objects over the rendered scene. They consume two or three times as much RAM as a bitmap, but they can be drawn much faster. The number of horizontal pixels of the overlay bitmap must be divisible by 4.

FONT *Keyword*,< *Filename*> ,*width,height*;

Assigns a set of characters to a keyword. *width, height* gives the size of the single characters in pixels. All characters must have the same size. The bitmap can either contain 11 characters - numbers 0..9 and space - for numeric displays, or the 128 or 256 characters respectively of the PC or ASCII set of characters for displays and texts. The sequence of alphanumeric characters in the bitmap must correspond to the PC (ASCII) character set. In the bitmap, the characters may appear in several lines. The number of characters and their arrangement is automatically determined from the character size and the size of the bitmap. The bitmap size must exactly be either 11 times, 128 times, or 256 times the given character size.

MODEL *Keyword*, < *Filename*> ;

Assigns a 3D polygonal model in .MDL format. You can use a freeware or shareware MDL tool, e.g. QMe or Meddle, to create the MDL files, and import DXF meshes. For creating palettes for the skin, use PCX2PAL to create a raw 768-byte PALETTE.LMP for QMe (simply copy it into the main directory), or Paintshop Pro to create a JASC .PAL file for Meddle.

The MDL file must obey the following restrictions: Not more than 1024 faces, the lesser the better; one skin only, which should be adapted to the level palette; only triangles as polygons; polygons must not penetrate each other; certain 'critical' polygons (especially long narrow ones) may have to be split into smaller ones if they are displayed in wrong order.

SOUND *Keyword*, < *Filename*> ;

Assigns a sound file in .WAV format to the keyword. A .WAV file can be renamed to .WEX; in this case the file is first looked for outside the resource. The sample rate may be 11 or 22 kHz. The DOS engine (that includes WED) accepts only 8-bit .WAV files, the windows engine 16-bit files too.

MUSIC *Keyword*, < *Filename*> ;

Assigns a song file in MID format to the keyword. The internal predefined instruments correspond to the general midi standard.

FLIC *Keyword*, < *Filename*> ;

Assigns an animation file in .FLI or .FLC format to the keyword. A .FLI file extension can also be named .WEX; in this case the file is first looked for outside the resource. This is senseful for very huge .FLI files, which are played a little faster as separate files outside the resource file.

4. Predefined Keywords

The following keywords, which must be given at the beginning of the WDL script, are used to define the basic modes for graphics and sounds:

VIDEO ***Keyword;***

Sets the screen resolution. The keywords for the following resolutions were predefined:

- 320x200** VGA resolution 320x200 pixels.
- X320x240** VGA resolution 320x240 pixels (mode X).
- X320x400** VGA resolution 320x400 pixels (mode X).
- S640x480** SVGA resolution 640x480 pixels (commercial version or above).
- S800x600** SVGA resolution 800x600 pixels (professional version).

The low resolutions are meant for fast action games, the high 'S' resolutions for adventures or commercial applications. Please note for the windows version that the 'X' resolutions are not supported by the DirectX drivers of some graphics cards.

NEXUS ***Number;***

Sets the size of the nexus, which refers to the internal data structure for picture rendering. The nexus size depends on the number of objects, walls and regions that will be visible consecutively. The bigger the nexus, the more complex scenes may be shown, but the more memory is needed. The given value for the nexus is 14. A nexus too small for a scene is indicated by an engine error message.

CLIP_DIST *Number;*

Default value for the region **CLIP_DIST** (usually 1000 steps). Only walls and objects that are entirely within the distance from the player position given in **CLIP_DIST** will be rendered. In individual regions you may define individual **CLIP_DIST**s that differ from the given value, which can accelerate the rendering of the picture in complex levels by up to 30%. Walls outside **CLIP_DIST** will be rendered monochromatically using the palette colour 1. A **CLIP_DIST** that is too small may cause image errors. This may sometimes be a desirable effect, e.g. with regions where everything beyond a certain distance is supposed to dissolve into darkness or fog.

LIGHT_ANGLE *number;*

This parameter allows you to define the direction a infinitely distant source of light is in within the range of 0..2 π . All walls with an **ALBEDO** type texture (see below) will reflect the light from the direction given according to their orientation. The value of 0 here corresponds to east (sunrise), the value of 3.145 corresponds to west (sunset).

DITHER *number (0 or 1);*

Using this parameter (default= 1= on) the 'dithering' effect can be switched off or on. With dithering smoother light and shadow ranges can be achieved.

IBANK *< Filename> ;***DRUMBANK** *< Filename> ;*

Loads the instrument bank, to be used for midi songs. *Filename* is an ADLIB instrument file in the .IBK-format. Percussions are put out through midi-channel 10. If no **DRUMBANK** is specified, only 5 standard-drums will be available; if no **IBANK** is specified, internal general midi compatible instruments will be used by the program.

MIDI_PITCH *Number;*

Number of octaves by which the pitchbend parameter changes the pitch on midi songs; values from 1..12, default is 2.

INCLUDE < *Filename* > ;

Reads additional WDL definitions from the separate WDL file < *Filename* > and then continues scanning the original WDL file. This way definitions common to all levels can be summarized in separate WDL files, which will be read using the INCLUDE keywords from the main WDL for every level. In the main WDL file any number of INCLUDE-keywords are allowed.

BIND < *Filename* > ;

The given file will included in the game by the compiler, e.g. for changing levels. You may give any number of BIND files.

MAPFILE < *Filename* > ;

Defines the corresponding WMP topographic file. This keyword must be included in the main WDL and cannot be read by way of INCLUDE.

SAVEDIR "*Dirname*";

Names the directory for saving games. If this directory does not exist, it will be created when first saving the game. Note that backslashes ("\") have to be given in C-notation, i.e. as double backslashes (e.g. "C:\\gstudio\\mygame"). The directory name given here may be replaced by the command line option **-DIR** with the VRUN.EXE during run time.

PATH "*Dirname*";

All additional files - bitmaps, sounds or midi - will be first searched for in the actual directory and then in the path given here (again state backslashes as "\\"). You can specify up to 16 **PATH** keywords, which will be searched in the given sequence.

DEFINE *NewName*[,*OldName*];

This keyword, like the **-d** command line option (see above), allows you to rename keywords and parameters in the WDL file, and to include or exclude WDL lines depending on conditions (nothing new for C-programmers, see below). Renaming does not affect the function of the respective values, but it does make rather large WDL actions a lot more 'readable', once the object skills or numbers are given meaningful names.

Example:

```
DEFINE MyNumber, -3456;
DEFINE HitEffect, SKILL3;
ACTION kill {
    SET    RESULT, M. HitEffect;
    SET    MyActor. HitEffect, MyNumber;
}
```

Characters like "{ ,;()< > " cannot be redefined, and **DEFINEs** may not be nested.

UNDEF *Name*;

A keyword defined by **DEFINE** can be 'undefined' by **UNDEF** for all following WDL lines.

IFDEF *Name*;**IFNDEF** *Name*;**IFELSE**;**ENDIF**;

These keywords are to skip WDL lines dependent on previous **DEFINEs** or command line option -d. All WDL lines between **IFDEF** and **ENDIF** are skipped if the **IFDEF** parameter was not **DEFINED** before. All lines between **IFNDEF** and **ENDIF** are skipped if the **IFNDEF** parameter was **DEFINED** before. The keyword **IFELSE** reverses the line skipping or non-skipping.

Example:

```
DEFINE hires; // alternatively start WRUN -d hires
IFDEF hires;
    VIDEO    S640x480; // SVGA high resolution
IFELSE;
    VIDEO    X320x400; // VGA mid resolution
ENDIF;
```

5. Palettes

Since the pixels of 256-color bitmaps contain no real colors, but only color numbers, you need a *palette* for your game to define which visible colors are represented by the color numbers. The palette thus contains a graphics file whose colors are taken as a reference. Your paint program stores the palette into the graphics file; that means you could theoretically give each texture its own palette. This, however, may make your textures look psychedelic, if their individual color palettes differs from the basic level palette. So all textures in your level should share the same palette. For this each paint program offers a menu function 'load palette' or 'adapt palette' or the like.

Palettes are responsible not only for the color representation, but also for **shading** - the darkening or 'fogging' of textures depending on the surrounding light and the distance from the player. To perform shading the colors in the palette have to be sorted in groups - *shading ranges* - of declining brightness (or inclining 'foggyness'). Textures in great distance or utter darkness will be given the the second colour of the palette.

The first three colors have a special meaning: The first one (color #0) corresponds to transparency, the second one (color #1) to darkness or foggyness, i.e. the color for wide distances, the third one (color #2) is the 'brightness' color of the lightsources in the level.

You can use only one palette at a given moment; but you can switch or fade between palettes at any time. The last defined palette in the WDL file will be used at game start. With the following definition a palette of 256 colors may be created from any graphics file in the format .PCX, .BBM, or .LBM:

PALETTE *Keyword* { ... }

Assigns a definition of the palette to the keyword. The definition may contain - in the sequence given - the following keyword assignments between winged brackets:

PALFILE < *Filename* > ;

Determines the 256-color graphics file (PCX,LBM, or BBM) whose colors are used for the basic palette. Please note that under windows the 1st color (#0) must be black and the last (#255) must be white.

RANGE *s/l*;

Shading range in the basic palette. A shading range is a group of the same colors with continually declining levels of brightness. *Sart* gives the number of the initial color (1..254), *Length* the number of colors in the range (up to 255). The initial color of the range must be the brightest, the last one the darkest color, which nearly ought to equal the **INFINITY** color #1.

Up to 24 shading ranges may be defined per palette. If there are no ranges defined, shading will be switched off, which results in slightly faster rendering. Colors outside all ranges are not shaded, so they can be used to represent sources of light.

FLAGS *Keyword1,Keyword2...;*

Here a list of keywords (flags) determining the properties of the palette may be given. The following flags may be set:

HARD

Using this keyword causes a 'hard' shading to be performed. The transitional area between bright and dark is smaller, colors will not be converted but merely shifted. However, this causes darkened textures to be displayed with greater contrast. Details of textures thus are better perceived.

AUTORANGE

This flag causes automatic shading of all colors on the palette that are included in any of the **RANGE**s given above, even if they are not sorted. The colours will be shaded according to their similarity to the **INFINITY** colour - the second color of the palette. This allows you to attain shading or 'fogging' effects even with entirely unsorted palettes. Nevertheless does a manual sorting of the colors result in a more differentiated colours **RANGE** in general and especially a more controllable shading.

BLUR

Setting this flag initiates the motion blur effect 'softer'. During movements the 3D image will be shown slightly out of focus, which moderates the 'pixeling' of textures rich in contrast. It also allows you to represent textures as semi-transparent (**DIAPHANOUS**). Disadvantage: start-up of the game and changing palettes takes about 2 seconds longer per **BLUR** palette. If all level's palettes are similar, setting the flag **BLUR** for the main palette only will suffice.

Example:

```
PALETTE my_pal {  
    PALFILE    <raw.pcx>; // unsorted palette, 2nd color = black  
    RANGE  16,224;    // shade all colors from #16 up to #240  
    FLAGS  BLUR, AUTORANGE; // don't bother about ranges  
}
```

6. Textures

Textures set the appearance of the world. There are textures for walls, floors, ceilings, backgrounds (sky textures), things, actors, panels and overlays which are all defined as follows:

TEXTURE *Keyword* { ... }

This assigns the keyword a texture definition. A texture definition may contain - in the given sequence - the following keyword assignments between winged brackets:

SIDES *Number;*

Number of sides of the texture. Each side is the equivalent of one or - for animated textures - several bitmaps. Depending on the object the texture was assigned to, the significance of the sides changes. Usually, bitmaps assigned to sides change depending on the angle from which the respective object is perceived. Hereby things and actors are given a spacial appearance. Texture for things and actors may have an unlimited number of sides, which are counted clockwise, the first side being equivalent to the frontal view.

Wall textures usually have only one side. Two-sided wall textures change the bitmap depending on the perspective the wall is perceived from: starting from the first vertex of the wall up to the middle the first side, from the middle to the end vertex the second side is visible. This way spatial elements of textures are shown more clearly. If the wall has four sides, the first two are assigned to its right side or front, the last two to its left side or back.

More than four sides are sometimes useful with multi-storied regions (see below): the first four sides are assigned to the uppermost region, the **BELOW** region has the next four sides and so on.

Sky textures may have any number of sides; they will be distributed evenly among the 360° panorama. For example does a **SKY** texture consisting of ten pages assume an area of 36° for each bitmap. Therefore the horizontal scale of sky textures is derived from the size of the bitmaps and the number of sides. There is no vertical scaling of sky textures.

CYCLES *Number;*

Number of phases (frames) of texture animation; default= 1.

FRAME *Number;*

Number of the start frame of an animated **MODEL** texture, between 1 and the max number of frames within the MDL file (default= 1).

BMAPS *Bitmap1, Bitmap2...;*

List of keywords, which were assigned the bitmap files of the texture. All bitmaps of wall, floor, or ceiling textures must have the same size, the bitmaps of actor's or thing's textures may differ in size. The number of bitmaps is unlimited, but must be equal to the product of (SIDES * CYCLES). In the **BMAPS** list the cycles of the front side are to be listed first, then those of all other sides counterclockwise around the object. Instead of the keyword of the bitmap the keyword **NULL** may also be used, but only with textures for things, actors or transparent walls. The respective cycle or side of the texture will then not be displayed, as if the object were invisible. In the case of walls at least one bitmap of the texture must be non-**NULL**.

The maximum width or height of a bitmap is 1024 pixels. Apart from that, a bitmap of a thing or actor texture can have any size. Bitmaps in the LBM format must have an even-numbered width. Wall textures can have any size horizontally; the vertical number of pixels must be a power of two, e.g. 64, 128 or 256 pixels. Floor and ceiling textures must be quadratic, only sizes of 64x64, 128x128 or 256x256 pixels are allowed.

Backdrop bitmaps (**SKY**) may have any horizontal size; the vertical size depends on the maximum visible background section (depending on the vertical angle **PLAYER_TILT**, see below). Usually the sky bitmap's lower edge touches the horizon. You can shift sky textures vertically and horizontally through the object's **OFFSET_X** and **OFFSET_Y** parameters and additionally through the predefined skills **SKY_OFFS_X** and **SKY_OFFS_Y** (see below). Vertically the sky texture is neither zoomed nor repeated. If the texture's border exceeds the top or bottom pixel line of the sky bitmap, the rest of the sky is automatically filled with this pixel line; so the first and last line of the sky bitmap should be monochrome.

FLIC *Flic*;

Texture FLIC animation, alternative to **BMAPS** (professional version only). *Flic* is a previously defined keyword denoting a FLI/FLC file. The size of the texture results from the size of this animation. The same rules concerning size that were given for **BMAPS** apply here. The *Flic* palette must correspond to the level-palette. As long as **ONESHOT** is not set, the animation is repeated in a loop.

TITLE *String*;

With the parameter **TITLE** a texture can be given a string, who is displayed with the texture FONT (see below). The string may be given as keyword or directly (TITLE "It's a title!"). The **TITLE** has the same restrictions as a bitmap would have for the same texture; e.g. the font height for a one-line title on a wall texture must have be power of two. If a texture has a **TITLE**, it must not be animated, and must have no **BMAPS**.

```
TEXTURE title_tex {  
    FONT    my_font;  
    TITLE   "That's a title!";  
}
```

MODEL *Model*;

3D animated model, alternative to **BMAPS**. *Model* is a previously defined keyword denoting a **MDL** model file. A model texture can only be assigned to things or actors. The size of the texture results from the positions of the model vertices (in pixels); the skin bitmap palette must correspond to the level palette. The same rules concerning size that were given for **BMAPS** apply here. Only one **DELAY** value is allowed. As long as **ONESHOT** is not set, the model animation is repeated in a loop. If you want to use only a part of the MDL frames for the animation, use the **FRAME** and **CYCLES** parameters. Example:

```
TEXTURE hero3d_tex {  
    MODEL   hero_md1;  
    DELAY   2;  
    SOUND   hero_snd;  
}
```

DELAY *Number, Number....;*

List of phase durations in ticks for every cycle of animation. The number of **DELAY** values must be equal to the value of **CYCLES**. If this list is not stated, animation is working with one tick per phase. The higher the **DELAY** value, the more does the animation speed remain constant on different frame rates.

MIRROR *Flag, Flag....;*

List of mirror flags (0 or 1) for every texture side. If the mirror flag is set at 1, all bitmaps of the concerned side are shown side-inverted. The number of flags must be equal to the number of **SIDES**. If this list is not given, all bitmaps are shown un-inverted.

OFFSET_X *Number, Number...;***OFFSET_Y *Number, Number...;***

Each bitmap of an animated thing or actor texture may be given an individual shift in horizontal and vertical direction by these parameters in pixel units. The number of **OFFSET_X/Y** parameters must be equal to the number of **BMAPS**. Example:

```
TEXTURE animthing_tex {  
    SCALE_XY 10, 10;  
    BMAPS    thing_map, thing_map, thing_map, thing_map;  
    DELAY    6, 6, 6, 6;  
    OFFSET_X 5, 10, 20, 40;  
    OFFSET_Y 5, 10, 20, 40;  
}
```

RANDOM *Number;*

Maximum value for a random factor of delay (0..1, default 0), which is added to every phase of a texture animation.

►SCALE_X *x;***►SCALE_Y *y;***

Scaling of the textures in horizontal and vertical direction in pixels per step. With floor and ceiling textures the value *x* indicates the scaling in the direction of X, *y* indicates the scaling in the direction of Y within the system of coordinates. The smaller the value given, the larger the texture will be displayed; the larger the value, the higher the pixel resolution in the object's

proximity. Should this keyword be omitted, the textures will be scaled by 16 pixels per step in each direction. The keyword does not apply to **SKY** textures.

► **AMBIENT** *Number;*

Brightness of the texture (-1 .. 1, 32 levels, default 0). At 0 the wall texture is only illuminated by the player's lightsource and the ambient light of the region; at 1 the texture itself shines brightly. Negative ambients reduce the power of reflexion of a texture; this could be used to simulate areas covered by shadows.

► **ALBEDO** *Number;*

The reflexion factor of a texture (0...1, default 0) respective to light coming from **LIGHT_ANGLE** or a separate source of light (cf. **GENIUS**). Buildings, columns, or ledges appear more spatial when given a texture **ALBEDO** (ca. 0.3).

RADIANCE *Number;*

Through this parameter (0..1, default 0) the maximum brightness of a texture can be determined. With high ambients, textures with a **RADIANCE** > 0 get colored with the 'brightness' color of the palette (#2).

► **SOUND** *Sound;*

Keyword which has been assigned a sound file (.VOC or .WAV). On floor textures this sound is played rhythmically while moving, on ceiling textures it will be heard - if SLOOP is set - continuously in the whole region.

► **SVOL** *Number;*

Maximum volume of the sound (0..1). If no value is given here, the sound will be played at volume 0.5.

►SDIST *Number;*

Critical horizontal range in steps within which the sound is audible (default 100). The closer one gets to the object to which the texture has been assigned, the louder it gets. With stereo soundcards you can locate the direction of the object from differing volumes on the right or left channel. Close to the texture the sound is played at the maximum volume determined by **SVOL**.

►SVDIST *Number;*

Critical vertical range in steps within which the sound is audible (default 0, i.e. inactive). Only effective with values > 0 and with things or actors. If the player is located above or below the object's height plus **SVDIST**, the sound is inaudible.

SCYCLES *Flag, Flag,...;*

List of sound trigger flags (0 or 1) for every phase of animation. If the sound flag is set at 1, the texture sound is played on the appropriate cycle. The number of flags must be equal to the number of **CYCLES**. If this list is not given, the sound initiates with the first cycle.

► **ATTACH** *texture*;

To attach an additional texture to a wall, thing or actor texture in order to show paintings, inscriptions, bullet holes, or shadows. The new texture is transparent. Another texture may be attached to the new texture in the same way.

In the case of walls any number of textures may be overlayed on the same wall - a whole row of bullet holes, for example. But please note that the chain must have an end! A texture which is - directly or indirectly - **ATTACH**ed to itself will cause an engine crash. In the case of things or actors only one single **ATTACH** texture is allowed, and the **ATTACH** parameter of the actor himself, not of his texture, must be used.

ATTACH textures may be displayed **DIAPHANOUS** (see below) or semi-transparent (**GHOST**). They may be animated, if the original texture also is animated. The number of **CYCLES** of the original and the **ATTACH** textures must correspond. The original texture's **DELAY** and **RANDOM** values apply to the animation. If only the **ATTACH** textures are meant to be animated, the original texture may be assigned a 'dummy-animation' of identical bitmaps.

► **POS_X** *number*;

► **POS_Y** *number*;

The **ATTACH** texture appears at the pixel positions given by these keywords relative to the upper left corner of the original texture. If both **POS_X** and **POS_Y** are 0 (default), they will both appear exactly in the upper left.

Advice: In order to position an **ATTACH** texture exactly in the middle of a wall independently from the height and scaling of the wall, the position should be calculated as follows (rule-of-thumb):

POS_X = **OFFSET_X** + **SIZE_X**/2 - **WA**/2

POS_Y = **HW****OFFSET_Y** - **HA**/2 - (**CEIL_HGT**+**FLOOR_HGT**)***SCALE_Y**/2

POS_X, POS_Y: Position of **ATTACH** texture in pixels,

WA, HA: Width and height of **ATTACH** texture in pixels,

OFFSET_X, OFFSET_Y: Offsets of wall in pixels,

HW Height of wall texture in pixels,

SCALE_Y: Scaling factor of wall texture,

SIZE_X: Length of wall in pixels,

CEIL_HGT, FLOOR_HGT: Height of ceiling and floor.

PORTCULLIS or **FENCE** walls allow for a much simpler way to calculate. Given a wall **POSITION** of 0 the vertical zero of an **ATTACH** texture refers to the lower edge of the wall, i.e. the attach texture needs to be shifted upwards by a negative **POS_Y** to become visible. If **POSITION**=1, the vertical zero of the **ATTACH** texture refers to the upper edge of the wall.

The following parameter determine the response to touch or clicking on with the mouse of objects that were given the respective texture. The skill **TOUCH_RANGE** allows you to specify the maximum distance within which the object will still react to the mouse in steps. Objects with the flag **IMMATERIAL** set are 'invisible' to the mouse. **ATTACH** textures may also respond to the mouse, but only the last (uppermost) texture is relevant here.

► **TOUCH** *string;*

The given text string is displayed as soon as the mouse pointer touches the wall, thing or actor the texture is assigned to and the predefined skill **TOUCH_MODE** (see below) is above or equal to 1. You can give the maximum object distance with the predefined skill **TOUCH_DIST** (default 100). The appearance of the text can be influenced by the synonym **TOUCH_TEXT** (see below).

FONT *font;*

Character set for the **TOUCH** or **TITLE** string.

► **IF_TOUCH** *action;*

The given action is triggered by touching the texture with the mouse pointer.

► **IF_RELEASE** *action;*

The given action is triggered by leaving the object's texture with the mouse pointer.

► **IF_KLICK** *action;*

When touching an object with the mouse pointer, by left clicking on the object this action is triggered instead of the 'common' **IF_KLICK** action.

FLAGS *Keyword1,Keyword2...;*

Here a list of keywords (flags) may be given, which determine the properties of a texture. The following flags can be set:

ONESHOT

If this flag is set, the texture is not permanently animated. It normally shows the first phase of animation. By setting of the flag **PLAY** to the assigned object (wall, thing, actor or region - v.) the texture can be animated for one cycle and it will stop at the last phase.

GHOST

If this flag is set, the texture is shown semi-transparent on things, actors or transparent walls.

DIAPHANOUS

Like **GHOST**, except that the textures will be represented diaphanous and not dithered. Yet, this flag only works with a **BLUR** palette and increases computing time for the respective textures by ca. 30%.

BEHIND

The **ATTACH** texture appears behind the original texture (e.g. with things oder actors).

SHADOW

The **ATTACH** texture appears at the actor's floor height. Using a dark **DIAPHANOUS** texture this way actor shadows can be generated.

Example:

```
TEXTURE shadow_tex {
    BMAPS    shadow_map;    // flat black spot (2nd color)
    FLAGS    DIAPHANOUS, BEHIND, SHADOW;
}
ACTOR guy {
    TEXTURE    guy_tex;
    ATTACH    shadow_tex;
}
```

LIGHTMAP

The ATTACH texture flag LIGHTMAP maps light and shadows onto a wall texture. The LIGHTMAP bitmap must contain only the colors #0..#15 and #241..#255 (otherwise: engine abort!). The transparent color #0 does nothing, colors #1..#15 darken the wall texture, colors #255..#241 brighten it (#241 = brightest). The darkest color is taken from the level palette color #1, the brightest color from the palette color #2, which should be a bright white or yellow. Example:

```
BMF torch_map, <torchlight.pcx>;
```

```
TEXTURE torch_tex {  
    BMF      torch_map;  
    FLAGS    LIGHTMAP;  
    POS_X    40;  
    POS_Y    10;  
}  
WALL stone_wall {  
    TEXTURE  stone_tex;  
    ATTACH   torch_tex;  
}
```

SKY

If this flag is set, there ensues no perspective, but a parallax projection of the texture. It will therefore be shown as 'background' (for mountains, sky, horizon etc.). **SKY** textures will not be zoomed, so that they will appear always at an infinite distance. You can assign **SKY** textures to walls, floors, and ceilings.

WIRE

For **MODEL** textures only: Displays the **MODEL** as a non-textured wireframe with **MAPCOLOR**.

CLUSTER

For **MODEL** textures only: Displays only the vertices of the **MODEL** polygons.

NO_CLIP

This flag forces thing's and actor's textures not to be cut off on the floor or ceiling of a region.

CLIP

This flag forces thing's and actor's textures to be cut off on the floor or ceiling of a region, and thus prevents objects to be visible through the ceiling of **BELOW** regions.

SLOOP

If this flag is set, texture sounds will be played continuously in a loop, independently of the animation phase. Volume and direction are continually adapted. As this does take up quite some computing time especially with wall textures and all sound channels will be continually busy within the texture's earshot this flag should be used economically.

CONDENSED

Setting this flag compresses the mouse text string (**TOUCH**) horizontally by 1 pixel. Especially italics often look better that way.

NARROW

Like **CONDENSED**, only the textstring will be compressed further.

SAVE

This flag must be set in the texture definition, if the parameter of the texture are to be changed by an action during the game. The texture will then be saved with the score (**SAVE** command).

WDL example: Definition of a multi-sided sky texture for a landscape with two towers in the east:

```
BMAP hills <huegel.lbbm>;  
BMAP tower <turm1bm>;
```

```

TEXTURE  tower_landscape {
    SIDES  10; // bitmap spreads over 36° viewing sector
    BMAPS  tower, hills, tower, hills, hills, hills, hills, hills, hills, hills,
            hills;
    FLAGS  SKY;
};

```

2nd. **Example:** Definition of a multi-sided animated texture. It will show a swimming fish:

```

BMAP PIRAN_0_1  <fish1.bbm>; // Fish animated in 4 Phases,
BMAP PIRAN_0_2  <fish2.bbm>; // and drawn in 5 views,
BMAP PIRAN_0_3  <fish3.bbm>; // here from the front
BMAP PIRAN_0_4  <fish4.bbm>;
BMAP PIRAN_45_1 <fishvr1.bbm>; // from right front (45°)
BMAP PIRAN_45_2 <fishvr2.bbm>;
BMAP PIRAN_45_3 <fishvr3.bbm>;
BMAP PIRAN_45_4 <fishvr4.bbm>;
BMAP PIRAN_90_1 <fishr1.bbm>; // from right (90°)
BMAP PIRAN_90_2 <fishr2.bbm>;
BMAP PIRAN_90_3 <fishr3.bbm>;
BMAP PIRAN_90_4 <fishr4.bbm>;
BMAP PIRAN_135_1 <fishhr1.bbm>; // from right back...
BMAP PIRAN_135_2 <fishhr2.bbm>;
BMAP PIRAN_135_3 <fishhr3.bbm>;
BMAP PIRAN_135_4 <fishhr4.bbm>;
BMAP PIRAN_180_1 <fishh1.bbm>; // from the back.
BMAP PIRAN_180_2 <fishh2.bbm>;
BMAP PIRAN_180_3 <fishh3.bbm>;
BMAP PIRAN_180_4 <fishh4.bbm>;

```

```

SOUND  PIRAN_SND <shwssh.voc>;

```

```

TEXTURE  PIRAN_TEX {
    SCALE_X  20; // Scale 20 pixels per step
    SCALE_Y  20;
    SIDES    8; // Object has 8 sides
    CYCLES   4; // and 4 phases
    BMAPS    PIRAN_0_1, PIRAN_0_2, PIRAN_0_3, PIRAN_0_4,
            PIRAN_45_1, PIRAN_45_2, PIRAN_45_3, PIRAN_45_4,
            PIRAN_90_1, PIRAN_90_2, PIRAN_90_3, PIRAN_90_4,
            PIRAN_135_1, PIRAN_135_2, PIRAN_135_3, PIRAN_135_4,

```

```
        PIRAN_180_1, PIRAN_180_2, PIRAN_180_3, PIRAN_180_4,  
// these are mirrored  
        PIRAN_135_1, PIRAN_135_2, PIRAN_135_3, PIRAN_135_4,  
        PIRAN_90_1, PIRAN_90_2, PIRAN_90_3, PIRAN_90_4,  
        PIRAN_45_1, PIRAN_45_2, PIRAN_45_3, PIRAN_45_4;  
MIRROR    0, 0, 0, 0, 0, 1, 1, 1; // show last 3 sides mirrored  
DELAY     3, 3, 2, 2;           // Cycle lengths in ticks  
SCYCLES   1, 0, 0, 0;          // Sound starts at first cycle  
RANDOM     0.2;                 // additional random delay  
SOUND     PIRAN_SND;  
SDIST     30; // Sound only audible within 30 steps  
};
```

Hints for Designing Textures

You have several ways available to produce bitmaps, depending on the atmosphere you wish to create in the game. You may use paint software, render them using 3D software or scan them from models. Even with the last two options you will most certainly have to work on the bitmaps manually and adapt them to your level palette.

All bitmaps for things, actors, and walls - with the exception of sky, panel, and overlay bitmaps - are zoomed. As that means that single rows of pixels are suppressed depending on distance, adjacent single pixels should not display great contrasts in color and brightness (on scanned bitmaps this is automatically the case). Dithering, lattice structures, thin lines and so on cause flickering and moiré effects and should be avoided. If sharp lines and rich contrasting edges are required they should blend in smoothly with the background on one side at least.

Wall and floor textures should normally fit together on all sides. Exceptions are such textures that cover a wall or floor completely and without repetition. The vertical size of a sky texture should allow for its upper and lower borders never to become visible in the field of vision. Otherwise the upper border should be of a single color, as it will be repeated on leaving the field of vision. Thing and actor bitmaps should be cut out with at least one pixel 'air' on all sides.

For the size of bitmaps some rules should be noted (see above). As textures may be scaled, their bitmap sizes do not determine their sizes in the game. Generally a texture bitmap should only be larger to show more texture details, not a larger image. Things and actors the player comes close to should have large bitmaps, floor and ceiling textures smaller ones correspondingly. As a general rule textures which are far away from the player should be scaled by a **SCALE_X/Y** less then 10; textures which may come close should be scaled by 20 or higher.

If the length of a wall or the scale or light value of a texture is changed by an action during gameplay, with all objects concerned (walls, things, or actors) the texture has to be re-set through a **SET** instruction (see chapter 'Actions').

When generating **LBM** or **BBM** bitmaps with the paint software Deluxe Paint please note that the width of the bitmap needs to be even-numbered. On saving of LBM bitmaps with Deluxe Paint II please note that the button 'old' at the menu SAVE AS must be deactivated. Allmost all converter programs (GWS, Paintshop Pro) save the LBM bitmaps with the 'old' format, which cannot be read from ACKNEX. Please use **PCX** instead.

Hints concering Sounds

The **WAV** sounds employed must be saved in 8 bit (DOS) resp. 8 bit or 16 bit (windows 95) mono format. ACKNEX itself will take care of the stereo effect. The **WAV** files should not contain any special events like *loop-points*, *regions* or *play lists*. For normal sounds usually a sampling rate of 11 kHz should suffice. High-frequency sounds (hissing, rushing or splashing) may be sampled at 22 kHz.

7. Walls

Vertical walls run along the connecting lines between two vertices. They separate the regions of the map. Their properties are defined in the wall definition:

WALL *Keyword* { }

The wall keyword defined in such a fashion may later be assigned to a connecting line with the WED. A wall definition may contain - in the given sequence - the following keyword assignments:

► TEXTURE	<i>Texture;</i>	Texture for the wall, up to 4 sides. On two-sided textures the bitmap changes depending on which vertex of the wall is closer to the viewer. Four-sided textures always differentiate between the right and left side of a wall.
► ATTACH	<i>Texture;</i>	Superimposed texture for the wall; only one side. The number of cycles (CYCLES) must either correspond to the wall texture's or equal 1.
► OFFSET_X ► OFFSET_Y	<i>Number;</i>	Shifts the wall texture OFFSET_X pixels to the left and/or OFFSET_Y pixels down. OFFSET_X must not be negative. If one of these keywords is used, the wall texture may not be shifted using the ALIGN function in WED. For sky textures OFFSET_Y is the distance of the horizon from the lower border of the bitmap.
► CYCLE	<i>n;</i>	The current animation phase of the texture.
► POSITION	<i>Number;</i>	The use of this keyword is dependent on the flags (see below).

MAP_COLOR	<i>n;</i>	Mode (0...1, default 1) for drawing of the wall on the automap. If 0 no map will be drawn; if 1 the wall will be drawn antialiased in a default color (COLOR_WALLS , COLOR_BORDER - see below).
► DIST	<i>Number;</i>	<p>Border distance of the nearest vertex of the wall to the player, default=0. By crossing this border distance an IF_NEAR or IF_FAR action (see below) may be triggered.</p> <p>If the DIST is set to 0 (default), an IF_NEAR action can be triggered simply by touching or passing through the wall. If the DIST is > 0, but less than half the length of the wall, an IF_NEAR action will not be triggered near the center of the wall, as only the distance to the nearest vertex is relevant!</p>
► SKILL1... ► SKILL8	<i>Number;</i>	Eight 'universal parameters', which initially have no influence on the properties of the wall. They can be changed and evaluated by an action.

The following parameters are set automatically, but may be evaluated in actions:

#DISTANCE	<i>Number</i>	Approximate ($\pm 20\%$) distance of the player to the nearest vertex of the wall; only valid for walls within the player's CLIP_DIST . May be evaluated by an action.
#LENGTH	<i>Number</i>	The length of the wall in steps.
#SIZE_X	<i>Number</i>	Length of the wall in pixels, depending on the scaling of the texture.

#LEFT #RIGHT	<i>Region</i>	The regions at the left and right side of the wall (to be evaluated in actions)
►X1, ►Y1, ►Z1	<i>Number</i>	Position of vertex1 of the wall. These parameters can be changed by an action, in order to shift or rotate walls or regions. Be aware that walls may not cross each other. If the length of the wall changes, a SHAKE command (see below) must be executed.
►X2, ►Y2, ►Z2	<i>Number</i>	Position of vertex2 of the wall.

The following keywords allow you to assign actions, which will be triggered by certain events associated with the wall:

►IF_NEAR	<i>Action;</i>	This action is triggered as soon as the player crosses the border distance (DIST) of the nearest vertex. If no border distance is defined, the action will be triggered by every touching or passing through of the wall.
►IF_FAR	<i>Action;</i>	This action is triggered as soon as the player distances himself from the wall beyond the border distance (DIST).
►IF_HIT	<i>Action;</i>	This action is triggered if the player hits the wall with the SHOOT -instruction or if the wall is hit by an exploding object.
►EACH_CYCLE	<i>Action;</i>	This action is triggered after every animation cycle of the wall texture, or after the end of an ONESHOT animation.
►EACH_TICK	<i>Action;</i>	This action is triggered after every frame rendering cycle (i.e. approx. every 1/16 second).

The following keyword allows you to set a whole list of further keywords (Flags) determining the properties of the wall. During actions flags take on either the value of 1 (=set) or 0 (=not set).

FLAGS	<i>Flag1, Flag2...;</i>	The following flags may be given (set) in this listing or evaluated or changed by actions:
► INVISIBLE		If this flag is set, the wall is invisible, but impenetrable. The same region must be on both sides of an invisible wall.
► PASSABLE		If this flag is set, the wall is passable (permeable) to the player and actors.
► IMMATERIAL		After setting this flag, the wall will no longer be influenced by SHOOT instructions or mouse clicks.
► IMPASSABLE		If this flag is set, the wall becomes impenetrable over its entire length and width, even if it is not visible. If the PASSABLE flag is set simultaneously, the wall remains passable for the player, but not for actors.
#VISIBLE		This flag is set automatically, as long as the wall is seen by the player. It can be evaluated in actions.
► SEEN		This flag is set automatically, as soon as the player has once seen the wall, and then stays set. It will be evaluated by the automap function.
► BERKELEY		If this flag is set, the wall exists only as long as it is viewed or the player is within its border distance. This flag saves rendering time in levels with many animated wall textures.
► TRANSPARENT		If this flag is set, the color 0 of a wall texture is transparent, e.g. for fences or lattices. Please note that the same region must be on both sides of a transparent wall.

- **PLAY** If this flag is set, the wall texture will be animated for one cycle; it will then stop on the last phase. On the texture concerned the flag **ONESHOT** has to be set. At the end of the animation, the flag **PLAY** is set automatically back to 0, and an **EACH_CYCLE** action will be triggered if necessary.
- CURTAIN** If this flag is set, the wall reaches from the floor up to the ceiling, independent of the region heights.
- **PORTCULLIS** This flag 'fastens' the wall texture to the upper or lower edge of the wall; useful for walls that move vertically, e.g. elevators or roll-down doors. If this flag is set, the wall texture is automatically adjusted to the lower border (at **POSITION**=0) or the upper border (at **POSITION**=1) of the wall, so that it can be moved with the wall.
- FENCE** By setting this flag you can 'cut off' transparent walls at the upper edge of their texture. The wall parameter **POSITION** gives the depth (in steps) of the lower wall edge within the floor, a negative **POSITION** lets the wall 'float' above the ground. By changing **POSITION** you can raise or lower the wall like a lattice. With **FENCE** walls the flag **PORTCULLIS** is set automatically.
- **SENSITIVE** After setting this flag the corresponding wall will trigger ist **IF_NEAR** action already when being seen by the player.
- **FRAGILE** If this flag is set, the **IF_HIT** action of the wall may be triggered by an **EXPLODE** instruction (see below).

FAR	If this flag is set, the wall will also be visible outside CLIP_DIST (see CLIPPING skill); useful for distant SKY walls, as it allows you to cut down computing time by lowering CLIP_DIST . Walls with FAR set may not border BELOW regions.
SAVE	If this flag is set, all properties of the wall are saved through the SAVE instruction.
►FLAG1... ►FLAG8	Eight 'universal flags', which have no influence on the properties of the wall at first. They may be changed or evaluated by actions.

For reasons of mathematical inaccuracy with the representation of textures, walls should have a maximum length of 200 steps and height of 1000 steps. Longer walls have to be split into appropriate sections.

8. Regions

Regions are closed areas of the map, bordered by walls, that may be stacked vertically at will. Their properties are defined in the region definition:

REGION *Keyword* { }

The region defined with this keyword may be assigned to the right or left side of walls using the WED. A region definition may contain - in the sequence given - the following keyword assignments:

- | | | |
|--|-----------------|---|
| ►FLOOR_TEX | <i>Texture;</i> | Assigns a texture to the floor of a region. The texture sound plays on moving in the region, rhythmically accompanying either the WALK or WAVE cycles (see below), depending whether the texture's SLOOP flag was set or not. Volume or pitch may be changed using predefined skills (PSOUND_VOL , see below). |
| ►CEIL_TEX | <i>Texture;</i> | Assigns a texture to the ceiling of a region. The sound of the ceiling texture plays when entering the region; if the SLOOP -flag is set, the sound is continued independent of the movements of the player. |
| ►FLOOR_HGT
►CEIL_HGT | <i>Number;</i> | Height of the floor or ceiling of the region in steps; slanted regions add the height (Z) of the vertices to this value. The difference of height between floor and ceiling should not exceed 1000 steps. |
| ►FLOOR_OFFS_X
►FLOOR_OFFS_Y
►CEIL_OFFS_X
►CEIL_OFFS_Y | <i>Number;</i> | Shifting of the textures of floor or ceiling in direction of the X- or Y-coordinate in pixels, relative to the coordinates of the playing-field. |

- **FLOOR_ANGLE** *Number;* Texture angels of the floor and ceiling textures; range 0..2ð, default 0. This angle corresponds to one revolution of the texture around the center **GENIUS** (see below). Revolved floor or ceiling textures do of course take up more computing time and are only possible with regions that are nor slanted.
- **CEIL_ANGLE**
- BELOW** *Region;* Assigns a predefined lower region in order to display balconies, bridges, multistoried buildings or hovering objects or to define underwater regions. The lower region may be assigned another **BELOW** region, too, and so on. In the level all regions 'connected' by **BELOW** assignments are shown stacked one over the other.
- #TOP** *Region;* The uppermost region, if this region is a part of a chain of **BELOW** region; otherwise the region itself. Is set automatically on game start.
- **GENIUS** *Object;* Assigns either a Thing or an Actor to serve as a source of light for the walls of the region, alternatively to **LIGHT_ANGLE** (cf. **ALBEDO**), and as the rotational center fo the **ROTATE** commands at the same time.
The object may be **INVISIBLE**, but needs to have a texture **AMBIENT** > 0 in order to serve as a source of light instead of the global **LIGHT_ANGLE**. It may also be situated outside the region and be assigned to more than one region simultaneously.

► AMBIENT	<i>Number;</i>	Basic brightness of the region (range 0 .. 1, 16 degrees, default 0). The values of AMBIENT and of texture and region together with the player source of light (PLAYER_LIGHT) and the distance from the player form the texture's brightness.
CLIP_DIST	<i>Number;</i>	All walls at a distance larger than given (in <i>steps</i>) from the player position are not rendered. If this value is not given, the region will use the default CLIP_DIST (see above). Regions with interiors that are never visible - e.g. if the floor and ceiling touch - should be given a CLIP_DIST of 0.
► SKILL1... ► SKILL8	<i>Number;</i>	Eight 'universal parameters', which initially do not have any influence on the properties of the region. They can be evaluated or changed by actions.
► IF_ENTER	<i>Action;</i>	This action is triggered as soon as the player enters the region. Still, the player must remain in the region for at least one frame, which is especially important to keep in mind with small regions and fast moving players.
► IF_LEAVE	<i>Action;</i>	This action is triggered as soon as the player leaves the region.
► IF_DIVE	<i>Action;</i>	This action is triggered as soon as the eye level of the player (PLAYER_Z) reaches or falls below floor level.
► IF_ARISE	<i>Action;</i>	This action is triggered as soon as the player's eye level reaches or rises above ceiling level.

► EACH_CYCLE	<i>Action;</i>	This action is triggered at the end of every texture animation cycle or at the end of an ONESHOT animation of the floor or ceiling texture.
► EACH_TICK	<i>Action;</i>	This action is triggered after every frame cycle.
FLAGS	<i>Flag1, Flag2...;</i>	Here a list of keywords (flags) can be set, which determine the properties of the region. The following keywords are possible:
FLOOR_ASCEND CEIL_ASCEND FLOOR_ DESCEND CEIL_DESCEND		If one of these flags are set, the corner points of the floor and/or ceiling are lifted upwards (ASCEND) or downwards (DESCEND) by the Z-values of their vertices. Floors and ceilings can be tilted this way (see note below).
#VISIBLE		This flag is set automatically as long as a floor, ceiling or wall of the region is visible. May be evaluated by actions.
► SEEN		This flag is set automatically as soon as the region is once seen by the player.
SAVE		If this flag is set, all properties of the region will be saved on game saving.
SAVE_ALL		If this flag is set, all walls and things within this region are saved by SAVE instructions. This makes sense if the region is moved during gameplay e.g. by ROTATE .
#HERE		This flag is set automatically as long as the player is within this region.

- **BASE** This flag assures that the player's floor height (**PLAYER_HGT**) does not change when entering the region. Useful with narrow holes or slits in the ground the player is not supposed to 'fall' into.
- STICKY** If this flag is set, the player will be moved together with the region's floor by **SHIFT** or **ROTATE** instructions.
- **PLAY** If this flag is set, the floor and ceiling textures will be animated for one cycle; they will then stop on the last phase. The texture itself must have the flag **ONESHOT** set. At the end of the animation the flag **PLAY** is set back automatically to 0 and an **EACH_CYCLE** action is triggered if necessary.
- **FLAG1...** Eight 'universal flags', which initially have no influence on the properties of the region. They can be changed or evaluated by actions.
- **FLAG8**

Please note the following restrictions for **BELOW** regions:

- # Things and actors without **GROUND** flag are placed on the uppermost region at game start. Things and actors with **GROUND** flag are placed onto the lowest region whose ceiling is above the object's feet.
- # Walls of **BELOW** regions may not have the **FAR**, **PORTCULLIS** or **FENCE** flag set.

Underwater-Regions:

- # If the **CEIL_HGT** of a **BELOW** region corresponds exactly to the **FLOOR_HGT** of the region above, the **BELOW** region may serve as an 'underwater' region. The predefined skill **PLAYER_DEPTH** (see below) does, with such **BELOW** regions, contain the difference between the floor heights and thus the 'depth' of the water; with other regions it is 0. **PLAYER_DEPTH** may for example be used to reduce **PLAYER_SIZE** while wading using an action. If the upper region does contain an **IF_DIVE**-and the lower an **IF_ARISE** action, floor and ceiling become permeable to the player.

Example:

```
REGION under_water { // to be defined BEFORE water region
    FLOOR_HGT -5;
    CEIL_HGT 1;
    FLOOR_TEX mud_tex;
    CEIL_TEX underwater_tex;
    IF_ARISE water_arise;
}
REGION water {
    FLOOR_HGT 1; // same as under_water's CEIL_HGT!
    FLOOR_TEX water_tex;
    CEIL_HGT 20;
    CEIL_TEX sky_tex;
    BELOW under_water;
    IF_DIVE water_dive;
}

ACTION water_dive { FADE_PAL blue_pal, 0.5; }
ACTION water_arise { FADE_PAL blue_pal, 0; }
```

Please note the restrictions for slanted regions (**FLOOR_ASCEND**, **CEIL_ASCEND**, **FLOOR_DESCEND**, **CEIL_DESCEND**):

- # Textures are projected vertically onto slanted floors or ceilings, so that they will align seamlessly despite being slanted. The angle should not be chosen too high to avoid a distorted projection; additionally only textures low in contrast should be used. As the representation of slanted areas does use up computing time they should preferably be used with moderation.
- # With large floor or ceiling bitmaps (256x256 Pixel) the **SCALE_X/Y**-factor of the texture should not exceed the value 32, otherwise the texture may show a 'swimming' effect in the distance. Smaller bitmaps may be scaled with a higher factor.
- # When assigning heights (Z values) to vertices, the floor and ceiling textures of the region must not be 'bent'.
- # Several identical regions with a different incline are not allowed to border the same **BELOW** region. In this case for each region a different region name would have to be defined.
- # Walls separating slopes should manually be set **INVISIBLE** and **PASSABLE**.
- # Be careful with the topography to avoid a view directly rectangular (90 degrees) onto a slope side, otherwise distortions may be visible.

- # The floor of a slope must not penetrate the ceiling (keep 0.01 steps security distance).
- # The **AMBIENT** of a slanted region should be chosen as to make sure that the luminosity of floor and ceiling resulting from the texture **AMBIENT** and **PLAYER_LIGHT** does never exceed 1, and never acquire a negative value.

9. Things, Actors, Ways

Things are one-dimensional items with only one vertex (as opposed to walls). Their properties are defined in the thing definition:

THING *Keyword* { }

An actor differs from a thing only in its more complex action and reaction behaviour.

ACTOR *Keyword* { }

Defining the actor this way allows you to place him at any place on the map via WED . Additionally, you can assign him a **way**, i.e. a list of **waypoints**. Such a way definition simply consists of the keyword **WAY** followed by the way name which then can be assigned to a waypoint list in WED.

WAY *Keyword*;

The actor who has been assigned this way, "wanders" in a cycle from one waypoint to the next. No collision detection is carried out, except if the actor flag **CAREFULLY** is set (see below).

A thing or actor definition contains the following assignments in given sequence:

- | | | |
|------------------|-----------------|--|
| ► TEXTURE | <i>Texture;</i> | The texture keyword, either a bitmap or a model texture. On multi-sided bitmap textures the side changes depending on the angle from which the actor or thing is seen. |
| ► ATTACH | <i>Texture;</i> | ATTACH texture for the object. The number of CYCLES and SIDES must either correspond to the object's texture's or equal 1. Things and Actors may only be assigned a single ATTACH texture; the texture parameter ATTACH itself will be ignored. |
| ► CYCLE | <i>Number;</i> | Current animation cycle of the texture. |

►OVERLAY

Overlay; Assigns an overlay keyword (see below) to the thing or actor which can then be accessed by actions, eg. for picking up the object with the mouse or representation within an inventory.

Example:

```
SET M. INVISIBLE, 1; // disappear  
SET MSPRITE, M. OVERLAY; // appear
```

These lines within an action, triggered by a texture **IF_CLICK**, will allow you to 'pick up' an object with the mouse pointer.

►TARGET

Keyword; Assigns a new destination or basic behaviour to the actor. Combining or event-triggered changing of these **TARGET**s may cause the Actor to show complex behaviour. The following target keywords may be assigned:

NULL

The actor does not move (default).

MOVE

The actor moves straight on in the direction of his **ANGLE**. His parameter **SPEED** (see below) allows you to define his horizontal speed. **VSPEED** defines the tangens of the vertical angle for the actor movement, e.g. with **VSPEED=1** the actor climbs upwards at an angle of 45°. If his **CAREFULLY** flag is set, his **IF_ARRIVED** action (see below) is triggered each time he crosses a region border.

BULLET

Like **MOVE**; but on hitting an obstacle the **IF_HIT** action of the actor is triggered. The flag **CAREFULLY** (see below) needs to be set to detect collisions.

STICK

The actor tries to keep his current distance from and angle to the player with maximum- **SPEED**. By changing the parameters **REL_ANGLE** and **REL_DIST** his position relative to the player can be changed.

If **CAREFULLY** is set, the skills **IMPACT_VX**, **IMPACT_VY**, and **IMPACT_VROT** (see below) are changed when the actor collides with walls, similar to player collisions. **STICK** actors may be used to give the player a three-dimensional 'structure', so that he may enter narrow corridors only sideways for example.

FOLLOW

The actor moves towards the player with **SPEED** and - if **VSPEED** is > 0 - tries to adjust his height to that of the player. The maximum vertical speed for height adjustment is determined by positive value of **VSPEED**. If **VSPEED** is negative, the actor will move upwards with **SPEED** at most, and downwards with **VSPEED**. If the **CAREFULLY** flag is set, the **IF_ARRIVED** action is triggered each time a region border is crossed.

REPEL

Like **FOLLOW**; but the actor moves away from the player, and his height does not change.

VERTEX

The actor moves towards the position corresponding to his target coordinates **TARGET_X** and **TARGET_Y** (see below). Upon arrival at this position the action **IF_ARRIVED** will be triggered.

HOLD

For polygonal (MDL) actors only: The actor will appear in front of the player, regardless of his position, tilt and angle. You can use this target to define 3D weapons or tools. The direction of the actor is defined by his **ANGLE**, his position in front of the player by his **HEIGHT**, **TARGET_X** and **TARGET_Y** parameters. **Example:**

```
MODEL sword_mod, <sword.mdl>;
TEXTURE sword_tex {
    SCALE_X    32;
    SCALE_Y    32;
    MODEL      sword_mod;
}
ACTOR sword_arm {
    TEXTURE     sword_tex;
    TARGET HOLD;
    ANGLE       3.14;
    TARGET_X    1;
    TARGET_Y    2;
    HEIGHT     -1;
}
```

Please note that this order of the parameters must be kept, and that there may only one **HOLD** actor be visible. You can define several **HOLD** actors, but the unused ones have to be set **INVISIBLE**.

NODE1
NODE2

The actor will be remote-controlled by the player plugged in via the number given in **NODE** (professional version only). Until communication with the node has been established the actor will remain invisible. If the number (1 or 2) *n* does correspond to the current node (command line option **-NODE**), the player will 'inherit' the position and perspective of this actor on game start.

Way

Stating a **WAY** keyword assigns this way to the actor. The actor immediately moves towards the first way point with **SPEED**. The keyword **WAYPOINT** is set to 1. **VSPEED** is ignored. On arrival at each waypoint the **IF_ARRIVED** action is triggered.

Wall, Thing Actor		Giving a keyword or synonym for a wall, a thing, or another actor assigns the first vertex or the position of the object as target to the actor, which he tries to reach horizontally and vertically with SPEED and VSPEED . If more than one object with the same keyword exists, the one with the lowest WED index number is chosen. When reaching the target the IF_ARRIVED action is triggered.
► WAYPOINT	<i>Number;</i>	Assigns the actor a waypoint on the way as destination. The actor moves directly towards this waypoint. During gameplay the number of the next directed waypoint can be evaluated with this keyword.
► TARGET_X ► TARGET_Y	<i>number;</i>	If an actor has WAY , FOLLOW , or an object as TARGET , these parameter contain the coordinates of his current target. With TARGET VERTEX the target coordinates may be given here.
► REL_ANGLE ► REL_DIST	<i>number;</i>	Angle and distance of the actor with TARGET STICK to the player; can be changed e.g. to move the actor around the player.
► HEIGHT	<i>Number;</i>	Height of the 'feet' of the thing or actor, either absolute (with GROUND flag set, see below) or relative to the floor level of its region. The perspective of objects with relative height sometimes seems more 'realistic' if they are shifted a little bit into the floor e.g. by HEIGHT = -0.25 .
► ANGLE	<i>Number;</i>	Object angle in radians (0..6.28). It is added to the angle given by WED and may be changed by an action.
► SPEED	<i>Number;</i>	Horizontal speed of the actor in steps per tick, default= 0.

► VSPEED	<i>Number;</i>	Vertical speed of the actor, dependent on its TARGET either as slope (tangens of the vertical angle) or as speed in steps per tick, default=0.
ASPEED	<i>Number;</i>	Through this parameter (default = 0 = disabled) his maximum rotation speed can be given in radians per tick. Especially multi-sided or MODEL actors behave more naturally on their WAYS with a rotation speed of around 0.2.
MAP_COLOR	<i>n;</i>	Color number (0..255, default 1) that represents the thing or actor on the map. If assigned 0 the object is not drawn, if 1 the object is drawn using a default color (skills COLOR_ACTORS , COLOR_THINGS - see below).
► DIST	<i>Number;</i>	Border distance from the object's center to the player, default=0. On crossing this border distance, an IF_NEAR or IF_FAR action may be triggered. If DIST =0, an IF_NEAR action can be triggered only by touching the object. Hint: If the DIST is > 0, but less then half the size of the object plus the width of the player, the player would have to penetrate the object to trigger an IF_NEAR action!
► SKILL1... ► SKILL8	<i>Number;</i>	Eight 'universal properties', which initially have no influence on the behaviour of the object. They can, however, be evaluated or changed by actions.

The following parameters are calculated by the program and can be evaluated by actions:

#DISTANCE		Approximate distance (+ /- 20%) of the player from the center of the object; only valid for objects within the player's CLIP_DIST .
#SIZE_X #SIZE_Y		Horizontal and vertical size of the object in steps; results from pixel size and scale of the texture.
►X, ►Y	<i>Number;</i>	Vertex position of the thing or actor. By changing this parameters actors may be 'beamed' to a new position during the game. He must also given his new region manually, and if his height above the floor is changed as a consequence, a SHAKE (see below) needs to be carried out.
►REGION	<i>Region;</i>	Region of thing or actor, may have to be re-assigned manually when changing positions. The region of things may not be changed as the collision detection would no longer work afterwards!

With the following keywords you can indicate actions which will be triggered through certain events associated with the the thing or actor. The actions will only be carried out if the object is within **CLIP_DIST** or has the flag **LIBER** (see below) set.

►IF_NEAR	<i>Action;</i>	This action will be triggered as soon as the player approaches the border distance of the thing or actor (DIST). If no border distance is defined, the action will be triggerd at every contact.
►IF_FAR	<i>Action;</i>	This action is triggered as soon as the player has moved away from the object beyond the border distance (DIST).

- | | | |
|---------------------|----------------|---|
| ► IF_ARRIVED | <i>Action;</i> | This action is triggered as soon as the actor with the CAREFULLY flag set crosses a region border or arrives at its target or at the next waypoint. |
| ► IF_HIT | <i>Action;</i> | This action is triggered if the object is hit with a SHOOT or EXPLODE instruction or if it collides with an obstacle (see BULLET , FRAGILE). |
| ► EACH_CYCLE | <i>Action;</i> | This action will be triggered after every animation cycle of the object's texture or at the end of a ONESHOT animation. |
| ► EACH_TICK | <i>Action;</i> | This action is triggered after every frame cycle (i.e. ever 1/16 sec approximately). |

With the following keyword a list of more keywords (flags) may be 'set', which determine the properties of the object. When evaluated in actionsflags either have the values 1 (set) or 0 (clear).

- | | | |
|--------------------------------|-------------------------|--|
| FLAGS | <i>Flag1, Flag2...;</i> | The following flags may be assigned (set) in this listing or evaluated or changed in actions: |
| #THING
#ACTOR | | The THING flag is always automatically set with things and actors, the ACTOR flag ist set with actors. Using these flags the type of an object synonym can be determined. |
| ► INVISIBLE | | If this flag is set, the thing or actor is invisible and passable, as if it was not there. The actor concerned is not moving, his actions aren't running and events are not triggered. On things or actors that are defined by WDL but not placed in the level, the INVISIBLE -flag is automatically set. |
| ► PASSABLE | | If this flag is set, the object is passable to the player. |
| #VISIBLE | | This flag is automatically set as long as a thing or actor is visible. It may be evaluated in actions. |

► **BERKELEY**

If this flag is set, the object is inactive (**INVISIBLE**) as long as it is not seen and the player is outside its **DIST**. It can be used to stop actor movement if he is out of sight. This flag also saves rendering time in levels with many animated things and actors.

► **LIBER**

If this flag is set, the object moves and executes actions even if it is outside the player's **CLIP_DIST**. Which of course takes up computing time... This flag is useful with objects sporting behaviour that is supposed to be independent from the player's movements, e.g. with guards patrolling, **EACH_TICK** time bombs or projectiles.

FAR

Actors that are invisible and farther away from the player than their **DIST** usually perform a **TARGET** movement (see skill **SKIP_FRAMES**) every 6th frame only. If this is not desired for a certain actor, his **FAR** may be set; then he'll move continuously regardless of **SKIP_FRAMES**.

► **GROUND**

If this flag is set, the **HEIGHT** of a thing or actor is not referred to the height of the region but to the Z coordinate. A thing or actor with **GROUND** set is vertically clipped ('cut off') at the heights of the floor and ceiling of its region, slopes not being taken into account. This way objects may e.g. submerge into and rise from water.

Things and actors without **GROUND** will always be displayed in full, even if parts of them penetrate into the floor or the ceiling. Still, walls and region's borders cause the penetrating parts to be clipped. If they are in a multi-storied region at game start they will be placed in the uppermost region.

Please note that if this flag is changed by an action, the **HEIGHT** will be automatically converted!

- CANDELABER** Only effective if **GROUND** is not set: the object will not be suited to the floor, but to the ceiling of its region. Useful for representing pendant objects like stalactites or candelabra.
- ▶ **SEEN** This flag is automatically set as soon as the object is seen by a player. It is for example evaluated by the automap function.
- ▶ **MOVED** This flag is set automatically if the actor has moved in the latest image cycle. It may be evaluated in actions and may then have to be reset manually.
- ▶ **PLAY** If this flag is set, the texture will be animated for one cycle and then stop on the last phase. On the texture concerned, the flag **ONESHOT** has to be set. At the end of the animation the flag **PLAY** is automatically set back to 0, and an **EACH_CYCLE** action may be triggered.
- ▶ **IMMATERIAL** After setting this flag the respective thing or actor will no longer be affected by **SHOOT** instructions or mouse clicks.
- ▶ **SENSITIVE** After setting this flag an **IF_HIT** action will be triggered with actors after they have collided with an obstacle, and the **IF_NEAR** actions will be triggered already when the actor is spotted by the player.
- ▶ **FRAGILE** If this flag is set, the **IF_HIT** action of a thing or actor may be triggered by an **EXPLODE** instruction.
- ▶ **CAREFULLY** If this flag is set, the actor performs collision detection on his way, and avoids walls, things and other actors. Also he detects a change of region, adjusting his height to the height of the region, if his flag **GROUND** is not set. This takes up computing time, however.

- MASTER** If an actor has the flags **MASTER** and **CAREFULLY** set, he will - as if he was the player - trigger **IF_LEAVE**, **IF_ENTER** and **IF_NEAR** events. Please note that **IF_NEAR** with a **DIST** > 0 won't work, he has to touch or pass the wall or object in order to trigger the event.
- SAVE** If this flag is set, all properties of the thing are saved at game saving. The **SAVE** flag is automatically set for actors.
- FLAG1...**
•**FLAG8** Eight 'universal flags', which initially have no influence on the properties of the object. They may be changed or evaluated by actions.

Hint: Since the bitmap-changing through **SIDES** on non-moving things or actors may look unnatural, only moving and animated actors should be multi-sided.

10. Actions

Actions can give the objects of the game world the pseudo-intelligent behaviour of state machines. There are 'global' actions, triggered at certain moments or by hitting a key, and actions corresponding to objects that will be triggered by object **events** like **IF_NEAR** or **IF_FAR**. An action consists of a list of instructions which will be executed one after another. **ACKNEX** is a multitasking engine; thus any number of actions execute simultaneously as in real life. The instructions represent a kind of a simple programming language, which allows you to influence the playing of the game in numerous ways.

ACTION *Keyword* { }

Assigns an action to the *keyword*. The following instructions can be given within an action:

SET [Object1.]*Keyword1*, [Object2.]*Keyword2*;

Perhaps the most common instruction of them all: **SET** assigns any keyword or object parameter a new value or another keyword. This way actions may change or disable themselves (by assigning the keyword **NULL**) even during the game. For *object1/2* the keyword or synonym (see below) of the region, the wall, actor or thing will be given, whose parameters are to be changed. The object must either be placed in the level or at least have its definition placed in the WDL file before the action.

Please note that if there is more than one object with this keyword placed within the level, the instruction only changes the first object placed via WED (cf. **SET_ALL**). For *keyword1* a **FLAG** keyword can also be assigned, which will be set by 1 or cleared by 0.

Examples:

```
SET tank.CEIL_TEX,water_tex; // Changes ceiling texture
SET bumper.FLAG3,1; // Sets FLAG3 of thing bumper
SET IF_S,NULL; // De-assigns the S key
```

SET_ALL *[Object1.]Keyword1, [Object2.]Keyword2;*

Like **SET**, except that **SET_ALL** affects all objects or regions with the keyword or synonym given via WDL, and not only the first. Positions (**X,Y,Z, X1,Y1,Z1, X2,Y2,Z2**) of objects may however not be changed with **SET_ALL**.

ACCEL *Skill, Number/Skill2;*

Adds the given number or skill2 to the first skill, but in a special way similar to an acceleration. The first skill is equivalent to a speed which is accelerated or decelerated by the value of the second parameter. Inertia and friction factors may be regulated with the predefined skills **INERTIA** (default 1) and **FRICTION** (default 0.5).

RULE *Skill / [Object.]Keyword = Ausdruck;*

Assigns the result of an arithmetic expression to the given skill or parameter. The arithmetic expression may contain numbers, further skills, object parameters, brackets, arithmetic operators and the following mathematical functions:

SIN(x), COS(x), TAN(x) ▶ Trigonometric functions

ASIN(x), ACOS(x) ▶ Inverse trigonometric functions

LOG(x), LOG10(x), LOG2(x) ▶ Logarithm of x

RANDOM(x) ▶ Random number between 0 and x

SQRT(x) ▶ Square root of x

SIGN(x) ▶ -1 if (x < 0), 1 if (x > 0), 0 if (x = 0)

ABS(x) ▶ Absolute amount of x

INT(x) ▶ Integer value of x

EXP(x) ▶ e power x

Additional to the basic operators + - * / , the following binary operators are supported (for integer values only):

% ▶ Modulo (integer remainder of a division)

| ▶ Bitwise OR

^ ▶ Bitwise Exclusive OR

& ▶ Bitwise AND

The operators can be combined with the "="-character:

+= ▶ adds the expression to the first parameter

-= ▶ subtracts the expression from the first parameter

***=** ▶ multiplies the first Parameter with the expression

/= ▶ divides the first Parameter by the expression

Because of the limited calculation accuracy (fixed point arithmetic with 3 places after decimal) no factors or intermediate values below 0.01 should be used. Please note that invalid operations - e.g. square root of a negative number - will cause an engine abort with error message!

Examples:

```
RULE x = (a + 1) * b / c;  
RULE z = x % 10;  
RULE actor.ANGLE = ASIN(3*x + 0.5);  
RULE x += 1; // increase skill x by 1  
RULE x += 2*TIME_CORR; // increase skill x time-corrected by 2
```

Please note that **RULE** can only assign numerical values. For all other parameters - flags, textures, actions and so on - use the **SET** or **SET_ALL** instructions.

IF (*expression*) { *instructions...* }

IF (*expression*) { *instructions...* } **ELSE** { *instructions...* }

will execute all instructions between the first pair of winged brackets only if the expression between the round brackets is true (i.e. evaluates to non-zero). Otherwise, the instructions between the second pair of winged brackets (after **ELSE**) will be executed. **ELSE** and the second set of instructions can be omitted.

The expression will mostly be a comparison between two sub-expressions, or a comparison of flags, textures or other parameters. Only parameters of the same type may be compared. For flag comparisons the values 1 (set) or 0 (not set) can be used. For comparisons of non-numerical parameters the value **NULL** for non-existence can be used. The following operators for comparison between two parameters or sub-expressions are available:

- || ▶ true if either is true (OR)
- && ▶ true if both are true (AND)
- != ▶ true if both are not equal
- == ▶ true if both are equal
- < = ▶ true if the first is below or equal to the second
- > = ▶ true if the first is above or equal to the second
- < ▶ true if the first is below the second
- > ▶ true if the first is above the second

All comparisons evaluate to a value of 0 for false, and 1 for true. Comparisons can be combined using brackets.

Examples:

```
IF (x<0) { // if x is below 0, then set y and z to -1
    RULE y=-1;
    RULE z=-1;
} ELSE {
    RULE y=1; // otherwise, set y and z to 1
    RULE z=1;
}

IF (((x+3)<9) || (y==0)) {
// set z to 10 if x+3 is below 9, or if y is equal to 0
    RULE z=10;
} ELSE {
// set z to 5 in all other cases
    RULE z=5;
}
```

Please note that the "equal" comparison is done with "=", not with "=" as one might expect! An **IF** instruction may appear within the winged brackets of another **IF** instruction; this way you can 'nest' several **IF** instructions.

WHILE (*expression*) { *instructions...* }

BREAK;

CONTINUE;

The **WHILE** instruction will repeat all instructions between the winged brackets as long as the expression between the round brackets is true resp. evaluates to non-zero. The expression will be evaluated at the begin of each repetition. **WHILE** is often used to modify a value slowly during a number of frames, e.g. to open a door or to move an elevator. For this a **WAIT** instruction (see below) has to be inserted between the winged brackets.

The instruction **BREAK**; between the winged brackets will end the repetitions and continue with the first instruction after the closed bracket. The instruction **CONTINUE**; will begin immediately with the next repetition.

As an **example** for an action using **WHILE** the floor of a region shall move up and down like an elevator between the heights 0 and 10, upon entering it:

```

REGION lift {
    ....
    IF_ENTER  lift_start;
}

ACTION lift_start {
    IF (lift.FLOOR_HGT>=10) {          // elevator already on top?
        WHILE (lift.FLOOR_HGT>0) { // then move down
            RULE  lift.FLOOR_HGT+=-0.2;
            WAIT  1;
        }
    } ELSE {
        WHILE (lift.FLOOR_HGT<10) { // move up
            RULE  lift.FLOOR_HGT+=0.2;
            WAIT  1;
        }
    }
}

```

GOTO *Label;*

Jumps to a target label in the action and proceeds from there with the subsequent instructions. *Label* may be any keyword followed by a colon as target mark anywhere between two instructions in the action.

Example:

```

    IF (energy<50) { // If skill energy is below 50...
        GOTO  alarm;
    }
    ...           // further instructions
alarm
    PLAY_SOUND alert, 0.3; // give acoustic warning
    FADE_PAL   red_pal, 0.5; // and optic warning

```

CALL *Action;*

Performs the given action immediately and then - either after the end of the **CALL**ed action or during a **WAIT** instruction (see below) in the new action - goes on with the next instruction of the current action.

END;

Terminates this action.

EXCLUSIVE;

Will terminate alle other actions previously triggered by the same wall, thing, actor, or region. This instruction can be used to prevent several actions of the same actor running simultaneously and disturbing each other.

WAIT *Number / Skill;*

Causes the action to stop for the given number of frame cycles. Instead of a fixed number, a skill can also be given. This instruction is often used within a **GOTO** or **WHILE** loop, during which a value has to change smoothly over a certain amount of time. During the **WAIT** time period all other actions keep running. If the action with the **WAIT** instruction was **CALL**ed from another action, this other action will also go on and perform its next instructions during the **WAIT** time.

EACH_TICK actions may not contain **WAIT** instructions, neither do actions directly (**CALL**) or indirectly (**IF_HIT**) triggered by **EACH_TICK** actions.

WAITT *Number / Skill;*

Like **WAIT**, but the action is halted for a given number of ticks. This instruction is able to wait for a fixed period of time.

PLAY_SOUND *Sound, Volume, [Balance];*

Starts a sound with the given volume. *Sound* is an previously defined keyword for a sound file, *volume* a number between 0 and 1, or a skill. With stereo soundcards you can shift the sound between the left and right channel by the optional parameter *Balance* (-1..+1). If *Balance* is not specified, both channels will be given the same volume.

Alternatively to a value for *Balance* the keyword of an object (wall, thing, actor) may be given. The sound will then emanate from the direction of the object within **SDIST** of its texture.

Example:

```
SOUND   crrahh, <crow.wav>;
TEXTURE  bird_tex  { BMAPS bird_map; SDIST 200; }
ACTOR   bird   { TEXTURE bird_tex; WAY bird_way; }
ACTION   bird_cry { PLAY_SOUND crrahh, 0.5, bird; }
```

PLAY_SOUNDFILE < *filename*> , *Volume*, [*Balance*];

Like **PLAY_SOUND**, but the sound is played directly from CD-ROM or hard disk without using up memory. Using an object instead of the *Balance* parameter is not possible here.

STOP_SOUND;

Aborts all sounds running.

PLAY_SONG *Music*, *Volume*;

Starts a new background song, which is repeated until another **PLAY_SONG** instruction. *Music* is a previously defined keyword for a midi file, *Volume* may be any value between 0 and 1. By playing a song with *Volume* 0 the music is switched off.

PLAY_SONG_ONCE *Music*, *Volume*;

Like **PLAY_SONG**, but plays the song only once.

PLAY_CD *Start*, *End*;

Plays the audio tracks of a CD. DOS 6.0 or higher and MSCDEX 2.2 or higher are required. The skills or numbers *Start* and *End* respectively indicate the first and the last track to be played (min 1, max. 99). If *End* is larger than the number of tracks on the CD, the CD will be played to its end. If *Start* is 0, the CD currently playing will be stopped; if *End* is 0, the CD will resume at the place it was stopped. If both are 0, nothing will happen.

After each execution of **PLAY_CD** the predefined skill **CD_TRACK** will assume the number of the track currently being played or 0, if the CD is not playing. Inbetween **CD_TRACK** won't change; to display the current track continously, a **PLAY_CD 0,0;** instruction has to be performed repeatedly, e.g. each second.

PLAY_FLIC *Flic*;

Starts a full-screen animation (professional version only). *Flic* is a keyword for a 320x200 FLI or FLC file with its own palette, defined beforehand via **FLIC**. During flic play the building of the screen and movements will stop, the palette of the animation file will be activated. The 16 'global' **EACH_TICK** actions (see below) keep running, and may be used to start sound effects or songs precisely at each desired frame number by comparing the predefined skill **FLIC_FRAME**.

PLAY_FLICFILE < *Filename*> ;

Like **PLAY_FLIC**; but reads the flic directly from the file. The memory claimed by the animation will be deallocated after playing. This instruction requires disk access. It is to be preferred if long animations are to be played without having to keep them in memory permanently.

STOP_FLIC;

Stops playing the current flic.

BEEP;

Plays a short sequence of notes on the PC speaker. Useful during level development, as it allows you to quickly find out that certain action was triggered.

INPORT *Skill, adr;*

The given *Skill* is set to the content of port #*adr* (professional version only). Through this instruction I/O ports can directly be accessed e.g. to control external devices or to implement new input devices.

Example:

```
SKILL   input {}
SKILL   my_port { VAL 372; }
INPORT  input, my_port;    // set input to the content of
                           // port #372
```

OUTPORT *Number/Skill, adr;*

The given number or *Skill* is sent to the port #*adr* (professional version only). Only values between 0 and 255 are possible.

FADE_PAL ***Palette, Number / Skill;***

Sets or changes the actual palette by fading. *Palette* is the keyword for a new palette definition, which is to be faded to from the current palette. The number or the skill gives the fade factor (value range from 0..1); by gradually increasing the factor soft fading is achieved. A fade factor of 1 or above makes the given palette the new current palette.

If the palettes differ not only in the shades of their colours, but also in the range of colours, a factor of or above 2 allows you to compute a new shading table. However this will take approx. 0.3 sec. if the flag **AUTORANGE** is set. With a factor of or above 3 a new **BLUR** table will also be computed if necessary (approx. 2 sec.).

Example:

```
PALETTE black_pal { PALFILE <BLACK.PCX>; }    // all colors just black
PALETTE level_pal { PALFILE <MPAL.PCX>; }      // define it AFTER black_pal!!
SKILL   fade {}
```

```

ACTION fade_out { // fades the picture softly to black
    SET      fade, 0;
    WHILE (fade<1) {
        RULE  fade += 0.1;
        FADE_PAL  black_pal, fade;
        WAIT    1;
    }
} // now everything is dark!

```

SETMIDI *Channel, status;*

GETMIDI *Channel, status;*

Set or get the device type for a midi channel. *Channel* must be in the range 0..15. For *status* the following midi devices are available: 0 = FM mode (default), 1 = digital mode (DDK option only) and 2 = external mode (Midi output via Wave port resp. Midi port of the sound card).

MIDI_COM *Statusb, data1, data2;*

Send data directly to a Midi channel. The skill statusb is composed of the sum of (Midi command*16) plus channel number (0..15).

Example:

```

SKILL stbef { VAL 11; } // command
SKILL statusb { VAL 0; } // status byte
SKILL channel { VAL 0; } // channel
SKILL data1 { VAL 123; } // 1. data byte
SKILL data2 { VAL 0; } // 2. data byte

```

```

ACTION PlayNoteTest {
    SET      stbef, 9; // command
    SET      data2, 100; // volume
    WHILE (data1<200) {
        RULE  statusb = 16*stbef+channel;
        MIDI_COM statusb, data1, data2;
        RULE  data1 += 5;
        WAIT  1;
    }
}

```

Please note that you may not send too fast to an external midi channel, because the midi port (only 31 kBit/sec) is not buffered.

INKEY *String;*

Copies the keyboard input into the string with the name or synonym given. The local keyboard layout will be activated automatically. The instruction then waits for the termination of the input via **[Return]** and only then proceeds with the action, similar to **WAIT**. However, no **SAVE/LOAD** instructions are executed during an **INKEY** input. **[Esc]**, **[Up]**, **[Down]**, **[PgUp]**, or **[PgDn]** allows you to abort the input at any time. The previous content of the string will be restored. The text of the input can be edited using **[BackSpace]**, **[Del]**, **[Cursor right]** and **[Cursor left]**. The keyboard layout corresponds to the respective country-specific layout.

If the string appears in a **TEXT** displayed on screen (see below), the input as well as the cursor (flashing character 127 from the corresponding **FONT**) are visible. If the end of the string is reached (results from the length of the initial string of the **STRING** definition), no further keyboard input apart from **[Return]** and **[Esc]** will be accepted. Trailing spaces will be cut off. After the termination of the input the skill **RESULT** will be at -1 if the input was aborted with **[Esc]**, 72 with **[Up]**, 73 with **[PgUp]**, 80 with **[Down]** or 81 with **[PgDn]**, else it will be 0. The string length can be evaluated by the predefined skill **STR_LEN**.

SET_STRING *String1, String2;*

Copies the content of the 2nd string into the 1st. The resulting length of the 1st string won't exceed its original length.

ADD_STRING *String1, String2;*

Like **SET_STRING**, but the 2nd string will be concatenated to the first. Please note that the original length of the first string may not be exceeded!

TO_STRING *String, Skill;*

Will copy the digit representation of the given skill into the given string.

SET_SKILL *Skill, String;*

If the string contains a number, the value of the given skill is set to its numeric representation.

FIND *Text,String;*

Sets the **OFFSET_Y** and **INDEX** parameters of the given **TEXT** to one of its strings whose first characters match the content of the given *string*. The skill **RESULT** is also set to the relevant **INDEX**. If no correspondence is found within the **TEXT**'s strings, **RESULT** will be set to -1, and **OFFSET_Y** and **INDEX** remain unchanged. After the **FIND** instruction the predefined skills **LINES** and **SIZE_Y** contain the numbers of character and pixel lines of the **TEXT**.

SET_INFO *String,Object;*

This instruction is the built-in debugger. It will write all interesting parameters of the given object into the string, which can be displayed onscreen. The string should have at least 1000 characters for this purpose. This instruction is only available in WED, not in the runtime module.

Example for a debugging WDL which freezes the game by pressing the right mouse button, let a mouse cursor appear, then shows all informations about any object clicked on with the left mouse button:

```

////////////////////////////////////
// At first the code to switch on and move the mouse:
OVLV      mouse_ovr, <arrow.pcx>;
OVERLAY   mouse_sprite { OVLVS mouse_ovr; }
MSPRITE   mouse_sprite;

ACTION mouse_toggle {           // switches the mouse on and off
    RULE MOUSE_MODE += 2;
    IF (MOUSE_MODE > 2) {       // was it already on?
        RULE MOUSE_MODE=0;
        END;
    }
    WHILE (MOUSE_MODE > 0) {
        WAIT 1;                 // now move it over the screen
        RULE MOUSE_X += 2*MCKEY_X;
        RULE MOUSE_Y += 2*MCKEY_Y;
    }
};
////////////////////////////////////
// Now the WDL code for showing the debugging info:
STRING info_strg, "
.....
.....

```

```
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
....."; // just a very  
  
long string  
  
TEXT    info_txt {  
    POS_X      4;  
    POS_Y      4;  
    FONT       debug_font;  
    STRING info_strg;  
}  
  
SYNONYM check_obj { TYPE ACTOR; }  
SKILL running { VAL 0; }  
  
ACTION scan_obj { // permanently update the object info  
    IF (running != 0) {  
        END;  
    }           // do nothing if action is already running  
    RULE   running=1; // (remember, this is multitasking!)  
    WHILE (running) {  
        SET_INFO    info_strg,check_obj;  
        WAIT         1;  
    }  
}  
  
ACTION obj_click {  
    BEEP;  
    SET check_obj, TOUCHED;  
    SET info_txt.VISIBLE, 1;  
    CALL  scan_obj;  
}  
  
ACTION freeze_unfreeze {  
    IF (MOUSE_MODE == 0) { // if the mouse is about to appear  
        SET MOVE_MODE, 0.5; // freeze anything
```

```
} ELSE {  
  SET   MOVE_MODE, 1;  
}  
CALL   mouse_toggle;  
}  
  
IF_RIGHT  freeze_unfreeze;  
IF_KLICK  obj_click;
```

Please note that you must not use the mouse actions anywhere else in your WDL files in order for this to work.

PRINTFILE *"Name",Zahl/Skill;*

Gives the name of a text file, into wich skill and string values can be written. The name is composed (like with **SAVE**) from the given string (max. 5 characters) plus a 3-digits number plus extension .TXT. The default name is "PRINT0.TXT".

PRINT_VALUE *Skill;*

Writes the given skill as number with 3 decimals into the text file given by **PRINTFILE**. If the file already exists, its original content is not written over, the number is appended at the end.

PRINT_STRING *String;*

Like **PRINT_VALUE**, but the string content - which can also be given directly between double quotation marks - is written into the file. The characters "\n" within the string create a line-feed within the file.

FREEZE *Bmap, Number/Skill;*

Copies the current 3D image into an existing bitmap, reduced and without panels and overlays. *Bmap* is a keyword that was assigned the bitmap. *Number/Skill* specifies the factor of reduction (1..16). The size of the image depends on the current **MOTION_BLUR**, so better set it temporarily to 0 before **FREEZING**. The bitmap may then be displayed as a **PICTURE** in a panel, to indicate a saved game. The modified bitmap is saved with the instruction **SAVE_INFO**; this may only be done one frame cycle after the **FREEZE** instruction at the earliest (insert a **WAIT** instruction!).

SCREENSHOT *"Name", Number/Skill;*

Grabs the content of the screen and saves it as PCX file with the given *Name* (max. 5 characters, in parantheses) plus a 3-digit number (*Skill*) and extension "PCX". The size of the picture depends on the current **MOTION_BLUR**, so better set it to zero one frame cycle before taking the **SCREENSHOT**.

NEXT_MY;**NEXT_THERE;****NEXT_MY_THERE;**

These instructions change synonyms (see below) for modifying several objects within a loop. The **NEXT_MY** instruction will place the **MY** synonym onto the next object with the same name. If the last object is reached, it will begin again with the first one. **NEXT_THERE** will the same way place the **THERE** synonym onto the next region with the same name. **NEXT_MY_THERE** will place the **MY** synonym onto the next object in the same region.

Example to lift all *lift1* regions by 1 step:

```
SET THERE, lift1;    // set synonym THERE to first region lift1
```

```
loop1:
```

```
  RULE    THERE. FLOOR_HGT += 1;
```

```
  NEXT_THERE;                // set it to next region
```

```
  IF (THERE!=lift1) { GOTO loop1; }
```

LOCATE [*Object*];

Through this instruction the lost region of the player, of a thing or an actor is found and re-assigned. This works even if by 'beaming' or moving an actor without **CAREFULLY** flag his original region got lost. **LOCATE** with an object parameter restores the region of the thing or actor concerned, **LOCATE** without parameter restores the actual player region (**HERE** synonym).

DROP *Object;*

The given thing or actor is placed in the line of sight of the player (**PLAYER_ANGLE**) at the distance of its **DIST** parameter. If the mouse pointer is active, the object is placed in the direction of the pointer instead (**MOUSE_ANGLE**). If this is not desired, the **MOUSE_ANGLE** has to be set to the **PLAYER_ANGLE** before the instruction. For collision detection to work, the new position of the object must lie in the same region as that of the player; so it's recommended to **DROP** it at a **DIST** of 0.

SHOOT [*Object*];

A most important instruction for action games. If the optional parameter *object* is omitted, an **IF_HIT** action is triggered with the nearest object - wall, thing or actor - in the player's line of sight. By the predefined skill **SHOOT_FAC** the 'gun-power', by **SHOOT_RANGE** the maximum range, by **SHOOT_X** and **SHOOT_Y** the deviation from the center of the 3D window is set (range of value from -1..+1). After the **SHOOT** instruction the skill **HIT_DIST** is given the distance of the object hit and the skill **RESULT** contains the power of the hit: **RESULT** = **SHOOT_FAC** * (1 - **HIT_DIST** / **SHOOT_RANGE**). The skills **HIT_X** and **HIT_Y** give the pixel coordinates of the texture pixel hit, relative to the upper left corner of the texture, so that an **ATTACH** bullet hole texture can be set there. If no object is hit, both skills are set to 0.

Only objects with no **IMMATERIAL** flag set will be 'hit' by this instruction. The transparent parts of a thing or actor texture will not be hit by a **SHOOT** instruction. **SHOOT** may be used for shots but also for activating nearby switches or to open doors.

If the keyword or synonym of a thing or actor is given for *object*, this instruction may establish whether the player is visible from the object. In this case the skills **HIT_DIST** and **RESULT** are set according to the distance of player and object; otherwise **HIT_DIST** and **RESULT** are set to 0. After the instruction the predefined skill **SHOOT_ANGLE** gives the angle from the object to the player, which may then be given to a projectile or the actor itself (**SET actor.ANGLE, SHOOT_ANGLE**);).

The skill **SHOOT_SECTOR** (default = 2ð) may be used to adjust the range of angle the actor uses when carrying out the **SHOOT** instructions. The

ANGLE of the the actor is at the centre of this range. If the player is outside this sector, **SHOOT** will merely return a **HIT_DIST** and **RESULT** of 0. This way you may get a monster to see the player only from the front while 'sneaking' on it from behind.

EXPLODE *Object;*

Triggers the **IF_HIT** actions on all walls, things or actors which had set **FRAGILE** flags and whose nearest edge is within the distance **SHOOT_RANGE** from the position of the actor with name or synonym *Object*. The **RESULT** skill is set depending on power of the hit - similar to **SHOOT** - to a value between 0 and 1.

After the instruction the skills **HIT_DIST** and **RESULT** are set as with **SHOOT**, should the player himself be hit. The skill **HIT_MINDIST** contains either the distance to the nearest object hit or 0 if no object with **IF_HIT**-action was within the critical distance.

If the **IF_HIT** action of the object hit itself contains an **EXPLODE** instruction, a wait (**WAIT1**;) should be inserted before the action, in order to avoid that two adjoining objects block each other's **EXPLODE** engines.

PUSH *Number/Skill;*

Triggers the **IF_HIT** action with the closest visible object (wall, thing, or actor) within the distance specified by *number* or *skill*. Can be used to open doors or start elevators. The skills **HIT_DIST** (either 0 or distance) and **RESULT** are set like with **SHOOT**.

SHAKE *Object;*

Tells the object (wall, thing, actor) that its position was changed. A **SHAKE** instruction is required after each direct setting of coordinates if the floor height of the object of the length of a wall was changed as a consequence.

Example:

```
RULE    M. X1 += 0.5;  
SHAKE   M;
```

LIFT ***Region, Number/Skill;***

Increases the z values of all vertices, things and actors belonging to the region *region* by the given amount or skill. You can use this instruction to accomodate the vertical position of things or actors to lifts moving up or down.

Please keep in mind that the flag **SAVE** must be set with the walls and objects of the corresponding regions, if the change of height is to be saved when saving the score. The same is true for the following region-changing instructions:

TILT ***Region, Number/Skill;***

Multiplies the z values of all vertices, things and actors inside the region by the given amount or skill. You can use this instruction to change the slope of a region during gameplay.

SHIFT *Region, dx, dy;*

Shifts all vertices, things and actors belonging to the region *region* in the direction of *x* or *y* on the system of coordinates by the amount or skill given. This instruction for example allows you to let vehicles or rafts move. Still, no wall of the region may neither touch nor cross any other wall of the region while moving!

ROTATE *Region, Number/Skill;*

Rotates all walls and objects of a region by the angel given. The centre of rotation is the region parameter **GENIUS**. The **GENIUS** may also be situated outside the region and be assigned to more than one region at the same time, causing them to rotate around a common centre. No wall of the region may neither touch nor cross any other wall of the region while rotating!

Example:

```

ACTOR wheel_center { TEXTURE any_tex; FLAGS INVISIBLE; }
REGION wheel {
    FLOOR_HGT 2;
    CEIL_HGT 20;
    FLOOR_TEX wood_tex;
    CEIL_TEX stone_tex;
    GENIUS wheel_center;
    EACH_TICK wheel_rotate;
    FLAGS SAVE;           // to keep any changes
}
ACTION wheel_rotate {
    ROTATE wheel, 0.05;    // 0.05 Rad = ca. 3 Degrees
    RULE wheel.FLOOR_ANGLE += 0.05; // rotate the texture also
}

```

SAVE *"Name", Number/Skill;*

Saves the game under a specifiable name plus a number in the game directory (see **SAVEDIR**). The name of the saved file is composed of the first five letters of the string "*Name*" plus a three-place number that represents the value of *skill* plus the extension .SAV.

LOAD ***"Name", Number/Skill;***

Loads, decompresses and decodes the score, which was saved under the name *Name* plus a number of three digits maximum that represents the value of *skill*. If the predefined skill **LOAD_MODE** is set to 1, the screen resolution will not be changed.

SAVE_INFO ***"Name", Number/Skill;***

Like **SAVE**; but saves only the **LOCAL** and **GLOBAL** skills, all bitmaps changed by **FREEZE** and all strings changed by **INKEY** in the game file.

LOAD_INFO ***"Name", Number/Skill;***

Like **LOAD**; but loads only those values saved with **SAVE_INFO**.

SAVE_DEMO ***"Name", Number/Skill;***

Records the movement of the player and actors and all other game events to a demo file in the game directory. The name of the saved file is composed of the first five letters of the string *"Name"* plus a three-place number that represents the value of *skill* plus the extension .SAV.

PLAY_DEMO *"Name", Number/Skill;*

Replays the demo file. Pressing of any key stops the replaying.

STOP_DEMO;

Stops recording or replaying of the demo file.

MAP < *Filename*> ;

Level change; loads a new level topography from the file < *filename*[.WMP]> and sets the player and all actors to the their new starting positions.

LEVEL < *Filename*> [, "*Resourcename*";

Complete level change (professional version only); loads a new world definition from the script < *filename*[.WDL]> , loads the topography given in **MAPFILE** and performs the new **IF_START** action. When the WDL file is contained in a new WRS-resource, it may also be specified. Please make sure not to include the resource name in brackets, as it would then be compiled into the level resource! If the predefined skill **LOAD_MODE** is set to 1, the screen resolution will not be changed. All regions, objects and skills - except the **GLOBAL** skills, see below - are replaced by the elements of the new level.

EXIT [*"Text"/String*];

Ends the game and returns to DOS. The optional text, which can be given either directly or as a previously defined string keyword, is issued on the DOS screen after the end of the game.

Besides the actions triggered by object events, the following keywords may be used to specify standard actions:

►EACH_TICK Action,Action...;

This list of up to 16 actions is performed after every frame cycle, i.e. approx. every 1/16 second. The actions are executed consecutively in the order given. With the instruction **SET EACH_TICK.n,Action** ($n = 1..16$) actions on whichever position on the list may be changed; **SET EACH_TICK.n,NULL** deletes the action concerned from the listing. **EACH_TICK** actions may not contain **WAIT** instructions nor may they execute **CALL** or **BRANCH** instructions referring to actions containing **WAIT**s.

The event **EACH_TICK** is especially suited to assign certain positions on the list specific actions with a fixed meaning that may then be changed or removed precisely. **EACH_TICK.16** could be used for the movement of the player.

IF_START Action;

The action specified, which may contain change of palette, title animation, songs or other, is performed at the start of the game.

►IF_LOAD

The action specified will be executed after loading of a score that was previously saved. It may be used to reset the panels or skills (**MOVE_MODE** etc.) used for saving to normal.

►IF_LEFT Action;

The given action is performed when the left mouse button or the joystick button 1 was pressed.

►IF_MIDDLE Action;

The given action is performed when the middle mouse button or the joystick button 3 was pressed.

►IF_RIGHT Action;

The given action is performed when the right mouse button or joystick button 2 was pressed.

►IF_KLICK Action;

The given action is performed when clicking left with the mouse pointer somewhere within the 3-D window, without hitting any object or panel.

►IF_MSTOP Action;

This action is triggered when the mouse pointer is active and the mouse was held stationary for ½ second. The skill **MOUSE_CALM** may be used to specify a maximum distance in mouse-pixels that is still considered to represent immobility (default 3). By evaluating the skill **MOUSE_MOVING** you may also find out, whether the mouse is held stationary (0) or is moving(1).

►IF_ANYKEY Action;

The action specified is performed when any key is pressed.

►IF_F1... Action;

The action specified is performed when the [F1] key is pressed. The following keys may be assigned actions by the same way: **IF_F2...IF_F12**, **IF_ESC**, **IF_TAB**, **IF_CTRL**, **IF_ALT**, **IF_SPACE**, **IF_BKSP**, **IF_CUU**, **IF_CUD**, **IF_CUR**, **IF_CUL**, **IF_PGUP**, **IF_PGDN**, **IF_HOME**, **IF_END**, **IF_INS**, **IF_DEL**, **IF_PAUSE**, **IF_CAR** (Full stop), **IF_CAL** (Comma), **IF_ENTER**, **IF_0...IF_9**, **IF_A...IF_Z**. Key actions may be redefined with **SET** instructions (e.g. **SET IF_F1, action;**) during gameplay.

11. Synonyms

Frequently one action is supposed to change multiple objects of regions. In order to avoid writing an action for each of these objects you may use a **Synonym** instead of the keyword indicating the object or region. Some synonyms are predefined(**MY**, **HERE**, **THERE**); any number of further synonyms may be defined by the user:

SYNONYM *Keyword* { ... }

Such a synonym definition may contain the following keywords - in the sequence given:

TYPE *Keyword*; Refers to the type of synonym. *Keyword* may either stand for **OVERLAY**, **TEXTURE**, **WALL**, **THING**, **ACTOR**, **REGION**, **PANEL**, **TEXT**, **STRING**, or **ACTION**.

DEFAULT *Keyword*; Optional: object the synonym stands for at gamestart.

You can use synonyms like normal keywords. Action synonyms can be started by **CALL** instructions. Synonyms can be assigned each other:

```
REGION lift_1 { ... }
SYNONYM lift_syn1 { TYPE REGION; }
SYNONYM lift_syn2 { TYPE REGION; }
SYNONYM lift_floor { TYPE TEXTURE; }

ACTION lift1_to_lift2 {
    SET lift_syn1, lift_1;
    SET lift_syn2, lift_syn1;
    SET lift_floor, lift_syn2.FLOOR_TEX;
}
```

The following synonyms are predefined:

- **HERE** This synonym stands for the region the player is in; important for the collision detection and height adjustment. If the player changes its region by direct displacement of the player position (skills **PLAY-ER_X, PLAYER_Y**) **HERE** may have to be manually reassigned.
- **THERE** Synonym either for the region that has triggered the current action (by **IF_ENTER, EACH_TICK**, etc), or the region of the object (**MY**) which has triggered the action.
- **MY** Synonym for the object that has triggered the current action (by **IF_NEAR, IF_FAR, EACH_TICK** etc.). This allows to directly adress singular objects without knowing their index (e.g. with **SET MY.TEXTURE, crash_text**). If the current action is not triggered by an object event, this synonym is undefined.
- #**HIT** Synonym for the object last hit with the **SHOOT** instruction by the player, or object last clicked with the mouse, or the object hit by the last **EXPLODE** instruction which is closest to the center of the explosion.
- #**TOUCHED** Object last touched by the mouse.
- #**TOUCH_TEX** Texture last touched by the mouse.
- #**TOUCH_REG** Region, whose floor or ceiling was last touched by the mouse.
- **TOUCH_TEXT** By assigning a **TEXT** to this predefined synonym all **TOUCH** texts will appear with the font and flags of the text concerned. If the **VISIBLE** flag of the text is set, the **TOUCH** texts will appear at its **POS_X/POS_Y** instead of at the mouse position.

12. Skills & role playing games

'Role Playing Games' is a traditional expression for games in which players and actors have individual properties (skills) like in real life. The nature and the effects of these properties can be defined freely. Skills are like variables in a programming language. There is a number of predefined skills which may for example determine the movemental behaviour of the player. Additionally any number of skills can be defined - state of health, combat prowess, magic ability, blood pressure, level of alcohol in blood and so on... all this is left to the author's imagination.

The skills are defined as follows:

SKILL *Keyword* { ... }

Such a skill definition may contain - in the given order - the following keywords:

- TYPE** *Keyword*; This is related to the type of skill. *Keyword* stands for **LOCAL** or **GLOBAL** or **PLAYER** (default). There are only differences concerning the behaviour of the skills during saving and loading: **GLOBAL** skills are saved with the instructions **SAVE** and **SAVE_INFO**, **LOCAL** skills only with **SAVE_INFO**, **PLAYER** skills only with **SAVE**.
- **VAL** *Number*; Fixed point number, starting value of a skill. If no value is given, the skill value is set to 0.
 - **MIN** *Number*; Fixed point number, lower margin of the skill. The skill value never drops below this margin.
 - **MAX** *Number*; Fixed point number, upper margin of the skill. The skill value never exceeds this margin.

GLOBAL skills are especially suited to all values that are supposed to be kept for each level of the game, like prowess, score, or severity. After starting the game or changing levels (instruction **LEVEL**) they may be reloaded from the last game score saved using a **LOAD_INFO** instruction.

LOCAL skills always keep the last value when loading a game score. They are suited for user-settings like volume or screen resolution that are not supposed to change when loading an old score.

Once the skills are defined, their values may be changed, like with any other keyword, by actions with **SET**-, **RULE**- or other instructions.

Example: The player has a property called HEALTH. This health should normally decrease continuously. If the player is hit by an evil actor, the HEALTH quality suddenly drops by a fixed amount, and then returns slowly to the previous value later. All this may be defined as follows:

```
SKILL health      { VAL 100; }
SKILL recover     { VAL 0; }
SKILL rec_factor  { VAL 0.5; }

ACTION permanent {
    WAITT 16; // wait for one second
    RULE health += rec_factor * recover - 0.005;
    RULE recover -= 1; // Decrease value down to 0
}

ACTION ouch {
    RULE health -= 25; // Hit - Health drops
    RULE recover = 10; // For 10 seconds time to recover
}

EACH_TICK permanent;
```

Please note that the values are choosen to allow the player a recovery time of exactly 10 seconds after each hit - if I have calculated right...

Besides the user defined skills, the following predefined skills are available, which may be evaluated in actions and - if marked with the triangle ► - may also be changed:

►**SCREEN_WIDTH**
►**SCREEN_HGT**

The width and height of the 3-D window in pixels. The default and maximum values are the result of the screen resolution set via the **VIDEO**. You can change the window size during gameplay; please note that the width must be divisible by 16, the height by 2.

By redefining these skills the maximum window size can be set at game start to save memory. These maximum values may then never be exceeded during gameplay.

►**SCREEN_X**
►**SCREEN_Y**

Horizontal and vertical distance of the 3-D window from the upper left corner of the screen in pixels; default = 0. The maximum distance results from the screen resolution set by **VIDEO** minus **SCREEN_WIDTH** and **SCREEN_HGT** respectively. The horizontal distance must be divisible by 4.

►**ASPECT**

Height-to-width ratio of the rendered scene within the 3-D window, range of value from 0.1..10. The default value of 1 sets a ratio of 1:1.

►**EYE_DIST**

Eye distance for 3-D glasses (default 0.5 steps). The command line option **-3D_SIMEYE** must be given. If this skill is set to 0, the stereoscopic effect is disabled. With stereoscopic mode on the effective frame rate halves.

►**SKY_OFFSET_X**
►**SKY_OFFSET_Y**

Horizontal and vertical shifting of all sky textures in pixels. **SKY_OFFSET_X** must never become negative. By periodically increasing **SKY_OFFSET_X** through an action you may achieve a 'wandering cloud' effect.

►MOTION_BLUR

Parameter for the motion blur effect (0..1). The motion blur effect automatically reduces the screen resolution during player movement; causing the movement to appear smoother on slower machines. The default value is 0 (no motion blur effect).

►BLUR_MODE

By setting this skill (default 0) to 1 the motion blur is permanently activated, even if the player doesn't move. By setting **BLUR_MODE** to 0.5 the blurring is activated if either the player or the mouse moves.

►RENDER_MODE

Rendering activity within the 3-D window. The following values are possible:

- 2: Complete rendering; necessary after changing the screen size or after direct displacement of the player position.
- 1: Partial rendering; necessary after direct displacement of objects or walls or the floor or ceiling level of a region.
- 0.5: Default setting. The image is rendered according to the movements of the player.
- 0: Rendering is suppressed (to show titles, credits, or full-screen menus).

If **RENDER_MODE** is returned to 0.5 automatically after each frame cycle.

►MOVE_MODE

Movement of players and actors, default=1. With this skill, actions and movement in the level may be 'frozen'. The following gradation is possible:

1: Default.

0.5: The movements of actors and object **EACH_TICK** actions are stopped.

0: Also the movement of the player and all object and region events won't be carried out any more.

-0.5: All actions, except the keyboard events, are stopped.

►CLIPPING

Controls the suppression of objects outside the **CLIP_DIST**. The following gradation is possible:

0: Default; walls are not visible if one of their vertices is outside the **CLIP_DIST**.

0.5: Walls are not visible if both of their vertices are outside the **CLIP_DIST**, and their flag **FAR** is not set. If you have a large outside area, use flag **FAR** for the surrounding **SKY** walls.

1: Walls are not visible if either both vertices are outside the **CLIP_DIST**, or a wall of their region is not visible. Walls with flag **FAR** are visible in each case. Use this with caution - it only works with your level split into small regions.

►LOAD_MODE

If this skill (default 0) is set to 1, the screen resolution won't be changed by a **LOAD** or **LEVEL** instruction.

- **THING_DIST**
► **ACTOR_DIST**
- Ratio of the thing's resp. actor's, and wall's **CLIP_DIST**. **THING_DIST** must be between 0 and 1 (default), while **ACTOR_DIST** must be equal or below **THING_DIST**.
- **MAP_OFFSX**
► **MAP_OFFSY**
- Horizontal and vertical deviation of the centre of the map from the centre of the 3D window in pixels; default 0.
- **MAP_CENTERX**
► **MAP_CENTERY**
- Horizontal and vertical deviation of the center of the map like **MAP_OFFSX** and **MAP_OFFSY**, however not in pixels but in steps. This has the advantage that independently from the scale of the map (**MAP_SCALE**) the center remains always at the same position.
- #**MAP_MAXX**
#**MAP_MINX**
#**MAP_MAXY**
#**MAP_MINY**
- Maximum and minimum X and Y coordinates of all the level's objects; is automatically calculated at game start.
- **MAP_EDGE_X1**
► **MAP_EDGE_X2**
► **MAP_EDGE_Y1**
► **MAP_EDGE_Y2**
- Left and right limits of the map display on screen, given as distance in pixels from the upper left edge of the screen. Defaults correspond to the current size of the screen.
- **MAP_SCALE**
- Scale of the automap relative to the size of the 3-D window; default=0.9. On a value of 1 the automap fits exactly in the window.
- **MAP_MODE**
- Map display mode; range of values from 0..1, default 0. On values > 0 the map of all regions and objects previously explored and seen is shown in the 3-D-window. On values of > = 1 all objects - including objects not yet seen - are shown.

- ▶ **MAP_LAYER** Number of the overlay layer (see **LAYERS**), the map appears upon. Range of value from 0..16, default 0. All overlays on higher layers are drawn over the map.

- ▶ **MAP_ROT** If this skill is set to 1, the map will rotate along with the player angle around the player symbol, similar to a radar display. However, only objects within **CLIP_DIST** will be shown.

- ▶ **COLOR_PLAYER** Color number for the player symbol on the automap (default=7).

- ▶ **COLOR_ACTORS** Default color number for actor symbols on the automap (default = 3).

- ▶ **COLOR_THINGS** Default color number for thing symbols on the automap (default = 13).

- ▶ **COLOR_WALLS** Default color for walls on the automap (Default = 244). If the color lies within a shading range, the walls are drawn with 'antialiasing'.

- ▶ **COLOR_BORDER** Default color for border walls on the automap (Default = 244). If the color lies within a shading range, the borders are drawn with 'antialiasing'.

- ▶ **MOUSE_MODE** Mouse representation. At a setting of 0 (default) the mouse pointer is not shown on the screen; if set at 1 the pointer (**MSPRITE**, see below) appears. If set at 2 mouse movements don't change the **FORCE_...** skills anymore, allowing the mouse to be moved independently from the player.

► TOUCH_MODE	If set to 1 (default) the texture strings (TOUCH) of touched objects are shown at the mouse pointer position.
#MOUSE_-MOVING	Mouse is moving (1) or has been immobile for 1/4 second (0).
► MOUSE_CALM	Maximum distance still recognized as immobility for MOUSE_MOVING (default 3)
► MOUSE_TIME	Time in ticks that is used to measure the distance MOUSE_CALM in order to determine MOUSE_MOVING ; default 4.
#MICKEY_X #MICKEY_Y	Movement of the mouse in dots since the last frame; can be used to set the skills MOUSE_X and MOUSE_Y .
► MOUSE_X ► MOUSE_Y	Horizontal and vertical position of the mouse pointer in pixels, relative to the upper left screen corner. If the mouse pointer is switched on through MOUSE_MODE , it may be moved over the screen by changing these skills. Example: RULE MOUSE_X += MICKEY_X; RULE MOUSE_Y += MICKEY_Y; The MIN and MAX values should be adjusted in such a way as to assure that the mouse pointer cannot move beyond the edge of the screen.
#MOUSE_ANGLE	Directional angle of the mouse pointer in the level, e.g. for throwing away an object at the mouse pointer (DROP); is automatically calculated from MOUSE_X and the PLAYER_ANGLE . May be changed directly to adjust the DROP angle.

► TOUCH_DIST	Maximum distance of an object to trigger the texture mouse events TOUCH , IF_TOUCH , IF_RELEASE , IF_KLICK . Default = 100 steps.
#TOUCH_STATE	This skill (default 0) takes on the value of 1 when the mouse pointer is on an object with TOUCH text, 2 if an IF_CLICK action was assigned to the object texture, and 3 if both are the case.
#JOYSTICK_X #JOYSTICK_Y	Movements of the joystick axes in the range -255...+ 255.
#STR_LEN	Number of characters of a STRING which was entered through the INKEY intruction.
#LINES	Number of the character lines of a TEXT ; will be set through the FIND instruction.
#SIZE_Y	Vertical pixel size of a TEXT ; will be set through the FIND instruction.
► SOUND_VOL	Volume for sound effects, range from 0..1; is to be multiplied by the individual volume of each sound.
► MUSIC_VOL	Volume for music, range from 0..1; is to be multiplied by the individual volume of each song. If this value is set to 0, the music is switched off.
► CDAUDIO_VOL	Volume of the currently playing CD audio track, range between 0 and 1. Please note that this works only under plain DOS. Under Windows 95 the CD audio volume is controlled by the mixer icon.

- #CHANNEL** Number of the sound channel (0..7), the last sound was assigned to for example by a **PLAY_SOUND** instruction. If the sound is not played for one reason or other (e.g. if all channels are busy with **SLOOP** sounds), **CHANNEL** acquire a value of -1
- #CHANNEL_0..**
#CHANNEL_7 These skills allow you to continuously query for the state of the 8 sound channels:
0 = No sound, channel free
1 = Normal sound is playing
2 = **SLOOP** sound playing.
- AMBIENT** Global light intensity (range -1..+1, default 0) that is added to all texture and region **AMBIENTs**.
- PLAYER_LIGHT** Power of a source of light the player carries with him (0..1, 16 grades, default 1).
- LIGHT_DIST** Distance the between player and the beginning of a shading area (default 10 steps). Objects closer than **LIGHT_DIST** will no longer be affected by **PLAYER_LIGHT**.
- #DARK_DIST** Distance of the area of darkness from the player, is automatically recalculated on every change of **PLAYER_LIGHT**.
- PLAYER_WIDTH** Minimal distance of the player from walls and objects; used for the collision detection. Default and minimum value 1.2 steps.
- PLAYER_SIZE** Distance between the 'feet' and the 'eyes' of the player in steps (default = 3 steps). Essential for calculating the eye level as well as for collision detection.

- **PLAYER_CLIMB** Maximum height difference in steps, which the player can overcome when changing regions - e.g. when ascending a staircase. Used for the collision detection. Default = 1.5 steps.
- **WALK_PERIOD** The quantity of steps for a period of rhythmical up/down or back/forth movements dependant on the course being followed, to represent walk or swim motions. Default = 4 steps. Simultaneously the floor texture sound is repeated as the sound of the player's footsteps.
- **WALK_TIME** Time constant for the **WALK** motion. Default = 4 Ticks.
- **WAVE_PERIOD** Number of ticks for a period of rhythmical time-dependant player movements. Default = 16.
- #WALK** Current steering of the rhythmical player walk motion (-1.0 .. +1.0).
- #WAVE** Current steering of the rhythmical wave motion (-1.0 ... +1.0).
- **PSOUND_VOL** Relative volume of the player's sound (sound of the floor texture) with a range of 0..2 (default = 1). The volume with which the sound is played results from the product of (**PSOUND_VOL** * **texture.SVOL**).
- **PSOUND_TONE** Pitch of the player's sound (sound of the floor texture) with a range of 0..4 (default = 1).

- ▶ **PLAYER_VX**
▶ **PLAYER_VY**

Speed of the player in X and Y direction of the net of coordinates in steps per tick. By changing this skill, the player can be moved horizontally; is automatically adapted in case of collisions with walls or objects.
- ▶ **PLAYER_VZ**

Speed of the player in Z-direction (vertical).
- ▶ **PLAYER_VROT**

Speed of rotation of the player's line of sight (**PLAYER_ANGLE**) in radians per tick. By changing this skill, the player's head can be turned around his vertical axis.
- ▶ **PLAYER_TILT**

Tangent of the player's vertical line of sight (turning around the transverse axis). By changing this skill the view of the player is lifted or lowered.
- ▶ **PLAYER_ARC**

Range of the player's eye's focal length in radians (0.2..2.0). May be changed during the game for 'zoom'- or 'stoned'-effects. The default value 1.0 represents approximately the field of vision of the human eye (ca. 60°).
- ▶ **FRICTION**

Coefficient for friction on the **ACCEL** instruction; range from 0 to 1, default 0.5.
- ▶ **INERTIA**

Coefficient for inertia (mass) on the **ACCEL** instruction; any value over > 0, default 1.
- ▶ **SHOOT_RANGE**

Maximum distance of an object for the **SHOOT** instruction; default 500 Steps.

► SHOOT_-SECTOR	Range (in radians) for the valid angle for the SHOOT instruction from object to player; default = 2 π (360°). Needs to be set to the desired value before each SHOOT instruction. If the player is outside this sector (relative to the actor's line of sight), SHOOT will always return a HIT_DIST and a RESULT of 0.
► SHOOT_FAC	'Hit-power' factor for the RESULT calculation on the SHOOT and EXPLODE instructions; range 0..1, default 1.
► SHOOT_X ► SHOOT_Y	Horizontal and vertical deviation of the SHOOT direction from the center of the 3-D window or from the center of the object within a range of value of 0..1.
# HIT_DIST	Distance to the last object hit by SHOOT or EXPLODE instructions in steps. If no object was hit, this skill is set to 0.
# HIT_MINDIST	Distance of the object hit strings by an explosion from the center of EXPLODE .
# RESULT	This skill represents the 'power' of an event, the impact of a hit or the result of an action, generally in a range of 0..1.
# SHOOT_ANGLE	Angle of the shooting actor relative to the player; is set by a SHOOT instruction and can be used to set the initial angle of a bullet.

#HIT_X**#HIT_Y**

Coordinates of the texture pixel hit by a **SHOOT** instruction, relative to the upper left corner of the texture. These skills are also valid during texture mouse actions (**IF_CLICK**, **IF_TOUCH**). For walls **HIT_X** returns the horizontal pixel coordinate of the **SHOOT** hit or the mouse click, including the texture offset (**OFFSET_X**). Can be used to place bullet holes onto walls, e.g. through the following action:

```
BMP    hole_map, <hole.pcx>;
TEXTURE hole_tex { BMP hole_map; }
SYNONYM last_hitobj { TYPE WALL; }
ACTION set_hole {
    SHOOT;
    IF (HIT == NULL) { // nothing hit?
        END;
    }
    // something hit before?
    IF (last_hitobj != NULL) {
        SET last_hitobj.ATTACH, NULL;
    }
    // remove hole from last wall
    SET HIT.ATTACH, hole_tex;
    // and place it onto new
    SET last_hitobj, HIT;
    SET hole_tex.POS_X, HIT_X;
    SET hole_tex.POS_Y, HIT_Y;
}
```

►SKIP_FRAMES

Actors which aren't visible and are outside their **DIST** from the player normally move only each 6th frame (covering a greater distance), to save rendering time. This skill (default 5) allows you to redefine the number of skipped frames.

►ACTOR_CLIMB

With this skill (default 1) you can define the maximum height of steps which actors can climb on stairs during their **TARGET** movements (like **PLAYER_CLIMB**, see below).

► **ACTOR_WIDTH**
► **THING_WIDTH**

These skills (default = 1) allow you to define a factor the texture width of all actors or things is to be multiplied with for the purposes of collision detection.

ACTOR_-
IMPACT_VX
ACTOR_-
IMPACT_VY
ACTOR_-
IMPACT_VZ

Rebound of an actor from an obstacle in direction x and y, or by colliding with the floor or ceiling of its region. These skills contain valid values only with the actor's **CAREFULLY** flag set, only within their **EACH_TICK** action and only immediately after an actor movement (**MOVED** flag set).

ACTOR_-
FLOOR_HGT
ACTOR_-
CEIL_HGT

Height of floor and ceiling at the position of the actor. These skills contain valid values only for actors with **TARGET** movement and **CAREFULLY** flag, only within their **EACH_TICK** action and only immediately after an actor movement (**MOVED** flag set).

► **PLAYER_X**
► **PLAYER_Y**

Position of the player on the X- and Y-axis of the coordinate system. Please note: By directly changing the player position no collision detection will occur, and you have to assign the new player region manually!

► **PLAYER_Z**

Absolute height of the player's eye level in steps, relative to the level's zero height (cf. **PLAYER_HGT**). By directly changing the player position no collision detection will occur!

► **PLAYER_ANGLE**

Horizontal line of sight of the player in radians (0..6.28). The angle 0 represents the direction of the positive X axis of the coordinate system (east).

#PLAYER_SIN #PLAYER_COS	Sine and cosine of the horizontal viewing direction. These values are needed for those rules calculating the components PLAYER_VX and PLAYER_VY from the player's speed.
#PLAYER_SPEED	Component of speed in direction of the player's line of sight; is calculated from PLAYER_VX , PLAYER_VY and PLAYER_ANGLE . When moving exclusively sideways this skill is 0; if the player is moving backwards, PLAYER_SPEED is negative.
#ACCELERATION	Acceleration in direction of the player's line of sight; is calculated from changes in PLAYER_SPEED . If the player is moving backward, ACCELERATION becomes negative.
#PLAYER_HGT	Relative height of player's feet above or below the region's floor level; important for correcting the vertical motion. Value is yielded from (PLAYER_Z - PLAYER_SIZE - FLOOR_HGT).
#FLOOR_HGT #CEIL_HGT	Real floor and ceiling height in steps at player position; is automatically calculated from the region floor or ceiling heights, the Z values of the edge vertices and the slope of the region.
►IMPACT_VX ►IMPACT_VY	Collision vector; change of speed by collision with walls or objects in X and Y direction. This values are automatically increased on every collision and have to be manually set back to 0 after evaluation. They may be used to generate an intensified 'bouncing-off' (pinball effect).
#PLAYER_- DEPTH	Depth of an underwater BELOW region below the player, its height of ceiling corresponding to the height of the floor of the player region.

►**PLAYER_LAST_X**,
►**PLAYER_LAST_Y**

These skills contain the last player position during the previous frame cycle and are used for the internal calculation of the skills **STEP**, **WALK** etc. If the player is supposed to meet without any **WALK** changes - eg when moving along with a vehicle - these values may be changed via an action.

►**IMPACT_VX**
►**IMPACT_VY**

Collision vector; change of speed as consequence of the player or a **STICK** actor colliding with a wall in X or Y direction. These values are increased automatically at each collision and have to be reset to 0 manually when evaluated. May be used to generate an increased rebound (pin-ball machine effect).

►**IMPACT_VZ**

Collision vector, change of speed in vertical direction as a result of collisions with floor or ceiling.

►**IMPACT_VROT**

If an actor with target **STICK** bounces against an obstacle, this skill will return its resulting change of angle relative to the player. **IMPACT_VROT** may be used to give the player a spin by rebounding.

#**BOUNCE_VX**
#**BOUNCE_VY**

These skills contain a reflection vector if the player collides with a wall. If they are added to the player's speed, the player is given a realistic 'bouncing' behaviour like a pinball.

#**SLOPE_AHEAD**
#**SLOPE_SIDE**

Ascent of the region floor in direction of the player's line of sight and perpendicular to it (rising per step). May be used for example for movement actions of the player, to raise his sight or let the player slide down a slope.

#SLOPE_X #SLOPE_Y	Ascent of the region floor in direction of the X and Y axis.
#MOVE_ANGLE #DELTA_ANGLE	The skill MOVE_ANGLE contains the real movement direction of the player, the skill DELTA_ANGLE the angular difference between MOVE_ANGLE and PLAYER_ANGLE .
#NODE	Number of the current node with multi-pler games; is yielded from the command line (commercial or professional version only).
►REMOTE_0 ►REMOTE_1	Skills to transmit information and events to a linked up PC with multi-player games (commercial or professional version only).
#TIME_CORR	Time correction factor. On a frame rate of 16 frames/sec. this skill has a value of 1; on higher frame rates the value drops, on lower ones it is proportionally increased. This is used to adapt speeds in the game to different frame rates.
#TIME_FAC	Time correction factor, inversed value of TIME_CORR . The faster the frame rate is the higher this skill becomes.
►TICKS	The value of this skill increases every 1/16 second by 1.
►SECS	The value of this skill increases every second by 1.
►STEPS	The value of this skill increases by the number of steps the player covers on his way.

#FLIC_FRAME

Gives the frame number of the currently playing flic. Can be used to start speech, sound effects or music within a global **EACH_TICK** action at the desired frame number, by instructions like this:

```
IF (FLIC_FRAME==50) {  
    PLAY_SOUND bang, 0, 7;  
}
```

#CD_TRACK

Number of the audio CD track currently playing; is updated with the instruction **PLAY_CD**.

**#MOUSE_LEFT,
#MOUSE_MIDDLE,
#MOUSE_RIGHT,
#JOY_4**

Condition of the mouse or joystick button concerned; 0= not pressed, 1= pressed.

The three mouse buttons correspond to the first, third and second joystick button.

#KEY_ANY

These skill takes on the value of 1 if any key is pressed, otherwise it is 0.

#KEY_F1...KEY_F12,
#KEY_ESC,
#KEY_TAB,
#KEY_SHIFT,
#KEY_CTRL,
#KEY_ALT,
#KEY_SPACE,
#KEY_BKSP,
#KEY_CUU,
#KEY_CUD,
#KEY_CUR,
#KEY_CUL,
#KEY_PGUP,
#KEY_PGDN,
#KEY_HOME,
#KEY_END,
#KEY_INS, #KEY_DEL,
#KEY_PAUSE,
#KEY_CAR,
#KEY_CAL,
#KEY_PLUS,
#KEY_MINUS,
#KEY_ENTER,
#KEY_0...KEY_9,
#KEY_A...KEY_Z

Condition of the key concerned; 0= not pressed,
1= pressed.

#FORCE_AHEAD

Analog value entered by mouse, joystick or keyboard (Cursor up / dn keys) for forward or backward movement of the player; range of value from ca. -1 .. +1. The range of value for the movement skills may be influenced by the keywords **KEY_SENSE**, **JOY_SENSE**, **MOUSE_SENSE**, **SHIFT_SENSE** (s.b.).

#FORCE_STRAFE

Value for left and right movement, entered by the [.] and [.] keys or by pressing **[Alt]** and moving the mouse or joystick sideways.

#FORCE_ROT	Value for the horizontal rotation of the line of sight, entered by the cursor right/left keys or by moving the mouse or joystick sideways.
#FORCE_TILT	Value for lifting and lowering the line of sight, entered by pressing [Pg up] or [Pg dn] .
#FORCE_UP	Value for vertical movements (e.g. jumping or ducking), entered by pressing the [Home] or [End] key.
►KEY_SENSE	Sensitivity of the player movements by operating the cursor key, [,] , [.] , [PgUp] , [PgDn] , [Home] and [End] . The skill gives the value for the corresponding FORCE skill on operating the button (default = 0.7).
►SHIFT_SENSE	Factor for raising the keyboard values for movement by pressing the [Shift] -key (default = 2).
►MOUSE_SENSE	Sensitivity of movement by mouse. The value of 1 represents middle sensitivity (default= 1).
►JOY_SENSE	Sensitivity of movement by joystick. The value of 1 represents a middle sensitivity (default= 1).

The following skills are for the purpose of testing only. They are available in the editor WED.EXE, but not in the runtime module VRUN.EXE:

#ERROR

Error code for the current image; may be displayed in a panel. The following error codes are possible:

- 0 = no error,
- 1 = **CLIP_DIST** too small,
- 3 = **NEXUS** too small
(too many walls visible),
- 5 = missing Bitmap,
- 6 = missing region texture,
- 7 = faulty sound,
- 8 = incorrect use of synonyms,
- 9 = missing region.

The last error code is held until a new error occurs. The occurrence of an error in the view is also signalled by a clicking noise from the PC speaker.

►DEBUG_MODE

If this skill is set to 1, the program stops after every frame cycle until the **[S]**-key is pressed (Single Step Mode). May also be activated by **[Ctrl][Alt][S]** or by the command line option - **SST**.

#MAX_DIST

Distance of the farthest visible vertex within **CLIP_DIST**; can be used to optimize the region **CLIP_DISTs**.

#ACTIONS

Number of currently active actions.

#ACTIVE_NEXUS

Utilization of the nexus.

#ACTIVE_TARGETS

Number of actors with **TARGET**.

#ACTIVE_OBJTICKS

Number of objects with running **EACH_TICK** actions.

13. Changing levels, saving the game, and multiplayer mode

Changing levels with the instruction **LEVEL** corresponds to restarting the game. All skills and parameters are reloaded from the WDL file, except the **LOCAL** and **GLOBAL** skills. For that, in both the new level and the old one exactly the same **LOCAL** and **GLOBAL** skills need to be defined in the same order; the 'normal' skills may differ. **GLOBAL** skills allow you to buffer parameters and 'transport' them to the new level, e.g. combat prowess and equipment of the player.

When changing the topography with the instruction **MAP**, all skills are kept, with exception of the player position. All walls, things, regions and actors are loaded from the new WMP file.

With the instructions **SAVE** and **LOAD** game scores may be saved and reloaded. The following parameters of the game will be saved:

- # The name of the actual .WDL- and .WMP file,
- # all **PLAYER** and **GLOBAL** skills (**LOCAL** skills are not saved),
- # all synonyms,
- # the state of the **EACH_TICK** list,
- # all key actions (**IF_F1** etc.),
- # the actual state of all currently running actions,
- # the positions and all other parameters of any **VIEW**, **PANEL**, **TEXT**, or **OVERLAY**,
- # the positions, skills and other parameters of all actors,
- # the flags of all regions, walls and things,
- # the positions, skills and all other parameters of any region, wall, thing, and texture, for which the flag **SAVE** is set.

In order to save the **ATTACH** texture of an object, the **SAVE** flag of this texture has to be set.

Please note that all **EACH_TICK** actions running at the time of saving are started anew after reloading the game. If another level is running at the time of loading, the levels will be changed automatically. The loading itself is - like on **LEVEL** and **MAP** - not carried out directly in the action with the **LOAD** instruction, but rather at the end of the action.

With the instructions **SAVE_INFO** and **LOAD_INFO**, which will work exclusively on **GLOBAL** skills, general parameters of a game, e.g. sound volume or screen resolution can be saved independently of game score saving. Also bitmaps changed with **FREEZE** are saved with this command. In the tutorial actions for level changing and game saving are discussed in more detail.

To let two player play the game as opponents (*Deathmatch* mode) two PCs may be linked via a nullmodem cable - e.g. a serial laplink-, Norton Commander-, or Interlink transmission cable - or be connected via a local area network (LAN) with IPX protocol. The same level must run on both computers. In order to establish the link the number of the serial port for communication must be given via command line option (option **-COM *n***) as well as the 'network number' (**-NODE 0** or **-NODE 1**) when starting the program. If you omit the **-COM** parameter, an IPX connection will be assumed. The **NODE** given on start up may be queried in actions with the skill **NODE**.

After starting, the engine will check wether an ACKNEX session is currently running on a serial link or the IPX network. If yes, the session will be joined; otherwise a new session will be opened.

In the level the 'opposing' player is represented by an actor with the **TARGET NODE0** or **NODE1**. Such a actor will perform the movements of the player on the PC linked up to the corresponding network number **NODE 0** or **NODE 1**. The PCs linked up in the simple serial multiplayer mode do not transmit the position of actors nor the changes to objects and regions. Nevertheless simple actions (firing, or moving lifts or doors) may be transmitted to the other PC by evaluating **REMOTE** skills.

14. User interface: Views, Panels, Texts and Overlays

At game start the 3-D window fills the entire screen; at first no menu is visible. The user interface, however, can be freely defined with the help of overlays, panels, and further 3-D windows (views).

Graphic elements within the 3-D window, like cockpits, tools, weapons, or inventory items, are best defined as **overlays**. Overlays are drawn faster than panels, but they have no buttons or displays and need three times the memory space. Up to 16 overlays may be shown one over the other.

OVERLAY *Keyword* { ... }

This assigns an overlay definition to the keyword. It may contain - in given order - the following keyword assignments:

LAYER *number;*

Determines the order of the overlay, if it overlaps with other screen elements. Elements with higher **LAYER** value will be placed over elements with lower **LAYER** value. The **LAYER** parameter cannot be changed during the game.

►**POS_X** *x;*

►**POS_Y** *y;*

Distance of the upper left corner of the overlay from the upper left corner of the first 3-D window or - if flag **ABSPOS** (see below) is set - from the upper left corner of the screen in pixels. By changing these values the overlay may be shifted across the screen during gameplay.

OVLYS *Ovly;*

The keyword for the actual overlay a bitmaps needs to be assigned to with the keyword **OVLY**.

#**SIZE_X**

#**SIZE_Y**

Size of the overly bitmap in pixels; may be evaluated in actions

FLAGS *Keyword1,Keyword2...;*

Here a list of keywords (flags) determining the properties of the overlays may be given. The following flags may be set:

ABSPOS

If this flag is set, the overlay position (**POS_X**, **POS_Y**) does not refer to the upper left corner of the 3-D window, but rather to the upper left corner of the screen.

►VISIBLE

Only by setting this flag the overlay will appear on the screen.

The mouse pointer is a predefined synonym for an overlay:

►MSPRITE Overlay;

This overlay appears as mouse pointer on the screen. The mouse pointer is moved with the predefined skills **MOUSE_X** and **MOUSE_Y**. The overlay parameters **POS_X** and **POS_Y** define the 'Hot Spot' pixel of the mouse pointer where clicking and touching will be performed.

Panels are used to represent operating features, displays, and images. Panels should be outside the 3-D window, otherwise their background has to be redrawn on every frame cycle, which costs computing time (s. Flag **REFRESH**).

PANEL *Keyword { ... }*

This defines a display panel with the name *keyword*. A panel definition may contain - in given sequence - the following assignments:

PAN_MAP *Bmap;*

Bitmap keyword for the panel background. The size of this bitmap determines the size of the panel. With every change of the panel position (**POS_X**, **POS_Y**) the background bitmap will be redrawn.

LAYER *number;*

Determines the order of the panel, if it overlaps with other screen elements. Elements with higher **LAYER** value will be placed over elements with lower **LAYER** value. The **LAYER** parameter cannot be changed during the game.

► **POS_X** *x;*► **POS_Y** *y;*

Distance of the upper left border of the panel from the upper left border of the screen. If these values are changed during gameplay by a **SET** instruction, the complete panel is drawn anew.

FLAGS *Keyword1, Keyword2...;*

Here a list of keywords (flags) determining the properties of the panels may be given. The following flags may be set:

TRANSPARENT

If this flag is set, the panel background as well as the bitmaps of the **PICTURE** and **WINDOW** display features (see below) will be rendered transparently, i.e. the color #0 will be suppressed.

REFRESH

If this flag is set, panels and displays are redrawn at every frame cycle. This is required e.g. for displaying a panel above the 3D window or having a text scroll across the panel without overwriting the background. Without **REFRESH** a display element is only redrawn if the corresponding skill is changed. Especially on larger panels constant redrawing costs considerable rendering time which reduces the frame rate; therefore in some cases it's better to use an **OVERLAY** for the panel background.

RELPOS

If this flag is set, the panel position (**POS_X**, **POS_Y**) is not given relative to the upper left corner of the screen, but rather to the upper left corner of the 3-D window. Panels with flag **RELPOS** set are clipped off at the edges of the 3-D window.

Within the following display element definitions the given **x,y** positions refer to the distance of the upper left corner of the respective element to the upper left edge of the panel:

BUTTON *x,y,BmapOn,BmapOff>ActionOn>ActionOff;*

Button on the panel. The size of the given button corresponds to the size of the bitmap with the keyword given by *BmapOn*. Normally the bitmap *BmapOff* appears on the panel; at clicking left on it the bitmap changes to *BmapOn*, and the action *ActionOn* is triggered. Release of the mouse button changes the bitmap again to *BmapOff* and triggers *ActionOff*. If the button is supposed to be invisible when inactive, use the keyword **NULL** instead *BmapOff*.

VSLIDER *x,y,len,Bmap,Skill;*

HSLIDER *x,y,len,Bmap,Skill;*

Vertical or horizontal slider for entering skills. The value *len* corresponds to the height or width of the slider range in pixels. The bitmap *Bmap* is used for the slider button, which can be pulled with the left mouse button. The skill borders (**MIN**, **MAX**) determine the value range of the slider: At start position the skill is given the value **MIN**, at end position **MAX**.

VBAR *x,y,len,Bmap,fac,Skill;*

HBAR *x,y,len,Bmap,fac,Skill;*

Vertical or horizontal bar graph display for the graphical representation of a skill in the panel. The bitmap is shifted vertically depending on the skill value. *len* is the vertical or horizontal size of the display in pixels. *fac* is a fixed point number, which when multiplied by the skill value results in the vertical shifting in pixels. The bitmap shown must have a minimum of $(len + fac * (\text{skill value maximum}))$ pixels in height resp. width.

DIGITS *x,y,len,Font,fac,Skill;*

Numeric display. Font is a character set either consisting of 11 characters (numbers 0..9 and space) or of 128 characters in ASCII order. The integer part of a skill is shown with *len* digits in the panel. The skill value is first multiplied by the factor *fac*. Please note the limited accuracy of skill arithmetics; a skill with value 0.1, which is multiplied by 1000 may be shown in the panel as '99'! Leading zeros are suppressed. If the font does not contain a 'minus' character, negative values are not shown. Instead of showing numbers you can show pictures or symbols with a one-digit display.

PICTURE *x,y,Texture,Skill;*

Animated picture. The given texture may contain up to 32 **SIDES**, which may be animated by **CYCLES** and change depending on the integer value of the given skill (1 = first side, 2 = second side and so on). The bitmaps of the texture will be shown unscaled.

WINDOW *x,y,dx,dy,Bmap,Skillx,Skill;*

Displays a cutout 'window' from a bitmap. The values *dx* and *dy* determine the size of the cutout in pixels. The source bitmap *bmap* must not be smaller than *dx* and *dy*. Both *Skillx* and *Skill* give the position of the cutout window on the bitmap in pixels, relative to the upper left corner. The window may only be placed inside the bitmap's borders.

MSPRITE *Overlay;*

Overlay for an alternative mouse pointer within the panel. If this keyword is not given, the mouse pointer keeps its normal look.

►IF_KLICK Action;

Instead of the 'global' **IF_KLICK** action, this given action is performed by left clicking with the mouse pointer anywhere within the panel background.

Texts are used for menus, for messages or for the dialogue with actors. They are constructed like bitmaps and textures: The 'raw' text is defined by the keyword **STRING**, the formatted text by the keyword **TEXT**.

STRING *Keyword*, "Text";

Definition of a character sequence with the name *Keyword* and the content *Text*. This text is between double quotation marks. Line feeds within the text are represented as '\n', quotation marks as '\"'.

TEXT *Keyword* { ... }

This defines a formatted text with the name *Keyword*. A text definition may contain - in given order - the following keywords:

LAYER *number*;

Determines the order of the text, if it overlaps with other screen elements. Elements with higher **LAYER** value will be placed over elements with lower **LAYER** value. The **LAYER** parameter cannot be changed during the game.

►POS_X *x*;**►POS_Y *y*;**

Distance of the upper left edge of the text from the upper left edge of the screen. The position may be outside the screen, but then only the visible part of the text is represented. The position of the text may be changed during the game. In this way longer texts may be scrolled horizontally or vertically over the screen.

►SIZE_Y *n*;

Height of the displayed text in pixels (default and maximum = height of screen). The text is only displayed within the vertical range of **POS_Y** and **POS_Y+SIZE_Y**. Characters situated outside this range are suppressed.

►OFFSET_Y *n*;

Number of the first pixel line of the text, which will be displayed on **POS_Y** position. This parameter allows you to scroll the text vertically pixel by pixel.

STRINGS *n*;

Maximum number of strings this **TEXT** may contain. The strings will be represented in succession.

FONT *Font;*

The character set for the text; must contain either 128 or 256 characters in ASCII order.

#SCALE_X**#SCALE_Y**

Size of a single character of the **FONT** in pixels; may be evaluated in actions.

►STRING *Keyword1,Keyword2...;*

The actual text, containing of one or more strings (define maximum number in **STRINGS** beforehand!). *Keyword* is the keyword for a previously defined text string. It may be changed in actions during gameplay. If a string keyword **NULL** is assigned, no text is represented.

►INDEX *n;*

Number of the string to be changed with the **STRING** keyword by an action; range 1..**STRINGS**, default 1.

FLAGS *Keyword1,Keyword2...;*

Here a list of keywords (flags) determining the properties of the text may be given. The following flags may be set:

CENTER_X**CENTER_Y**

Setting these flags causes the text to be centered by **POS_X** horizontall and by **POS_Y** vertically. If the respective flag is not set, the text will be displayed justified left or up resp.

CONDENSED

Setting this flag causes the text to be 'compressed' horizontally by 1 pixel per character. Especially fonts in italics look better that way.

NARROW

Similar to **CONDENSED**, but text is compressed further.

►VISIBLE

Only by setting this flag the text will appear on the screen.

For the creation of rear mirrors, missile cameras or the like, or for multiplayer games multiple 3D windows can be displayed (with the commercial or professional version only). Each 3D window, except the original one, can be created through the **VIEW** definition:

VIEW *keyword* { ... }

defines a 3D window with name *keyword*. Within the view definition the following keywords can be given:

LAYER *number*;

Determines the order of the view, if it overlaps with other views. Views with higher **LAYER** value will be visible over views with lower **LAYER** value. Please note that panels, overlays, and texts will be displayed over all 3D windows independently of their **LAYER** values.

►POS_X *x*;

►POS_Y *y*;

Distance of the upper left border of the view from the upper left border of the screen.

►SIZE_X *dx*;

►SIZE_Y *dy*;

Size of the view in pixels. For the position and size the same restrictions as for **SCREEN_X**, **SCREEN_Y**, **SCREEN_WIDTH**, **SCREEN_HGT** apply.

►GENIUS *Keyword*;

Keyword of a thing or actor. The view represents a look through his eyes. If the actor is moving, he should have the **FAR** flag set.

FLAGS *Keyword...;*

Here keywords (flags) determining the properties of the view may be given.
The following flag may be set:

►VISIBLE

Only by setting this flag the view will appear on the screen.

Example for a SVGA screen horizontally split into two 3D windows:

```
ASPECT  0.5;                // to get the right proportions
SKILL SCREEN_X { VAL 0; }  // 1st View, upper half of screen
SKILL SCREEN_Y { VAL 0; }
SKILL SCREEN_WIDTH { VAL 640; } // SVGA mode only!
SKILL SCREEN_HGT   { VAL 240; }

VIEW my_view {
  POS_X      0;              // 2nd window, lower half
  POS_Y      240;
  SIZE_X 640;
  SIZE_Y 240;
  GENIUS my_actor; // You'll look through his eyes
  FLAGS      VISIBLE;
}
```

Publishing your game

Let's assume your game or your 3-D application is ready now, and runs as desired if started with WED -RUN. How can this stuff now become a distributable product on a CD-ROM?

Runtime modules creation

Since V3.9, ACKNEX supports two operating systems: DOS and Windows 95/98 via DirectX 5. So you can choose whether to deliver your game together with the DOS or with the windows runtime module (or with both).

To create the runtime modules, open the starting level of your game with WED. Now select **PUBLISH** (lite or commercial) respective **COMPILE & PUBLISH** (professional version). After that you'll find some new files in your current game directory: WRUN.EXE and WWRUN.EXE (using **PUBLISH**) resp. VRUN.EXE, WVRUN.EXE and a .WRS resource file (using **COMPILE & PUBLISH**). Additionally, you have to copy the WWRUN.MDF and WWRUN.WDF files from the WED directory into your game directory. Please note that every time you compile a resource file for the further levels of your game, the runtime modules will be created anew; you may erase them, you'll need only the modules for your starting level.

Now you can test your game, using the following DOS commands:

WRUN <i>levelname</i>	to start the DOS version
WWRUN <i>levelname</i>	to start the windows version
VRUN <i>levelname</i>	to start the DOS version with resource
WVRUN <i>levelname</i>	to start the windows version with resource

If your game won't start, it will be most likely due to the following reasons:

- # One of your levels exceeds the restrictions for royalty free publication (see below).
- # You've forgotten to add your main WDL file to the resource using the **BIND** keyword.

- # WWRUN.MDF or WWRUN.WDF are not copied into the runtime modules directory.
- # The resource file has a different name than the main WDL file.
- # On your PC one of the Win95B versions is running, with which due to a bug windows applications cannot be started out of a DOS box. In this case create a desktop link and start it this way.
- # DirectX 5 is not installed on your PC.

Now you can create the necessary links and icon files to start the game by double clicking on it - we assume taht you're familiar with windows and know how to do it - and then copy all stuff into a single directory, which then can burned onto the CD-ROM. The game will start directly from CD-ROM. A separate setup program is not necessary. If you want to include an DirectX 5 installation package, you can obtain it from the Microsoft site. For distribution of DirectX 5, please regard the license terms of Microsoft, Inc.

Features

If you need additional features for your game, you can **upgrade** from one version to the next one - the versions are 100% upwards compaible. We'll normally release one major **update** per year. Look at the list of upgrade fees, prices and new features at our web page to decide wether it's worth the money. In the following table you'll find listed all features differing for the three GameStudio versions as of V3.9:

Feature / Version	Lite	Commercial	Professional
Max. objects per level (walls+ things+ actors)	1500	1500	10000
2-Player mode	no	serial link	serial link & network
Multiple views	no	yes	yes
Max. resolution	320x400	640x480	800x600

Shutter glasses	no	no	yes
FLIC player	no	no	yes
Polygon models	no	wireframed	textured
CD audio	no	no	yes
OUTPORT/INPORT	no	no	yes
LEVEL instruction	no	no	yes
Resource compiler	no	no	yes
Runtime modules	yes	yes	yes
Email support	no	90 days	unlimited

If you need some **special features** for your game which are not yet supported by the engine, just ask us. We'll develop **ACKNEX** game engines especially adapted for your purpose. You'll find the latest news about 3D GameStudio on our web page.

Royalties

Normally, you may distribute games or demos created with 3D GameStudio **royalty free**. Exceptions are games performed at public places, in arcade machines, at events, fairs, TV shows and so on; or 'big' games with polygonal objects (MDLs) or with more than 1500 objects per level. For such games the runtime module created by WED won't suffice; you have to purchase a special runtime module from us. Just give us the 'Magic Key' number WED displays for your starting level resource file so we can produce the module and send or email it to you (normally within one working day). Please note that this runtime module will only run with the corresponding resource file; if you recompile it - even if nothing was changed with the single files - the 'Magic Key' may change.

For the runtime module a single **royalty** payment is due to Conitec, which is dependent on the price and distribution of your game and listed in the following table. Alternative royalty agreements are possible in certain cases.

End user price	below \$30	\$30 and above
Demo	\$100	- -
Shareware game (self distributed)	\$1500	\$2000
Advertising game	\$2500	\$2500
Commercial game * (national distribution)	\$4000	\$7500
Commercial game * (international distribution)	\$8000	\$15000

* Game which is sold by a distributor, or sold in shops

Depending on the version, you've got some graphic files together with 3D GameStudio. The graphics distributed in the **DEMO** directory are not royalty free; you may use them for exercising, but not for distribution. If you've got the action game in the **SKAPH** directory, you may use these graphics royalty free - but only for your games produced with 3D GameStudio.

Problems & Hints

Updating from a previous version

Version 3.9 contains two differences to previous versions, which may in seldom cases - depending on your WDL files - lead to an undesired behaviour.

The first one is the handling of divisions by zero in **RULE** instructions. These were tolerated by older versions, but not by V3.9 and above. So if you experience a "division by zero" abort in your game, please check the indicated action carefully! By limiting the dividend or divisor values, take care that the result of a division never will exceed a value of $\pm 2,000,000$.

The second difference is a bug fix. Let's suppose you have an actor in your level which fires rockets at the player. This is normally done by defining some rocket actors, setting them at the bad guy's position and letting them run towards the player with **TARGET BULLET**. To prevent that the rocket immediatly collides with the bad guy himself, and explodes, you have either to place it outside his collision radius, or you have to set him **PASSABLE** for one frame before firing the rocket.

If you've forgotten this, V3.9 won't work, but the old versions will sometimes. Why? You'll remember that to save rendering time, an unseen actor moves only every 6th frame (see **SKIP_FRAME** skill). Due to a bug in all engines before V3.9, the skipping of 6 frames will also happen immediately after clearing a **BULLET** actor's **INVISIBLE** flag. So the bullet will not appear at the position it was placed at - i.e. inside the actor - but somewhere in the air. This looks bad, and prevents firing at a very short distance, so we've removed this bug. But the fix now may cause the side effect that the bullet won't run at all. Don't forget to take care of the **PASSABLE**!

Windows version problems

The windows 95/98 runtime module has many advantages, but also a disadvantage to the DOS runtime module: It will need DirectX 5 installed, and some older graphics cards' directx drivers won't support the 320x240 or 320x400 resolution. So if you're developing a game using 320x400, please take care that at least one other resolution - e.g. 640x480 - is included also. Also, the drivers especially of some older graphics cards won't support the stereoscopic mode, which is

necessary to use shutter glasses. And if you're seeing white spots in the picture, which don't show up in the DOS version, you've probably forgotten to set color #255 to white...

DOS version problems

The engine integrated in WED is the DOS one. Due to the multitasking it - as well as the DOS runtime module - will run distinctly slower (up to 25%) under Windows 95 than under plain DOS. Additionally there can be problems with the joystick port, which you may see as a 'spontaneous shaking' of the player. In this case start with joystick disabled (option **-NJ**).

Some rather large levels with many bitmaps may exceed the virtual memory automatically reserved for the DOS box. In this case **ACKNEX** will abort start up, returning the message '*Out of Memory*'. If you have sufficient space available on the harddisk, you may increase the virtual memory to the required value with the DOS box setting (click right on DOS icon, ► *Properties* ► *Memory* ► *memory for DPMI*).

Some sound cards don't play midi songs within a DOS screen if there was any song played before under windows. This is an error of the 95 release of Windows 95A and has nothing to do with **ACKNEX**.

Many notebook video chipsets won't run in SVGA mode. Also, even on normal-sized PCs there is a known problem with S3Trio cards produced between mid-1996 until mid-1997. Due to a bug in the VESA BIOS of these cards, they won't run in SVGA mode. In all these cases either use a lower video resolution, or produce a windows runtime module to test the game.

You can change between WED and other Windows 95 applications with **[Alt]-[Return]** if WED was started with the option **-VGA**.

Restrictions

Once again a summary of the software restrictions, which should be observed on level design:

Total number of objects (walls, things, actors) per level	10000
Actors per level	500

Regions per level	1000
Walls and things per region	200
Steps per wall (length)	200
Height difference between 2 regions (steps)	200

On the lite and commercial version the total number of walls plus things/actors per level is limited to 1500.

Frame rate

The frame rate which is responsible for the 'smoothness' of the motions can fluctuate by up to 200% depending on level design and player position. The frame rate is influenced not only by the processor performance, size and resolution of the 3-D window, but also by the following factors:

Number of objects (Walls, Things, Actors) in the level: around 25%

Number of objects within **CLIP_DIST**: up to around 30%

(This is the reason why **CLIP_DIST**'s should be small!)

Simultaneously running actions and ways: up to around 50%

Number of visible walls and objects: up to around 200%

Relation of floors,ceilings, things to walls in the picture: around 40%

Relation of inclined to straight floor areas: up to around 100%

Also the movement action of the player has a strong influence on the subjective perception of speed in the game. For a good frame rate on old 486 PC's we recommend an implementation of no more than 2000 walls and objects per level of which no more than 50 should be simultaneously visible. No more than 20 actors with way and **CAREFULLY** flag should move around simultaneously. It should be avoided to let actors run over tilted floors. A large level with widely scattered rooms is better for the performance than a small, narrow level with rooms laid close together.

Index

- .WEX 24
- + = 76
- = 76
- 3D_SIMEYE 20, 102
- C 5, 20
- COM 19, 123
- CT 5
- d 19, 31
- E 20
- GOD 20
- I 4
- IPX 19
- NC 20
- NJ 20
- NM 20
- NODE 19
- OM 19
- OS 19
- OT 19
- RUN 4
- S 4
- SST 20
- VESA 4
- VGA 4
- W1 20
- W2 20
- WND 20
- X 5
- XT 5
- * = 76
- / = 76
- 3-D glasses 102
- ABOUT 12
- ABS 76
- ABSPOS 124, 125
- ACCEL 75, 111
- ACCELERATION 115
- ACOS 76
- ACTION 74
- ACTIONS 121
- ACTIVE_NEXUS 121
- ACTIVE_OBJTICKS .. 121
- ACTIVE_TARGETS .. 121
- ACTOR 17, 63, 70
- ACTOR_CEIL_HGT .. 114
- ACTOR_CLIMB 113
- ACTOR_DIST 105
- ACTOR_FLOOR_HGT
..... 114
- ACTOR_IMPACT_VX
..... 114
- ACTOR_IMPACT_VY
..... 114
- ACTOR_IMPACT_VZ
..... 114
- ACTOR_WIDTH 114
- ADD_STRING 85
- ALBEDO 40, 57
- ALIGN 10
- All IFDEFs 7
- AMBIENT 40, 58, 109
- ANGLE 64, 66, 67
- ASIN 76
- ASPECT 102, 132
- ASPEED 68
- ATTACH ... 42, 50, 63, 90,
122
- ATTACHED 42
- AUTORANGE 34, 83
- BASE 60
- BBM 26, 27, 49
- BEHIND 44
- BELOW 46, 55, 57, 60
- BERKELEY 53, 71
- BIND 31, 133
- BLUR 35, 83
- BLUR_MODE 103
- BMAP 27
- BMAPS 37
- BOUNCE_VX 116
- BOUNCE_VY 116
- BRANCH 79, 98
- BREAK 78
- BULLET 64, 70, 137
- BUTTON 127
- CALL 79, 80, 98
- CAREFULLY .. 64, 72, 73,
89
- CD_TRACK 118
- CDAUDIO_VOL 108
- CEIL_ANGLE 57
- CEIL_ASCEND 59, 61
- CEIL_DESCEND ... 59, 61
- CEIL_HGT 56, 115
- CEIL_OFFSET_X 56
- CEIL_OFFSET_Y 56
- CEIL_TEX 56, 74
- CENTER_X 130
- CENTER_Y 130
- CHANNEL 109
- CHANNEL_0 109
- CHECK 11
- CLIP 46
- CLIP_DIST .. 20, 30, 55, 58,
71, 104, 105, 121
- CLIPPING 55, 104
- CLUSTER 45
- COLOR_ACTORS 68,
106
- COLOR_BORDER ... 51,
106
- COLOR_PLAYER ... 106
- COLOR_THINGS 68,
106
- COLOR_WALLS .. 51, 106
- COMPILE & PUBLISH
..... 126, 133
- COMPILE & PUBLISH . 7
- CONDENSED 46, 130
- CONTINUE 78
- CONTINUE; 78
- COPY 8
- COS 76
- CREATE 8
- CURTAIN 54
- CYCLE 50, 63
- CYCLES . 36, 38, 39, 41, 47,
128
- DARK_DIST 109

Deathmatch	123	FLC	26	HOLD	66
DEBUG_MODE	121	FLI	26	HSLIDER	127
DEFAULT	98	FLIC	28, 38	IBANK	30
DEFINE	31, 32	FLIC_FRAME	82, 118	IBK	26
DELAY	39, 48	FLIP	9	IF	77, 78
DELETE	8	FLOOR_ANGLE	57	IF_ARISE	58
DELTA_ANGLE	117	FLOOR_ASCEND	59, 61	IF_ARRIVED	64, 65, 67, 70
DIAPHANOUS	35, 42, 44	FLOOR_DESCEND	59, 61	IF_BKSP	119
DIGITS	128	FLOOR_HGT	56, 115	IF_DIVE	58
DirectX	29, 133	FLOOR_OFFS_X	56	IF_ENTER	58, 73
DISPLAY	11	FLOOR_OFFS_Y	56	IF_F1...	97
DIST	51, 52, 67-69, 71, 73, 90, 113	FLOOR_TEX	56	IF_FAR	51, 52, 68, 69
DISTANCE	51, 69	FOLLOW	65, 67	IF_HIT	52, 54, 64, 70, 72, 90, 91
DITHER	30	FONT	27, 43, 130	IF_KLICK	43, 97, 108, 128
DROP	90	FORCE_AHEAD	119	IF_LEAVE	58, 73
DRUMBANK	30	FORCE_ROT	120	IF_LEFT	96
DUPLICATES	12	FORCE_STRAFE	119	IF_LOAD	96
DXF	28	FORCE_TILT	120	IF_MIDDLE	96
EACH_CYCLE	52, 54, 60, 70, 72	FORCE_UP	120	IF_MSTOP	97
EACH_TICK	52, 59, 70, 82, 96	FRAGILE	54, 70, 72, 91	IF_NEAR	51, 52, 68, 69, 73
EDIT	8	FRAME	37, 38	IF_RELEASE	43, 108
ELSE	77	FREEZE	88	IF_RIGHT	96
END	79	FRICTION	75, 111	IF_SPACE	119
ENDIF	32	GENIUS	40, 57, 93, 131	IF_TAST	97
ERROR	121	GETMIDI	84	IF_TOUCH	43, 108
event	74	GHOST	44	IFDEF	5, 7, 19, 32
EXCLUSIVE	80	GLOBAL	94, 100, 122, 123	IFELSE	32
EXIT	7, 95	God mode	20	IFNDEF	5, 7, 32
EXP	76	GOTO	79, 80	IMMATERIAL	43, 53, 72, 90
EXPLODE	54, 70, 72, 91, 112	GRID	11	IMPACT_VROT	65, 116
EYE_DIST	102	GROUND	60, 67, 71, 72	IMPACT_VX	65, 115, 116
FADE_PAL	83, 84	HARD	34	IMPACT_VY	65, 115, 116
FAR	55, 71, 104, 131	HBAR	127	IMPASSABLE	53
FENCE	54	HEIGHT	66, 67, 71	IMPORT	8
FILE	6	HELP	12	INDEX	86, 130
FIND	86, 108	HERE	59, 89, 99	INERTIA	75, 111
FLAG1	55, 73	HIT	99	INKEY	85, 108
FLAG8	55, 73	HIT_DIST	90, 91, 112	INPORT	83
FLAGS	34, 44, 53, 59, 70, 130, 132	HIT_MINDIST	91, 112	INT	76
		HIT_X	90, 113	INVISIBLE	53, 70, 137
		HIT_Y	90, 113		

IPX	19, 123	MAP_MAXX	105	128	
JOIN	9	MAP_MAXY	105	MUSIC	28
JOY_4	118	MAP_MINX	105	MUSIC_VOL	108
JOY_SENSE	120	MAP_MINY	105	MY	89, 99
JOYSTICK_X	108	MAP_MODE	105	NARROW	46, 130
JOYSTICK_Y	108	MAP_OFFSX	105	NEXT	9
KEY_A...KEY_Z	119	MAP_OFFSY	105	NEXT_MY	89
KEY_ANY	118	MAP_ROT	106	NEXT_MY_THERE ..	89
KEY_F1...KEY_F12 ...	119	MAP_SCALE	105	NEXT_THERE	89
KEY_SENSE	119, 120	MAPCOLOR	45	NEXUS	29
KEYWORDS	11	MAPFILE	31, 95	NO_CLIP	46
LAYER .. 124, 126, 129, 131		MASTER	73	NODE	117
LAYERS	106	MAX	100, 127	NULL	64, 77
LBM	26, 27, 49	MAX_DIST	121	OFFSET_X .. 37, 39, 42, 50	
LEFT	52	MDL	26, 28, 38, 135	OFFSET_Y .. 37, 39, 42, 50,	
LENGTH	10, 51	MICKEY_X	107	86, 129	
LEVEL	95, 104, 122	MICKEY_Y	107	ONESHOT .. 44, 52, 54, 59,	
LIBER	71	MID	26	60, 70, 72	
LIFT	92	MIDI_COM	84	OPEN WMP	6
LIFTED	59	MIDI_PITCH	30	OUTPORT	83
LIGHT_ANGLE ... 40, 57		MIN	100, 127	OVERLAY	64
LIGHT_DIST	109	MIRROR	39, 48	OVLY	27
LIGHTMAP	45	MODE	7	OVLYS	124
LINES	86, 108	MODEL	28, 38, 45	PALETTE	83
LOAD	94, 104, 122	MODIFY	8	PALETTE	33
LOAD_INFO	94, 123	MOTION_BLUR .. 88, 103		PALFILE	33
LOAD_MODE 94, 95,		MOUSE_ANGLE .. 90, 107		PAN_MAP	125
104		MOUSE_CALM	107	PANEL	125
LOCAL	100, 122	MOUSE_LEFT	118	PASSABLE 53, 70, 137	
LOCATE	89	MOUSE_MIDDLE 118		PATH	31
LOG	76	MOUSE_MODE . 106, 107		PCX	26, 27
LOG10	76	MOUSE_MOVING ... 107		PICTURE	128
LOG2	76	MOUSE_RIGHT 118		PLAY	54, 60, 72
Magic Key	7, 135	MOUSE_SENSE .. 119, 120		PLAY_DEMO	95
MAP	95, 122	MOUSE_TIME	107	PLAY_FLIC	82
MAP_CENTERX	105	MOUSE_X	107, 125	PLAY_FLICFILE 82	
MAP_CENTERY 105		MOUSE_Y	107, 125	PLAY_SONG	81
MAP_COLOR	51, 68	MOVE	64	PLAY_SONG_ONCE . 81	
MAP_EDGE_X1	105	MOVE_ANGLE	117	PLAY_SOUND	80
MAP_EDGE_X2	105	MOVE_MODE	104	PLAY_SOUNDFILE .. 81	
MAP_EDGE_Y1	105	MOVED	72	PLAYER_ANGLE 90,	
MAP_EDGE_Y2	105	MPU401	19	111, 114	
MAP_LAYER	106	MSPRITE 64, 106, 125,		PLAYER_ARC	111
				PLAYER_CLIMB 110	

PLAYER_COS	115	REPEL	65	SHOOT_RANGE . . .	90, 91, 111
PLAYER_DEPTH . . .	115	RESET HOR	9	SHOOT_SECTOR . . .	112
PLAYER_HGT . . .	60, 115	RESULT . . .	86, 90, 91, 112	SHOOT_X	90, 112
PLAYER_LIGHT	109	RIGHT	52	SHOOT_Y	90, 112
PLAYER_SIN	115	ROTATE . . .	57, 59, 60, 93	SIDES	39, 47, 128
PLAYER_SIZE	109	ROTATE & SCALE . . .	9	SIGN	76
PLAYER_SPEED	115	RULE	76, 137	SIN	76
PLAYER_TILT . . .	37, 111	SAVE . . .	55, 59, 73, 93, 122	SIZE_X . . .	51, 69, 124, 131
PLAYER_VROT	111	SAVE AS	6	SIZE_Y . . .	69, 86, 108, 124, 129, 131
PLAYER_VX . . .	111, 115	SAVE WMP	6	SKIP_FRAME	137
PLAYER_VY . . .	111, 115	SAVE ALL	59	SKIP_FRAMES	71, 113
PLAYER_VZ	111	SAVE_DEMO	94	SKY	45, 104
PLAYER_WIDTH	109	SAVE_INFO	94, 123	SKY_OFFS_X	37, 102
PLAYER_X	114	SAVEDIR	31, 93	SKY_OFFS_Y	37, 102
PLAYER_Y	114	SCALE_X	39, 130	SLOOP	46, 56
PLAYER_Z	58, 114	SCALE_XY	47, 49	SLOPE_AHEAD	116
POLYGON	11	SCALE_Y	39, 130	SLOPE_SIDE	116
PORTCULLIS	54, 60	SCREEN_HGT . .	102, 131	SLOPE_X	117
POS_X . . .	42, 99, 124-126, 129, 131	SCREEN_WIDTH . .	102, 131	SLOPE_Y	117
POS_Y . . .	42, 99, 124-126, 129, 131	SCREEN_X	102, 131	SOUND	28, 40
POSITION	50, 54	SCREEN_Y	102, 131	SOUND_VOL	108
PREV	9	SCREENSHOT	89	special features	135
PRINT_STRING	88	SCROLL	11	SPEED	64, 65, 67
PRINT_VALUE	88	SCYCLES	41, 48	SPLIT	9
PRINTFILE	88	SDIST	41, 48	SQRT	76
PSOUND_TONE	110	SECS	117	STARTTEXT	95
PSOUND_VOL . . .	56, 110	SEEN	53, 59, 72	STATISTICS	11
PUBLISH	133	SENSITIVE	54, 72	STEPS	117
RADIANCE	40	SET	74	stereo	41
RANDOM	39, 76	SET_ALL	75	STICK	65, 67
RANGE	34	SET_INFO	86	STICKY	60
RECTANGLE	10	SET_SKILL	85	STOP_DEMO	95
REFRESH	125, 126	SET_STRING	85	STOP_FLIC	82
REGION	56	SETMIDI	84	STR_LEN	85, 108
REGIONS	7	SHADOW	44	STRING	108, 129, 130
REL_ANGLE	65, 67	SHAKE	52, 69	SVDIST	41
REL_DIST	65, 67	SHIFT	60, 93	SVGA	29
RELPOS	126	SHIFT_SENSE . . .	119, 120	SVOL	40, 41
REMOTE_0	117	SHOOT . .	52, 70, 90, 91, 99, 111-113	SYNONYM	98
REMOTE_1	117	SHOOT_ANGLE . .	90, 112	TAN	76
RENDER_MODE	103	SHOOT_FAC	90, 112	TARGET	64, 68, 138

TARGET_X	65-67	TOUCHED	99	WAV	26, 28
TARGET_Y	65-67	TYPE	98, 100	WAVE	110
TEXT	86, 99, 108, 124, 129	UNDEF	32	WAVE_PERIOD	110
TEXTURE	36, 47, 50, 63	UNDO	8	WAY	18, 63, 66, 67
THERE	89, 99	update	134	WAYPOINT	67
THING	17, 63, 70	upgrade	134	WAYS	7
THING_DIST	105	VAL	100	WED	4
THING_WIDTH	114	VBAR	127	WEX	23, 24, 26-28
THINGS	7	vertex	17, 65	WHILE	78, 80
TICKS	117	VERTICES	7	Windows 95	137, 138
TILT	92	VIDEO	29, 102	WIRE	45
TIME_CORR	117	VIEW	131	WRS	133
TIME_FAC	117	VISIBLE	53, 59, 70, 125, 131, 132	WRUN.EXE	6, 133
TITLE	38, 43	VOC	26	WVRUN.EXE	7, 133
TO_STRING	85	VRUN	31	WVRUN.EXE	6, 133
TOP	57	VRUN.EXE	7, 133	WVRUN.MDF	133
TOUCH	43, 46, 99, 107, 108	VSLIDER	127	WVRUN.WDF	133
TOUCH_DIST	43, 108	VSPEED	64, 65, 67, 68	X1	52, 69
TOUCH_MODE	43, 107	WAIT	78, 80	X2	52
TOUCH_RANGE	43	WAITT	80	Y1	52, 69
TOUCH_REG	99	WALK	110	Y2	52
TOUCH_STATE	108	WALK_PERIOD	110	Z1	52
TOUCH_TEX	99	WALK_TIME	110	Z2	52
TOUCH_TEXT	43	WALKTHROUGH	8	ZOOM	11
TOUCH_TEXT	99	WALL	9, 17		
		WALLS	7, 12		