

3D Game Al chemy

For DOOM, DOOM II, Heretic and Hexen

Hexen Reference

Click on one of the chapter titles:

Programming The Action Making Moving Scenery Hexen's Special Codes Hexen's Scripting Language

PROGRAMMING THE ACTION

Passing reference has been made in chapters in the book to Hexen's improved programmability. This chapter takes a closer look at the new features of this particular game variant.

HEXEN'S NEW FEATURES

To provide improved programmability, the designers of Hexen had to alter some of the basic ways in which the game engine operates and also restructure the way data is stored within WAD files. These were the main changes:

- A completely revamped special action code scheme.
- A new linedef structure, permitting more versatile action specification and triggering.
- A new Thing structure, permitting greater control over their actions and allowing them to initiate actions.
- A new scripting capability, to permit total control over individual triggered events.
- A new MapInfo resource, enabling maps to be linked in a new way and providing increased control over the use of sky textures and other level-wide characteristics.
- A new PolyObj construct to provide programmable moving scenery elements.

I'll take a look at each of these changes in turn.

HEXEN'S NEW ACTION CODE SCHEME

At the heart of the changes made to Hexen lies its new action code scheme. Instead of using the fixed, restrictive system of the earlier DOOM variants,

By Steve Benner

the programmers at Raven Software have devised a neat and sophisticated new scheme that allows all the familiar DOOM actions to be invoked, but in a much easier way, and with far greater control and flexibility. The basis of the new scheme is an ability to parameterize each action. No longer is the triggering nature and control of the action tied to the action itself, as it is in earlier variants. In Hexen, the WAD designer can specify not only how each individual action should be triggered, but also how the action is to progress.

Actions in Hexen are now controlled by supplying up to five *arguments* (or *parameters*) when the action is triggered. These arguments are used by each action to control certain aspects of its operation. The new Door_Raise action, for instance, which replaces DOOM's simple line-type 1 (Door: Open and Close), uses the arguments passed to it to determine the speed with which it operates and how long it stays open, as well as determining which sectors it operates on.

NOTE: Full details of each of Hexen's new action codes, and the parameters they take, can be found in "Hexen's Special Codes" on the CD-ROM.

To support this new action code scheme, and to take full advantage of it, other structures within the WAD file have had to be changed—principally the linedef and the Thing resources.

CHANGES TO THE LINEDEF STRUCTURE

Hexen's linedefs have gained the following items:

- A new flag
- A new attribute field
- An extended special action characteristic

All of these changes are designed to make the specification and control of the actions that lines trigger easier and more flexible.

REPEATABLE FLAG

The new line flag in Hexen is the Repeatable flag. The setting of this flag determines whether the line's special action can be triggered more than once. This flag replaces the hard-coded nature of an action's repeatability found in earlier game variants.

ACTIVATION TYPE

The new line attribute is its Activation Type field. This attribute determines how the line's special action code will be triggered. The possible values of this field are given in Table 1.

Value	Line Activated By
0	Player crossing the line
1	Player using the line (with spacebar)
2	Monster crossing the line
3	Projectile impacting the line (includes fist)
4	Player pushing the line (by walking into it)
5	Projectile crossing the line (includes fist)

Table 1. Values for Hexen's Activation Type line attribute.

This attribute replaces the hard-coded nature of this characteristic of line triggerings used in earlier versions of the game.

EXTENDED SPECIAL ACTION CHARACTERISTIC

To be able to pass parameters to its special action, each linedef now has five argument fields in addition to the Special Action Code field.

CHANGES TO THE THING STRUCTURE

One of the most exciting changes to Hexen is its new capability to allow Things to trigger actions by means other than walking across lines. This action is achieved through a new Special Action Characteristic within the Thing structure. This is identical to the new, extended Special Action Characteristic of lines; it consists of a Special Action Code field and five arguments, passed to the special action when it is triggered.

In addition to triggering special actions, Hexen's Things can themselves be the subject of special actions. This requires additional changes to the structure of a Thing, as detailed shortly.

TRIGGERING THING ACTIONS

Things have access to exactly the same set of special actions as do lines. It is now possible, therefore, to use Things to trigger actions that traditionally needed players to move to particular locations—or that required the rather cumbersome and inflexible use of specific strange sector tags, such as DOOM's 666.

In Hexen, the Special Action Characteristic of a Thing is triggered in the following circumstances:

- When the Thing is killed, if it is a monster.
- When the Thing is banished, using the Banishment Device, if it is a monster.
- When the Thing is destroyed, if it is a destructible object, such as an urn or a tree.
- When the Thing is picked up, if it is a gettable object, such as a power-up artifact or puzzle piece.

NOTE: Some special types of Thing do not fall into any of the preceding categories, and they use their Special Action fields for other purposes. Details of all of these types of Things are in Chapter 34, "Essential Thing Information."

THING PARTICIPATION IN SPECIAL ACTIONS

For Things to be subjected to special actions themselves, individual Things need to be capable of being identified. They therefore have a new attribute: their Thing Identification (or *tid*, for short). This attribute operates in exactly the same way as a sector's tag field; it is an arbitrary value, which can be shared by any number of Things if they are required to participate in some group action. Some actions require Things to be tagged to lines or to sectors (or both). This tagging is achieved through the Thing's tid field. I will return to the actions that affect Things later in this chapter.

OTHER CHANGES TO THINGS

Raven Software has made several other changes to the Thing structure in Hexen that affect the way Things are used within a WAD. These changes will be examined shortly.

HEXEN'S SCRIPTING FACILITY

In addition to the increased flexibility of Hexen's special actions, this DOOM variant also provides unprecedented control of the game through its unique scripting language, ACS. Full details of this language are reserved for "Hexen's Scripting Language" on the CD-ROM. For now, the important points to note are how this facility both expands the versatility of Hexen's special actions and binds all of Hexen's new features into a coherent control scheme. ACS achieves this result through its own unique capabilities and its interaction with the other new features of the game:

- ACS scripts have full access to the special action codes.
- ACS scripts can themselves be triggered by other scripts, by special action codes triggered from lines, or from Things.
- ACS scripts can be virtually any length (provided that there is sufficient memory), enabling complex actions to be triggered and controlled.
- ACS provides full conditional control of the execution of its scripts.
- ACS scripts have access to many of the game's internal parameters, allowing game-time decisions to be made and special actions controlled accordingly.
- ACS scripts have access to many of the game's internal control functions, enabling them to control many aspects of game behavior that were previously inaccessible.

Many people regard Hexen's scripting language as the single most important advance in the continuing evolution of DOOM. It is certainly very powerful, and it will amply repay any time spent studying it.

THE MAPINFO RESOURCE

Hexen's new MapInfo resource controls the way maps are arranged to provide Hexen's new notion of level clusters around a series of hubs. It also controls various other aspects of each map. Unlike other map resources in a WAD, the MapInfo resource occurs only once for an entire WAD. Any MapInfo placed in a PWAD must therefore contain information about every map that the player can visit. If your map replaces only selected maps of the original game, and you want to change the default MapInfo information, you need to duplicate the original (unchanged) settings and supply these in your new MapInfo alongside your new settings.

Table 2 details the contents of this resource. Each map that a player can visit must have an entry in the MapInfo resource matching the information given in this table.

Entry Name	Contains
map	Number (1 to 60) of the map followed by its name; for example, 1 "Winnowing Hall".
warptrans	The number of this map if the <code>-warp</code> command-line parameter (or cheat key equiva- lent) is used. This enables you to create a map with one number in the editor and then reassign its position in the WAD through the MapInfo resource without renumbering. It also enables you to get around having gaps in the map numbering sequence. Only values in the range 1 to 31 are valid here.
next	The number of the map to teleport to next on exit of a timed Deathmatch.
cdtrack	The CD track to play as background music while the map is active.
cluster	The cluster to which the map belongs (clusters are explained shortly).
sky1	The name of the texture to use for standard sky in this map, followed by the speed at which it should be scrolled; for example, $SKY1$ 2.
sky2	The name of the texture to use for the alternative sky, and its scrolling speed. This sky is used in sectors with the special sector characteristic 200: Alternate Sky.
doublesky	If specified, this causes the display of $sky2$ behind $sky1$, allowing layered cloud effects.
lightning	If specified, this causes periodic lightning flashes (and crashes of thunder) while the map is active. The sky texture switches from sky1 to sky2 for the duration of the lightning flash.
fadetable	The name of the fade table, or fog map, used while the map is active.

Table 2. Hexen's MapInfo contents.

Listing 1 gives a sample MapInfo entry for a Hexen map.

Listing 1. A sample MapInfo entry.

map 1 "Winnowing Hall"
warptrans 1
next 2
cluster 1
sky1 SKY2 2
sky2 SKY3 0
lightning
doublesky
cdtrack 13

MAP CLUSTERS

Hexen's capability to enable players to move freely backward and forward between maps has serious implications for the engine's capability to keep track of the status of the game. To limit the amount of information that must be tracked and saved, Hexen requires that its maps be divided into *clusters*. You nominate which cluster (identified by number) a map belongs to in the MapInfo resource. After a player has moved from one cluster to another, return is not permitted. (The map starts over if the player ever does return—it is up to you to ensure that your maps link correctly.) It is recommended that the number of maps assigned to any one cluster should not exceed 6 or 7, unless your maps are particularly small. Larger maps might require smaller clusters.

A new Hexen backdrop and milestone text message are given at the end of each cluster.

HEXEN'S PROGRAMMABLE SCENERY

The final new item in Hexen's WAD files is the PolyObj, or Polygonal Object. This construct provides Hexen's unique sliding and rotating walls. This feature of the game is dealt with in full in "Making Moving Scenery" on the CD-ROM.

HEXEN'S THINGS

After the extended action-control features and scripting capabilities of Hexen, one of the most noticeable improvements to its programmability is the enhanced flexibility of the use of Things. These are the additional features of the Thing structure in Hexen:

- A new Thing identification field (tid) to enable Things to be the subject of special actions
- A new Special Action Characteristic to enable Things to trigger special actions
- A new Starting Altitude attribute
- A new Dormant attribute
- New uses for the Facing Angle
- New controls to determine a Thing's appearance in single-player games
- Improved control over Things' appearance in multiplayer games

Some of these new features have already been examined. I'll now look at the rest.

NEW THING ATTRIBUTES

Hexen has introduced two new attributes of Things: a Starting Altitude and Dormancy.

STARTING ALTITUDE

All Things in Hexen can be assigned a Starting Altitude. This attribute determines how far above its sector floor the Thing is placed when the map starts. The Thing is subjected to gravity immediately when the player enters the map for the first time, so this attribute is of any real use only for Things placed close to a Player Start position—unless you can think of a special use for it....

DORMANT THINGS

Dormancy in monsters is a concept new to Hexen. It relies on the interrelationships between Things and other actions for its effectiveness. Things with the Dormant attribute cannot be awakened by any of the normal means, but wake up only when activated by a Thing_Activate special action. Thus, you can hold back particular monsters until specific points during play. A more detailed look at the special codes acting on monsters and other Things follows shortly.

CONTROLLING THINGS' APPEARANCES

Hexen provides new flags to determine when Things should appear in Hexen games. The difficulty level flags used in earlier game variants have been extended to include Player Class flags—controlling whether Things appear during different types of single-player games—and also better control over appearances in multiplayer games. These are the complete appearance control flags for Things in Hexen:

- Appears on Easy skill settings (1 and 2)
- Appears on Normal skill setting (3)
- Appears on Hard skill settings (4 and 5)
- Appears for players of Fighter class
- Appears for players of Cleric class
- Appears for players of Mage class
- Appears in single-player games
- Appears in Cooperative games
- Appears in Deathmatch games

Any of these flags can be set independently of any other, allowing much better control over each Thing's appearance in the game than was possible in the earlier game variants.

NOTE: For multiplayer games involving more than one player class, all Things flagged for appearance for each participating class are present in the game and equally visible to all players.

Figure 1 shows the increased capabilities and attributes of Things in Hexen. Only players of Fighter class playing alone will meet this particular Ettin. They might regret killing it, too.

Figure 1. Setting a Hexen Thing's attributes.

(lass:	Preview:
Ammo a	- water
Lineny P	ASSERTED -
(spe:	
24(1))	
Demon Isader Diagon "[tag, tag, tag, tag, tag]	
211m	1 27
Fighter boss	
	1 CAttributes
Special: Earthquake[intensity, duration, 1	
Argumente: 6 128 50 100 99	Clevel 1/2 Cleric
Position	Lavel 3 Mage
N N	E Level 4/5 Single
X 47 Format O O	Deal Cooperative
101 WO OE	Dogmant Deathmatch
	Fighter
Z 0 00	
s 225	Tag Mumber: 0

NEW USES FOR THE FACING ANGLE

Some of Hexen's Things have a new use for their Facing Angle field. These are specifically tied to the construction of PolyObjs; the details are presented in "Making Moving Scenery" on the CD-ROM.

ACTIVATING THINGS

As already noted, Things in Hexen can be subjected to special actions, in a similar way to sectors. Full details of these actions are given in Table 12 in "Hexen's Special Codes" on the CD-ROM. It is worth looking in a little detail at the codes to control a Thing's dormancy, however. Any Thing flagged as Dormant can be activated only by a special action of type Thing_Activate. In keeping with all other Hexen action codes, this special action can be triggered from any of the following items:

- A line, triggered by player, projectile, or monster
- Another Thing, being killed, destroyed, or acquired
- An Action Control Script, which can itself have been triggered by any of the preceding items

As you can see, the potential for activating monsters has suddenly become very large indeed!

After a monster has been activated, it behaves as any other awakened monster.

Any monster can also be made dormant during play if it is subjected to the Thing_Deactivate special. This special causes the monster to freeze its activity immediately and revert to a dormant state. It can then be reactivated only with the Thing_Activate special action.

EXIT: MOPPING UP AND MOVING ON

This chapter has introduced you to the principal enhancements made to the control of actions and Things in Hexen. The other major addition to that game is its capability to create moving scenery elements. This aspect of the game is the subject of "Making Moving Scenery" on the CD-ROM.

MAKING MOVING Scenery

In this chapter, the attention is turned to another of Hexen's new features—the PolyObj, or Polygonal Object. This new structure is provided to enable the construction of some rather special scenery elements in Hexen—walls that can slide sideways or rotate (or both)!

INTRODUCING THE POLYOBJ

The Polygonal Object—more properly termed merely the *PolyObj*—is a brand-new construct, developed by Raven Software for use in Hexen. From the outset, it is important to realize that the PolyObj is, in fact, only a notional (or functional) construct. It is built from a collection of lines and special Things and is activated by means of some special action codes, in the same way that a door or a lift or a teleport might be, for example. Like most of DOOM's other functional constructs, the PolyObj has some fairly strict constructional requirements.

A POLYOBJ'S COMPONENT PARTS

In essence, a PolyObj consists of the following components:

- A series of lines, marking out a closed shape.
- Appropriate codes on some or all of these lines, assigning them to a particular PolyObj.
- An anchor point, to define the center of rotation of the PolyObj.
- A starting point, to define where on the map the PolyObj will be when the map loads.

In addition, there are some fairly stringent requirements for the arrangements of these components. I'll look in some detail at the nature of each of these components, though, before considering the actual steps that need to be taken to build one.

A POLYOBJ'S LINES

The series of lines that mark out a PolyObj must form a closed shape. This shape is always viewed from its outside—like looking at a solid door, or a pillar, for example. Lines marking out a PolyObj therefore are normally one-sided, facing out from the shape. To be treated as a PolyObj, these lines need to be marked in some way.

MARKING THE LINES OF A POLYOBJ

Lines can be marked as belonging to a PolyObj in two ways, both of them involving special action codes on the lines. The first (and easiest) method is to place the special action code PolyObj_StartLine on just one of the PolyObj's lines. The presence of this code signals to the Hexen game engine that the line belongs to a PolyObj. Hexen locates the remaining lines of the PolyObj for itself by following lines from vertex to vertex from one end of the marked line until a return is made to the other end of that line. For most simple, convex shapes, this method is adequate and it can be used most of the time.

NOTE: Hexen's special actions were introduced in "Programming the Action" on the CD-ROM. The values for these special action codes are given in "Hexen's Special Codes" on the CD-ROM.

If the shape is too complex for the game engine to pick up by line-following, a second method needs to be employed. In this method, the special code PolyObj_ExplicitLine must be placed on every line of the PolyObj. The second argument of this special code provides each line's position in the rendering sequence of the PolyObj. This sequence must start at 1 on the first line to be rendered and must run without gaps and in steps of 1, through each of the lines that make up the shape. This second method provides total control over which lines are incorporated into the PolyObj, as well as the order in which they are rendered by the graphics engine. It is more tedious to use, however, and has the disadvantage of preventing any other special action codes from being used on any of the lines of the PolyObj.

DEFINING A POLYOBJ

To distinguish among the various PolyObjs in use in the level, each PolyObj must be assigned a unique identifying number. This designation is made through the first argument of the special code used to assign its lines. In addition, each PolyObj can be assigned a partner, in the form of another PolyObj. The partner is termed its *mirror*. Provided that the mirror is not already moving at the time, the pair of PolyObjs will operate in concert when the main PolyObj is triggered. The mirror will move in the opposite direction of the primary PolyObj, however, rotating counterclockwise to its partner's clockwise motion, and vice versa. Horizontal motion of the primary PolyObj will be matched by movement of the mirror in the opposite direction. Each PolyObj is assigned its mirror when it is defined, through one of the arguments of the defining special code. In this way, each PolyObj can have only one mirror. A mirror PolyObj must itself be defined as a PolyObj in the usual way, and it can, of course, be treated as a primary PolyObj in its own right. Two PolyObjs can act as mutual mirrors of each other, but there is rarely any need for this action, and Raven Software cautions against it. Generally, though, it seems to cause no problems.

Finally, when a PolyObj is defined by means of either of the special codes just outlined, a sound can be attached to it. This is the sound that will be used when the PolyObj is made to move. Again, this information is passed to the engine via an argument of the defining special action code. Details of the sounds available are given in Table 4 of "Hexen's Special Codes" on the CD-ROM.

POLYOBJ ANCHOR POINTS

To define the point around which a PolyObj will rotate (when asked to do so), it is necessary to place a special Thing, called a *PolyObj Anchor Spot*, on the map and associate this with its particular PolyObj. Note that after it is assigned to a PolyObj, the Anchor Spot is considered to be part of that object, so it also takes part in any subsequent moves imposed on it. If you place the Anchor Spot in the center of a PolyObj, for example, that is where it will remain—at the center of the shape—no matter where subsequent movements take the PolyObj.

A PolyObj's Anchor Spot need not be placed at its center, though, nor within it, nor even close by it! The purpose of the Anchor Spot is merely to provide a single point of reference to the PolyObj when one is needed. When rotational movements are imposed on a PolyObj, for instance, the Anchor Spot provides its axis of rotation. You can therefore make a PolyObj carry out its rotations (literally) on the spot by placing its Anchor Spot at its center. Or you can make it follow the arc of a circular path by placing the Anchor Spot some distance outside of the shape. The choice is yours. You just need to ensure that there is sufficient space for the PolyObj to carry out the motions you intend to subject it to.

Each PolyObj must have one and only one Anchor Spot associated with it.

POLYOBJ START SPOTS

In addition to its Anchor Spot, each PolyObj needs to have one more special Thing associated with it: its Start Spot. To understand the need for this item, it is necessary to consider something of the game engine's use of the Binary State Partition (or nodes tree) and the implication this has on the use of walls that can be asked to move.

POLYOBJS AND THE NODES TREE

Little has been said about the nodes tree in the book. Largely, this is because this structure is not easily understood, and in any case, an understanding of it is not essential to the understanding of the building of WADs. Fortunately, it isn't necessary to delve too deeply into this structure even now.

You should be aware that the purpose of the nodes tree is to provide the graphic engine with a quick way of determining which surfaces are in view at any given time. Now, DOOM's nodes tree is a strictly two-dimensional

one. It provides DOOM with information only about which walls are hidden by other walls; no ceiling or floor information is provided. Because of this, the information it contains remains valid only as long as a wall does not alter its horizontal position. Floor and ceiling height can change with complete impunity, however. You have probably realized this fact from using a map editor. The nodes are usually rebuilt if you have moved a vertex (and, therefore, a line) but not if the only changes you have made are to sector information. Now you know why all of DOOM's doors open vertically: it saves the game engine from having to decide what to do about the fact that a horizontal wall movement will make the nodes tree invalid!

So how does Hexen (which is, after all built on the same game engine) manage? The answer is that it cheats! But it cheats very cleverly. The only problem is that it requires you to help it cheat.

Now, it so happens that there are fewer problems with nodes tree inaccuracies if walls are interposed in views that should be open than if walls that should be blocking a view are removed. In the former case, the game engine merely needs to be told that there are now some new walls in the way. The view is being restricted, not increased, so there is no extra work to do, merely work being prevented. Hexen makes use of this fact in its handling of PolyObjs.

To make sure that the nodes tree reflects the maximum visibility around PolyObjs, Hexen requires that all PolyObjs be built away from real game space. The area in which they will operate must be created in the completely empty and open state. The business of moving the PolyObj into place is then handled by the game as the level loads.

All of this no doubt makes the PolyObj sound like a very complicated structure. In reality, however, it is surprisingly easy to build, provided that you follow a few simple rules.

BUILDING A POLYOBJ

Following is a step-by-step look at the process of creating a PolyObj.

MAKING SPACE

The first step in creating a PolyObj is to provide it with some space in which to perform its functions. Normally, this simply means creating an open sector into which the PolyObj will be moved at runtime. Do not create complicated areas for your PolyObjs; straightforward rectangles and regular polygonal areas are best. You should also make sure that you build one sector for each PolyObj you use; Hexen crashes back to DOS with a More than one PolyObj in the same subsector message if you don't.

DRAWING THE LINES

After you have the spaces the PolyObjs will occupy, you can think about drawing whatever PolyObjs are needed for the task you had in mind.

Because PolyObjs are simply collections of lines, they can be drawn anywhere on the level map, provided that they are located in areas where they will not confuse the binary nodes partition when they move. Essentially,

this means that they should be kept well away from any areas into which the player can ever see. Usually, it is best to assign an area of the map well away from the actual game space.

You should draw the lines of the PolyObj in the orientation in which you want it to first appear. You need not assign these lines to any sector, though, unless you need to in order to satisfy your editor. You might find it convenient to create a sector out of the way somewhere in which you can construct all of your PolyObjs. This method can save you confusion and accidents later. Make sure that all the lines are single-sided and flagged that way. Their lines should face out from the shape; it is meant to be a solid structure, remember.

Put whatever texture you want to appear on each PolyObj line on its right main texture slot. All other textures are ignored because the line is single-sided. The final rendered height of the PolyObj is derived from the sector in which it is placed by the game at runtime. You don't need to do anything to ensure that it fits.

DEFINING THE POLYOBJ

If your PolyObj is simple in shape (as most should be), you can use the PolyObj_StartLine method of defining it. This is much simpler than the other method, and it alone is a good reason to keep PolyObjs simple in form. To use this method, just place a special action code of this type on any one of the PolyObj's lines. Set this action's first argument to the number by which you want the PolyObj to be referenced. Any value will do, in the range 1 to 255, but make sure that each PolyObj in any one map is assigned a unique number. Set the second argument to reference this PolyObj's mirror, or leave it as zero if no mirror is required. You will later need to create the mirroring PolyObj and assign it this identifying number so that the game can find it when it comes to move the first. The third argument of the PolyObj_StartLine special specifies the sound to be used to accompany movements of the PolyObj—Table 4 of "Hexen's Special Codes" on the CD-ROM has details.

More complex PolyObj shapes need to be defined with the PolyObj_ExplicitLine special instead. This is more tedious to use, but no more complicated. The special must be placed on every line of the PolyObj, with an identical first argument to identify the lines as belonging to the same PolyObj. This argument is the PolyObj's unique identifying number. The second argument specifies the rendering order of the lines; mark these in sequence from 1. The third and fourth arguments specify the mirror and associated sound.

NOTE: A full specification of the parameterization of these PolyObj specials is given in Table 13 of "Hexen's Special Codes" on the CD-ROM.

PLACING THE SPOTS

The next step is to attach the Anchor Spot to your new PolyObj. Place the Anchor Spot at the point about which you want the PolyObj to rotate when activated. If you want the PolyObj to only slide, place the Anchor Spot at some convenient location, coincident with one of its vertices, say.

Having done this, you need a Start Spot. Place this at the point on the map (in the real game space) where you want the PolyObj to appear. More specifically, it marks the point at which the Anchor Spot (with the PolyObj

attached to it) is moved by Hexen when the map loads. Don't worry about positioning these Spots so as to make your PolyObj fit exactly up to the other walls of your map, by the way. It won't matter to Hexen if your PolyObj passes into existing walls, because the nodes tree takes care of correct rendering of this—the map walls clip the PolyObj as necessary. Watch out for overlapping PolyObjs, though. Chaos results if overlapping occurs!

Two different sorts of Start Spots are available to choose from: a standard Start Spot and a crushing Start Spot. PolyObjs located with standard Start Spots stop their movement if they catch a player in them. The crushing variety pushes the player along and causes a high rate of damage if the player has nowhere to go!

ASSIGNING THE SPOTS TO THE POLYOBJ

The final step in the construction of the PolyObj is to attach the Anchor and Start Spots to the PolyObj. You achieve this task by matching the *facing angle* of both of the Spots to the identifying number of the PolyObj. To do this, you need an editor that allows you direct access to this field so that you can enter a number, rather than specify one of eight map directions. This nonstandard use of the facing angle field dates from before Raven Software had implemented Thing tagging.

ACTIVATING A POLYOBJ

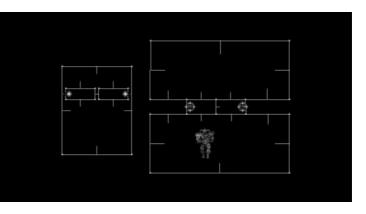
After a PolyObj has been constructed, it becomes simplicity itself to activate. All you need to do is use an activation special of an appropriate type to put it through its paces. Special codes are provided to make PolyObjs rotate, move sideways, or combine these motions. Table 14 in "Hexen's Special Codes" on the CD-ROM provides full details of these codes.



CAUTION: Hexen will happily move PolyObjs through "solid" walls, and if properly constructed, this effect will not look odd. Indeed, there are even times when you might want this effect—with doors that slide open and disappear into walls, for instance. Unlike standard DOOM doors, which are just a shrinking upper texture, PolyObjs don't change size, however. Sliding doors really do move through the walls. If there is a corridor on the other side, it might end up blocked!

Figure 1 shows the map necessary to construct two rooms separated by a pair of doors that slide open when activated from either side. The game area is the area on the right, consisting of four sectors: one for each of the rooms, and one for each of the PolyObjs that make up the doors. These have been built in the sector shown in the left side of the figure. Note the placements of the Anchor Spots (one in each PolyObj) and the Start Spots (in the game space). Special lines to activate these doors (with a PolyObj_DoorSlide action) can be placed either on each door face or on the lines marking their destination sectors. You can find this sample WAD, HEXDOOR.WAD, on the accompanying CD-ROM.





CONSTRUCTION SUMMARY

Here, in summary, are the steps involved in building a Hexen PolyObj:

- 1. Draw the empty sector the PolyObj will occupy when the game is played.
- 2. Draw the lines making up the object. Most PolyObjs will be viewed only from their "outside," so draw a complete shape, with the right sides facing out. Place this shape in unused space, well away from normal map lines.
- 3. Assign the PolyObj_StartLine code to one of the lines of your PolyObj if it is a simple convex shape. Alternatively, use PolyObj_ExplicitLine codes on all of its lines if the shape is more complex.
- 4. Set the first argument byte of the lines to which you just assigned the defining PolyObj specials to a number unique for this PolyObj. If you're using the PolyObj_ExplicitLine method, make sure that all lines have the same number here.
- 5. If using the PolyObj_ExplicitLine method, work around the lines of the PolyObj, setting each line's second special argument byte to the rendering order for that line.
- 6. Put the identifying number of the PolyObj's mirror in the appropriate special argument byte of each of the preceding defining lines.
- 7. Place an Anchor Spot at your PolyObj's center of rotation. Set its facing angle to the identifying number of the PolyObj.
- 8. Place a Start Spot of the desired type (crushing or noncrushing) where you want the PolyObj's Anchor Spot (and the PolyObj with it!) to be placed as the level loads. Set the Start Spot's facing angle to the identifying number of the PolyObj.
- 9. Finally, set up appropriate trigger events to activate your PolyObj, using the suitable PolyObj special action codes.

EXIT: MOPPING UP AND MOVING ON

This concludes the training mission for those who want to become Grand Masters in the art of Hexen editing. "Programming the Action" and this chapter have done little more than provide an introduction to the extensions of the DOOM game engine provided within the Hexen game variant. You will find more details of its advanced programming features in the book.

HEXEN'S SPECIAL CODES

This chapter takes a detailed look at the way in which the system of special action codes has been extended in the game variant Hexen. The complete range of codes available in Hexen is presented, along with details of the other codes that the designer will need to know to utilize the main actions.

CONTROLLING HEXEN'S ACTIONS

The designers of Hexen have seen fit to change completely the way in which events are triggered and controlled within the game. These changes give the designer unprecedented control over the way events are handled by the game engine. No longer are triggering mechanisms hard-coded into the action type; individual triggerings can now be controlled by means of a line's attribute flags. In addition, Hexen's special actions can be triggered in other ways, not just from lines, as was the case in earlier variants of the game engine. And finally, as well as the extended triggering mechanism, Hexen also provides the designer with the ability to parameterize each available action. Up to five numerical values can be passed to an action each time it is triggered. Each of Hexen's special actions uses these parameters in particular ways to control certain aspects of their operation.

SPECIAL ACTION PARAMETERS

Many of the actions have parameters in common, so it is helpful to look first at the more general parameters that can be used.

TAGS AND TIDS

As in DOOM, many special action codes rely on tags to provide them with the target of their attentions. Sectors are tagged via their tag field, just as they are in DOOM. Lines no longer have a specific tag, though; their Special Attribute has been extended to allow a tag to be specified as one of the five arguments passed to the line's special action. Another change in Hexen is that Things can now be tagged. This task is achieved through a new Thing Attribute, called either their tag or their Thing Identification (tid, for short) field. A Thing's tag (or tid) acts just like a sector's tag: it marks the Thing as a target of an action. As with DOOM, if an action requires a tag to be specified, and a value of zero is used, most actions will apply their effect to all untagged targets, with the usual spectacular results!

NOTE: The alterations to the way special actions are specified and controlled in Hexen is bound up inextricably with that game's new scripting capabilities. It is therefore difficult to describe one of these mechanisms without constant reference to the other. If you would like to know more about Hexen's scripting language before becoming involved in its special codes, read "Hexen's Scripting Language" on the CD-ROM before reading this chapter.

TICS

Some actions can be made to occur for specific durations. Hexen uses two principal units of time: the second, and a division of the second called the *tic*. There are 35 tics to the second. (Don't ask why!) Some special actions are parameterized in seconds; others, in tics. Occasionally, a unit of eight tics (the*octic*) is also used. Details are given in the tables that follow.

BYTE ANGLES

Some special actions need to be given a map direction (or *vector*) along which to act. This angle usually needs to be specified more accurately than the coarse settings used to provide Things with their map facings. A new unit of measurement, called a *byte angle*, is therefore used. This system divides a circle into 256 "degrees," working counterclockwise from the east. Under this scheme, therefore, the main directions are as given in Table 1.

Direction	Byte Angle
East	0
Northeast	32
North	64
Northwest	96

Table 1. Principal byte angle directions.

Direction	Byte Angle	
West	128	
Southwest	160	
South	192	
Southeast	224	

KEYS

Some actions require a player to have acquired a particular key before they will operate. Instead of matching keys to the action codes, as in DOOM, Hexen allows the key to be specified as a parameter. The codes for each key present in the game are given in Table 2.

Table 2. Hexen's key codes.

Key Code	Туре оf Кеу
1	Steel key
2	Cave key
3	Axe key
4	Fire key
5	Emerald key
6	Dungeon key
7	Silver key
8	Rusted key
9	Horn key
10	Swamp key
11	Castle key

PUZZLE ITEMS

In addition to keys, Hexen requires players to acquire certain puzzle pieces. These are also coded for use as parameters to certain special actions. These codes are given in Table 3. This table also lists the predefined names of these items that can be used in ACS scripts.

Table 3. Hexen puzzle-piece codes.

Code	ACS Identifier	Item
0	ZZ_Skull	Yorick's skull
1	ZZ_BigGem	Heart of D'Sparil
2	ZZ_GemRed	Ruby planet
3	ZZ_GemGreen1	Emerald planet 1
4	ZZ_GemGreen2	Emerald planet 2
5	ZZ_GemBlue1	Sapphire planet 1
6	ZZ_GemBlue2	Sapphire planet 2
7	ZZ_Book1	Daemon codex
8	ZZ_Book2	Liber oscura
9	ZZ_Skull2	Flame mask
10	ZZ_FWeapon	Glaive seal
11	ZZ_CWeapon	Holy relic
12	ZZ_MWeapon	Sigil of the Magus
13	ZZ_Gear	Clock gear
14	ZZ_Gear2	Bronze clock gear
15	ZZ_Gear3	Clock gear with bronze hub
16	ZZ_Gear4	Clock gear with bronze ring

SOUNDS

A few special actions can have the sounds associated with them specified via their parameters also. The codes for the various sounds available are given in Table 4.

Table 4.	Hexen's	special	sound	codes.
----------	---------	---------	-------	--------

Code	Sound Name	Description
1	Heavy	Heavy stone moving
2	Metal	Metallic slam
3	Creak	Hinges creaking
4	Silence	Silence
5	Lava	Lava flowing
6	Water	Water bubbling

Code	Sound Name	Description
7	Ice	Ice creaking
8	Earth	Earth rumbling
9	Metal2	Metallic clang

TIP: Most actions have a sound associated with them that is determined by the game engine during play. Usually, this sound is determined by the combination of the type of action and the speed of its operation. If you don't like the type of sound that accompanies the action you are using, try varying the speed a little to see whether you can make Hexen choose another!

NOTE: All Hexen's sounds can also be invoked through its scripting language. See "Hexen's Scripting Language" on the CD-ROM for details.

THING SPAWNING CODES

In addition to the standard Thing identification codes used when laying out a map (which are given in Chapter 34, "Essential Thing Information"), Hexen also has another set of codes for spawnable Things. These Things can be spawned by the appropriate spawning actions (detailed later in this chapter). Table 5 lists these codes for spawned objects and also gives their ACS identifier codes.

	9	
Spawn Code	ACS Identifier	Description
1	T_CENTAUR	Centaur
2	T_CENTAURLEADER	Centaur Leader
3	T_DEMON	Gas Chaos Serpent
4	T_ETTIN	Ettin
5	T_FIREGARGOYLE	Afrit
6	T_WATERLURKER	Stalker
7	T_WATERLURKERLEADER	Stalker Leader
8	T_WRAITH	Reiver
9	T_WRAITHBURIED	Reiver Leader
10	T_FIREBALL1	Fireball

Table 5. Hexen's spawnable Things.

Tal	ble	5.	cont	inued

Spawn Code	ACS Identifier	Description
11	T_MANA1	Blue Mana
12	T_MANA2	Green Mana
13	T_ITEMBOOTS	Boots of Speed
14	T_ITEMEGG	Porkelator
15	T_ITEMFLIGHT	Wings of Wrath
16	T_ITEMSUMMON	Dark Servant
17	T_ITEMTPORTOTHER	Banishment Device
18	T_ITEMTELEPORT	Disc of Repulsion
19	T_BISHOP	Dark Bishop
20	T_ICEGOLEM	Wendigo
21	T_BRIDGE	Glitter Bridge
22	T_DRAGONSKINBRACERS	Dragonskin Bracers
23	T_ITEMHEALTHPOTION	Health Vial
24	T_ITEMHEALTHFLASK	Quartz Flask
25	T_ITEMHEALTHFULL	Mystic Urn
26	T_ITEMBOOSTMANA	Krater of Might
27	T_FIGHTERAXE	Timon's Axe
28	T_FIGHTERHAMMER	Hammer
29	T_FIGHTERSWORD1	Quietus Piece 1
30	T_FIGHTERSWORD2	Quietus Piece 2
31	T_FIGHTERSWORD3	Quietus Piece 3
32	T_CLERICSTAFF	Serpent Staff
33	T_CLERICHOLY1	Wraithverge Piece 1
34	T_CLERICHOLY2	Wraithverge Piece 2
35	T_CLERICHOLY3	Wraithverge Piece 3
36	T_MAGESHARDS	Frost Shards
37	T_MAGESTAFF1	Bloodscourge Piece 1
38	T_MAGESTAFF2	Bloodscourge Piece 2
39	T_MAGESTAFF3	Bloodscourge Piece 3
40	T_MORPHBLAST	Blast from Porkelator
41	T_ROCK1	Rock

Spawn Code	ACS Identifier	Description
42	T_ROCK2	Rock
43	T_ROCK3	Rock
44	T_DIRT1	Rubble
45	T_DIRT2	Rubble
46	T_DIRT3	Rubble
47	T_DIRT4	Rubble
48	T_DIRT5	Rubble
49	T_DIRT6	Rubble
50	T_ARROW	Arrow
51	T_DART	Dart
52	T_POISONDART	Poison Dart
53	T_RIPPERBALL	Ripperball
54	T_STAINEDGLASS1	Glass Shard
55	T_STAINEDGLASS2	Glass Shard
56	T_STAINEDGLASS3	Glass Shard
57	T_STAINEDGLASS4	Glass Shard
58	T_STAINEDGLASS5	Glass Shard
59	T_STAINEDGLASS6	Glass Shard
60	T_STAINEDGLASS7	Glass Shard
61	T_STAINEDGLASS8	Glass Shard
62	T_STAINEDGLASS9	Glass Shard
63	T_STAINEDGLASS0	Glass Shard
64	T_BLADE	Barbed Blade
65	T_ICESHARD	Ice Shard
66	T_FLAME_SMALL	Small Flame
67	T_FLAME_LARGE	Large Flame
68	T_MESHARMOR	Mesh Armor
69	T_FALCONSHIELD	Falcon Shield
70	T_PLATINUMHELM	Platinum Helm
71	T_AMULETOFWARDING	Amulet of Warding
72	T_ITEMFLECHETTE	Fletchette
73	T_ITEMTORCH	Torch

Tab	le 5.	continued

Spawn Code	ACS Identifier	Description
74	T_ITEMREPULSION	Chaos Device
75	T_MANA3	Combined Mana
76	T_PUZZSKULL	Yorrick's Skull
77	T_PUZZGEMBIG	Heart of D'Sparil
78	T_PUZZGEMRED	Ruby Planet
79	T_PUZZGEMGREEN1	Emerald Planet 1
80	T_PUZZGEMGREEN2	Emerald Planet 2
81	T_PUZZGEMBLUE1	Sapphire Planet 1
82	T_PUZZGEMBLUE2	Sapphire Planet 2
83	T_PUZZB00K1	Daemon Codex
84	T_PUZZB00K2	Liber Obscura
85	T_METALKEY	Steel Key
86	T_SMALLMETALKEY	Cave Key
87	T_AXEKEY	Axe Key
88	T_FIREKEY	Fire Key
89	T_GREENKEY	Emerald Key
90	T_MACEKEY	Dungeon Key
91	T_SILVERKEY	Silver Key
92	T_RUSTYKEY	Rusted Key
93	T_HORNKEY	Horn Key
94	T_SERPENTKEY	Swamp Key
95	T_WATERDRIP	Drip of Water
96	T_TEMPSMALLFLAME	Temporary Small Flame (very short-lived!)
97	T_PERMSMALLFLAME	Small Flame
98	T_TEMPLARGEFLAME	Temporary Large Flame (very short-lived!)
99	T_PERMLARGEFLAME	Large Flame
100	T_DEMON_MASH	Ghost Gas Chaos Serpent
101	T_DEMON2_MASH	Ghost Fire Chaos Serpent
102	T_ETTIN_MASH	Ghost Ettin
103	T_CENTAUR_MASH	Ghost Centaur
104	T_THRUSTSPIKEUP	Rising Spike

Spawn Code	ACS Identifier	Description	
105	T_THRUSTSPIKEDOWN	Hidden Spike	
106	T_FLESH_DRIP1	Splash of Flesh 1	
107	T_FLESH_DRIP2	Splash of Flesh 2	
108	T_SPARK_DRIP	Spark	

CAUTION: Any attempt to spawn items other than those listed in Table 5 will crash the game engine.



HEXEN'S SPECIAL ACTIONS

For convenience, the list of special actions available in Hexen is divided here into various categories:

- Door actions
- Other ceiling actions
- Moving platform actions
- Simple floor actions
- Combined floor and ceiling actions
- Stair-building actions
- Lighting actions
- Thing actions
- PolyObject actions
- Script actions
- Teleporting actions
- Various additional Hexen-specific effects

In the sections that follow, each of these categories is examined in detail. Tables provide a full list of the parameters taken by each action as a series of arguments (Arg1 to Arg5). Also given are the names of the function to call in an ACS script to initiate each action.

DOOR ACTIONS

Details of Hexen's door actions are given in Table 6. Apart from the way they are controlled through their parameters (and the added flexibility that ensues), they all operate in the same way as DOOM's doors.

Table 6. Hexen's door actions.

Code	Function Name	Arg1	Arg2	Arg3	Arg4	Arg5	Action
10	Door_Close	tag	speed	_	_	_	Close door
11	Door_Open	tag	speed	_	_	_	Open door
12	Door_Raise	tag	speed	delay	-	-	Open, pause, close door
13	Door_LockedRaise	tag	speed	delay	key	-	Open, pause, close (keyed) door

These are the arguments used to control these actions:

tag	Specifies which sectors should respond. If this value is nonzero, sectors with this value in their tag field carry out the specified action. If this argument is zero, the sector on the left side of the triggering line responds, as with standard DOOM doors.
speed	Specifies the speed at which the door opens. If this value is zero, the door never opens. A value of 1 creates a door that moves at about 4 pixels per second. A value of 255 produces a door that moves rather faster than one of DOOM's turbo doors.
delay	Specifies the amount of time (in tics) that the door remains open. If a zero delay is specified, the door remains open until closed by some other action.
key	Specifies which key is required to open the door. The key codes are given in Table 2.

CEILING-MOVING ACTIONS

Hexen's other special codes that carry out actions just on ceilings are given in Table 7.

Table 7. Hexen's codes for moving ceilings.

Code	Function Name	Arg1	Arg2	Arg3	Arg4	Arg5	Action
40	Ceiling_LowerByValue	tag	speed	distance	-	-	Lower ceiling by specified amount
41	Ceiling_RaiseByValue	tag	speed	distance	-	-	Raise ceiling by specified amount
42	Ceiling_CrushAndRaise	tag	speed	damage	-	-	Start perpetual crushing ceiling(s)
43	Ceiling_LowerAndCrush	tag	speed	damage	-	-	Lower ceiling to floor with crush

Code	Function Name	Arg1	Arg2	Arg3	Arg4	Arg5	Action
44	Ceiling_CrushStop	tag	-	-	-	-	Pause a crushing ceiling
45	Ceiling_CrushRaiseAndStay	tag	speed	damage	-	_	Perform one cycle of lower, crush, and raise
69	Ceiling_MoveToValueTimes8	tag	speed	height	neg	_	Move ceiling to new height (neg \times $-1 \times height \times 8$)

The arguments used to control these actions are similar to those for doors, except for the following arguments:

tag	Specifies which sectors should respond. Sectors with this value in their tag field carry out the specified action. A value of zero should <i>not</i> be used here.
distance	Specifies the vertical distance to move (in pixels).
neg	Allows negative floor heights to be specified by use of a value of 1.
damage	Specifies the amount of damage to apply at each crush.

MOVING PLATFORM ACTIONS

Hexen's lifts and moving platforms offer a wider range of options than do DOOM's, but they operate much the same way. The codes are given in Table 8.

Code	Function Name	Arg1	Arg2	Arg3	Arg4	Arg5	Action
60	Plat_PerpetualRaise	tag	speed	delay	-	-	Start a perpetual down, wait, up, wait action
61	Plat_Stop	tag	-	-	-	_	Pause a perpetual moving platform
62	Plat_DownWaitUpStay	tag	speed	delay	-	-	Perform lift action (down to lowest adjacent, wait, up, stay)

continues

Table 8. continued

Code	Function Name	Arg1	Arg2	Arg3	Arg4	Arg5	Action
63	Plat_DownByValue	tag	speed	delay	distance	_	Perform lift action (down specified amount, wait, up, stay)
64	Plat_UpWaitDownStay	tag	speed	delay	-	_	Perform lift action (up to highest adjacent, wait, down, stay)
65	Plat_UpByValue	tag	speed	delay	distance	_	Perform lift action (up specified amount, wait, down, stay)

The arguments used with these actions are as given for the preceding category.

FLOOR-MOVING ACTIONS

Hexen's special codes for moving floors in single, one-off movements are given in Table 9.

Table 9. Hexen's codes for moving floors.

Code	Function Name	Arg1	Arg2	Arg3	Arg4	Arg5	Action
20	Floor_LowerByValue	tag	speed	distance	-	_	Lower floor by specified amount
21	Floor_LowerToLowest	tag	speed	-	-	-	Lower floor to level of lowest adjacent
22	Floor_LowerToNearest	tag	speed	-	-	_	Lower floor to level of next lower adjacent
23	Floor_RaiseByValue	tag	speed	distance	-	-	Raise floor by specified amount
24	Floor_RaiseToHighest	tag	speed	-	-	-	Raise floor to level of highest adjacent
25	Floor_RaiseToNearest	tag	speed	-	-	_	Raise floor to level of next higher adjacent
28	Floor_RaiseAndCrush	tag	speed	damage	-	_	Raise floor to ceiling with crush

Code	Function Name	Arg1	Arg2	Arg3	Arg4	Arg5	Action
35	Floor_RaiseByValueTimes8	tag	speed	distance	-	-	Raise floor by $8 \times$ distance units
36	Floor_LowerByValueTimes8	tag	speed	distance	-	_	Lower floor by $8 \times distance$ units
46	Floor_CrushStop	tag	-	-	-	-	Stop a crushing floor
66	Floor_LowerInstant	tag	_	distance	-	-	Lower floor by specified distance instantaneously
67	Floor_RaiseInstant	tag	-	distance	-	-	Raise floor by specified distance instantaneously
68	Floor_MoveToValueTimes8	tag	speed	height	neg	-	Move ceiling to new height $(neg \times -1 \times neight \times 8)$

Again, the parameterization of these actions is as given for moving ceilings.

_

SIMULTANEOUS FLOOR AND CEILING ACTIONS

Hexen has additional special codes to move a floor and ceiling simultaneously. These are listed in Table 10.

Table 10.	Hexen's co	des for s	simultaneous	floor and	ceiling n	novements.

Code	Function Name	Arg1	Arg2	Arg3	Arg4	Arg5	Action
95	FloorAndCeiling_LowerByValue	tag	speed	distance	_	_	Lower both floor and ceiling by specified amount
96	FloorAndCeiling_RaiseByValue	tag	speed	distance	_	-	Raise both floor and ceiling by specified amount
							continues

Code	Function Name	Arg1	Arg2	Arg3	Arg4	Arg5	Action
29	Pillar_Build	tag	speed	distance	-	_	Raise floor, lower ceiling to specified distance above starting floor
30	Pillar_Open	tag	speed	f_dist	c_dist	_	Raise ceiling and lower floor by specified amounts
94	Pillar_BuildAndCrush	tag	speed	distance	damage	-	Act as Pillar_Build but with crush

Parameterization is as given previously, except for these parameters:

 f_{dist} Specifies the vertical distance to move the floor.

c_dist Specifies the vertical distance to move the ceiling.

STAIR-BUILDING ACTIONS

The sector arrangement required for self-building stairs in Hexen is much simpler than in earlier variants of the game. The stair-building specials all operate by first finding the sector with a tag that matches the first argument of the action, then creating stairs by traversing adjacent sectors marked with the stairSpecial1 and stairSpecial2 special characteristic (see Chapter 35, "Special Sector Types," for the values). These specials must alternate between the two, and the sector chain must not branch. This is exactly like using Hexen's Pulsed Lighting effect discussed in Chapter 13, "Special Sectors."

Table 11 gives the codes for these actions in Hexen.

Code	Function Name	Arg1	Arg2	Arg3	Arg4	Arg5	Action
26	Stairs_BuildDown	tag	speed	height	delay	reset	Build stairs leading down
27	Stairs_BuildUp	tag	speed	height	delay	reset	Build stairs leading up

Table 11. Hexen's codes for self-building stairs.

Code	Function Name	Arg1	Arg2	Arg3	Arg4	Arg5	Action
31	Stairs_BuildDownSync	tag	speed	height	reset	-	Build stairs leading down, all steps moving at once
 32	Stairs_BuildUpSync	tag	speed	height	reset	-	Build stairs leading up, all steps moving at once

These are the new arguments used to control these actions:

height	Specifies the vertical height of each step.
reset	Specifies how long (in tics) the staircase remains standing. If this value is nonzero, the
	entire staircase reverts to its original configuration after this time. If a zero is specified
	here, the staircase does not reset.

LIGHTING ACTIONS

Hexen's special actions that act on sector lighting levels are detailed in Table 12.

Table 12.	Hexen's	codes f	for	producing	a liahtina	g changes.

Code	Function Name	Arg1	Arg2	Arg3	Arg4	Arg5	Action
109	Light_ForceLightning	tag	-	-	_	-	Trigger a lightning flash and accompanying thunder-crash
110	Light_RaiseByValue	tag	change	-	-	_	Increase lighting level by specified amount
111	Light_LowerByValue	tag	change	_	-	_	Reduce lighting level by specified amount
112	Light_ChangeToValue	tag	value	-	-	-	Change lighting to specified level
113	Light_Fade	tag	value	tics	_	-	Change lighting to specified level over specified period
114	Light_Glow	tag	upper	lower	tics	-	Cycle lighting between specified levels over specified period

Code	Function Name	Arg1	Arg2	Arg3	Arg4	Arg5	Action
115	Light_Flicker	tag	upper	lower	-	-	Flicker lighting at random between specified levels
116	Light_Strobe	tag	upper	lower	u-tics	l-tics	Switch lighting between two levels, spending specified times at each

These are the arguments used to control these actions:

tag	Specifies which sectors respond to the action.
value	Specifies a new lighting level.
change	Specifies the amount by which the lighting is to change.
lower	Specifies the lower level of lighting.
upper	Specifies the upper level of lighting.
tics	Specifies the time to be taken to perform the change.
u-tics	Specifies the time to be spent at the upper lighting level.
l-tics	Specifies the time to be spent at the lower lighting level.

THING ACTIONS

In Hexen, actions can now be made to happen to Things. Many of these actions can be initiated in the normal way, although some can be activated only by the Things themselves. Details are given in Table 13.

Table 13. Hexen's codes for	actions affecting Things.
-----------------------------	---------------------------

Code	Function Name	Arg1	Arg2	Arg3	Arg4	Arg5	Action
72†	ThrustThing	angle	distance	-	-	-	Push the Thing the specified distance in the specified direction
73†	DamageThing	damage	-	_	-	-	Do the specified amount of damage to the Thing
130	Thing_Activate	tid	-	-	-	-	Activate the specified Thing(s)

17

Code	Function Name	Arg1	Arg2	Arg3	Arg4	Arg5	Action
131	Thing_Deactivate	tid	-	-	-	-	Deactivate the specified Thing(s)
132	Thing_Remove	tid	_	-	-	-	Remove the specified Thing(s) from the map
133	Thing_Destroy	tid	-	-	_	-	Kill the specified Thing(s)
134	Thing_Projectile	tid	type	angle	speed	vspeed	Spawn projectile of specified type, moving along given vector at the specified speeds
136	Thing_ProjectileGravity	tid	type	angle	speed	vspeed	Spawn projectile moving along given vector and subject to gravity
135	Thing_Spawn	tid	type	angle	-	_	Spawn a Thing of specified type, facing given direction
137	Thing_SpawnNoFog	tid	type	angle	-	-	Spawn a Thing without fizz and glitter
Special	codes marked † have meaning	only when	used as a spec	cial actior	ı triggered	from a Thiı	ng.

These are the arguments used to control actions aimed at Things:

_

tid	Specifies which Thing(s) should respond to the action. For the spawning actions, this value specifies which of the map's Spawn Spots should be used to determine the starting location of the new Thing(s).
angle	Specifies the direction in which the action is vectored. This is a Hexen byte angle, not a normal DOOM map angle.
speed	Specifies a spawned Thing's initial horizontal speed.
vspeed	Specifies a Thing's initial vertical speed.
type	Specifies the type of Thing to spawn.

NOTE: Only Things capable of being activated respond to the Thing_Activate and Thing_Deactivate codes. Only Things of spawnable type can be made to appear with the spawning actions. The codes used must be the special spawning codes presented previously in Table 5—not the normal Thing identification codes used when laying out a map.

POLYOBJ ACTIONS

PolyObj special actions fall into two general categories:

- Line specials, used to define a PolyObj to the game engine. (See Table 14.)
- True PolyObj special action codes, used to trigger the operation of up to a pair of PolyObj's. (See Table 15.)

Table 14. Hexen's codes for defining PolyObjs.

Code	Function Name	Arg1	Arg2	Arg3	Arg4	Arg5	Action
1	Polyobj_StartLine	ро	mirror	sound	-	_	Define the start line of the specified PolyObj and its mirror
5	Polyobj_ExplicitLine	ро	order	mirror	sound	_	Define the rendering order of lines within the specified PolyObj pair

Details of the use of these special codes are given in "Making Moving Scenery" on the CD-ROM.

Table 15. Hexen's codes for controlling and activating PolyObjs.

Code	Function Name	Arg1	Arg2	Arg3	Arg4	Arg5	Action
2	Polyobj_RotateLeft	ро	speed	angle	_	_	Rotate the PolyObj the specified distance counter- clockwise
3	Polyobj_RotateRight	ро	speed	angle	_	-	Rotate the PolyObj the specified distance clockwise
4	Polyobj_Move	ро	speed	direction	distance	_	Move the PolyObj the specified distance in the given direction

Code Function Name Arg1 Arg2 Arg3 Arg4 Arg5 Action 6 Move the PolyObj 8 Polyobj MoveTimes8 ро speed direction distance times the specified distance in the given direction 7 Rotate the object Polyobj_DoorSwing ро speed angle delay counterclockwise and then back 8 Move the object Polyobj_DoorSlide ро angle distance delay speed along the specified vector and back again 90 Add specified Polyobj OR RotateLeft ро speed angle counterclockwise rotation to the PolyObj's current motion 91 Add specified Polyobj_OR_RotateRight po speed angle clockwise rotation to the PolyObj's current motion 92 Add specified Polyobj OR Move distance ро speed angle _ movement along a vector to PolyObj's current motion 93 Add 8 times the Polyobj_OR_MoveTimes8 ро angle distance speed specified movement along a vector to PolyObj's current motion

These are the arguments used to control these actions:

ро	Specifies which PolyObj should respond to the action.
angle	Specifies the angular change that the action acts through. This is a Hexen byte angle, not a normal DOOM map angle.
direction	Specifies the vector along which the action is directed. This is a Hexen byte angle, not a normal DOOM map angle.
distance	Specifies the number of map units to move the PolyObj.

19

SCRIPT ACTIONS

As well as using scripts to control sectors, script execution can itself be controlled from special triggers. The codes for controlling scripts are presented in Table 16.

Code	Function Name	Arg1	Arg2	Arg3	Arg4	Arg5	Action
80	ACS_Execute	script	тар	s_arg1	s_arg2	s_arg3	Execute the specified script, passing up to three arguments
81	ACS_Suspend	script	тар	-	_	-	Suspend execution of the specified script
82	ACS_Terminate	script	тар	-	-	-	Terminate execution of the specified script
83	ACS_LockedExecute	script	тар	s_arg1	s_arg2	key	Execute a specified script, provided that the key is present

Table 16. Hexen's codes for controlling scripts.

These are the arguments used to control these actions:

script	Specifies which script should respond.
тар	Specifies the map that contains the script.
s_argn	The arguments to be passed to the script.
key	Specifies which key must be in the player's inventory before the script will respond.
delay	Specifies the number of tics before the action reverses.

TELEPORTING ACTIONS

Hexen's teleporters are rather more powerful than those of earlier variants of the game, as can be seen from Table 17.

Code	Function Name	Arg1	Arg2	Arg3	Arg4	Arg5	Action
70	Teleport	tid	-	_	_	-	Teleport triggering object to specified map spot
71	Teleport_NoFog	tid	-	-	-	-	Teleport triggering object silently, and without fog

Table 17. Hexen's codes for producing teleports.

Code	Function Name	Arg1	Arg2 Arg3	Arg4	Arg5	Action
74	Teleport_NewMap	тар	player —	-	-	Teleport player to appropri- ate start spot on new map
75	Teleport_EndGame	-		-	-	End game and run finale script

These are the arguments used to control these actions:

tid	Specifies the destination map spot.
тар	Specifies the destination map.
player	Specifies which player start spot is the destination.

MISCELLANEOUS LINE ACTIONS

As with DOOM, Hexen uses some codes that can act only on lines. Table 18 lists these codes.

Table 18. Hexen's special line codes.

Code	Function Name	Arg1	Arg2	Arg3	Arg4	Arg5	Action
121	Line_SetIdentification	line	_	-	_	_	Identify the line to script functions that require line IDs
100	<pre>Scroll_Texture_Left</pre>	speed	-	-	-	-	Scroll the line's texture(s) to the left
101	Scroll_Texture_Right	speed	-	-	-	-	Scroll the line's texture(s) to the right
102	Scroll_Texture_Up	speed	-	-	-	-	Scroll the line's texture(s) upward
103	<pre>Scroll_Texture_Down</pre>	speed	-	-	-	-	Scroll the line's texture(s) downward

These are the arguments used in conjunction with these actions:

line Specifies the unique identifying number of the line.

speed Specifies the speed of scroll in pixels per tic.

MISCELLANEOUS HEXEN-SPECIFIC ACTIONS

The final category of actions are those that are highly specific to the game of Hexen. Two of these involve more moving scenery. The floor waggling action moves a specified floor up and down over a specified range. The action appears damped: it takes a time to work up to its full waggle range, and it fades out gradually afterward. The earthquake action, on the other hand, is sudden and does high damage to a player within its damage radius. Details of these and other Hexen-specific codes are given in Table 19.

Code	Function Name	Arg1	Arg2	Arg3	Arg4	Arg5	Action
129	UsePuzzleItem	item	script	s_arg1	s_arg2	s_arg3	Activate a script on use of the specified puzzle item
140	Sector_ChangeSound	tag	sound	-	-	-	Change the sound associated with a sector
138	Floor_Waggle	tag	range	speed	offset	duration	Waggle the floor of tagged sector(s) over the specified height range
120	Earthquake	intensity	tics	damrad	tremrad	tid	Create an earth- quake at all matching foci

Table 19. Miscellaneous codes for Hexen-specific actions.

These are the new arguments here:

item	Specifies the puzzle item from the list of puzzle codes. These codes are detailed in Table 3.
intensity	Specifies the strength of an earthquake in the range 1 (weak) to 9 (strong).
tics	Specifies duration of the event in tics.
tid	Specifies the Things that are to be used as the foci of each quake (a minimum of two seems to be needed).
damrad	Specifies radius of damage in 64×64 cells.
tremrad	Specifies radius of tremor in 64×64 cells.
range	Specifies the operating range of the floor waggle when at maximum amplitude. The starting floor height acts as the midpoint of this range.

offset	Allows phasing offsets to be applied in the event of there being multiple floor
	waggles going on at once.
duration	Specifies duration of the event in seconds. Use zero for perpetual motion.

Using Hexen's Puzzle Pieces

In order to serve any purpose in a level, Hexen's puzzle pieces need to be associated with particular locations. This is achieved by placing a UsePuzzlePiece special action code on the line(s) where a particular puzzle piece is intended to be active. The action is keyed to its particular puzzle piece (identified by a value of 0 to 16, as shown in the descriptions of the pieces in Table 3) by means of its first argument. The action's second argument specifies the number of the ACS script to execute if the action is triggered when the specified puzzle piece is in the player's inventory. The remaining three arguments can be used to pass parameters to the script being activated.

You should note that only special actions triggered by Player Uses or Player Pushes events can be used reliably with the UsePuzzlePiece special action. The action is only ever capable of being triggered once, regardless of the state of the triggering line's Repeatable flag.

EXIT: MOPPING UP AND MOVING ON

In this chapter, full details of the Special Action Codes available within Hexen have been presented. "Hexen's Scripting Language" on the CD-ROM continues the listing of Hexen-specific information with complete details of that game's scripting language.

HEXEN'S SCRIPTING LANGUAGE

In this chapter, we offer a quick tour of Hexen's scripting language, seen by many as the most significant enhancement to the DOOM engine provided by that game variant.

There is insufficient space in this book to do more than provide you with a cursory glance at the capabilities of Hexen's scripting language. All the language's main elements are presented here, but we had to assume that you have at least a passing acquaintance with the main concepts of high level programming languages. If you find the notion of variables and their scope, for instance, something of a mystery, you might care to review a beginner's text on the C programming language (on which Hexen's scripting language is modeled).

Sadly, there is also insufficient space to show you many examples of the wonderful effects that can be achieved using scripts in Hexen. The best way to discover many of these is to examine the scripts incorporated into the main Hexen IWAD by Raven Software's designers. You can do this using a tool such as Luc Cluitmans' De-ACC Script Decompiler or the WadAuthor editor if you have Windows. You will find both of these tools on the *Alchemy* CD-ROM that accompanies this book.

ACTION CODE SCRIPT

The Hexen Script Language is called the Action Code Script, or ACS. Each map of a Hexen WAD needs to have a series of ACS scripts associated with it to achieve the special effects specific to that map. The script can either be stored separately in its own ACS file or can be contained within the WAD in the script resource. Before Hexen can use the scripts,

– By Steve Benner and Ben Morris however, you have to use a special compiler (ACC.EXE) to turn the text of the scripts into executable pseudocode, which is then bound to the WAD through its *behavior* resource, always the last lump of a Hexen PWAD.

NOTE: The script resource within a WAD file is an unofficial WAD resource, not supported by Raven or id Software. This particular resource was proposed and adopted by the WAD-building community to make the distribution of the text of ACS scripts easier.

To find out how to incorporate ACS scripts in your own Hexen WADs, consult the help files and other documentation that came with your game editor.

IDENTIFYING AND USING SCRIPTS

Individual scripts within a map are identified by a number in the range 0 to 255. Individual scripts may be activated in a number of ways. Firstly, they can be started by the special action ACS_Execute. Alternatively, a script can be declared as OPEN, which causes it to start executing automatically as soon as a player enters the map that contains that script. Thirdly, a script can be started by another script by means of the language's ACS_Execute() function. All active scripts run concurrently until they terminate or are suspended. A script started with the ACS_Execute() function will run concurrently with its parent (and all other active scripts), which will not pause to wait for its child script to conclude.

WRITING SCRIPTS

ACS is very similar to C (but rather simpler) so if you are familiar with that programming language you should have no difficulty mastering Hexen scripts. Let's take a closer look at the elements of the ACS language.

COMMENTS

Comments are ignored by the script compiler. Two forms are available—the multi-line (block) comment and the single line comment:

```
/*
   This is an example of a block comment.
   It can occupy several lines.
*/
int a; // And this is a comment on the end of a line of code
```

DIRECTIVES

ACS supports two directives: #include and #define. The following sections explain their use.

#INCLUDE

The #include directive tells the compiler to include the specified file in the compilation.

#include "filename"

Map scripts should generally start with the line

```
#include "common.acs"
```

to bind in the predefined names and variables supplied by Raven Software and included with their ACS compiler, ACC.EXE. This allows you to use functions by means of their names (given in "Hexen's Special Codes" on the CD-ROM) rather than their code numbers. Once this line has been incorporated into a script, all of Hexen's special actions can be invoked by the script simply by calling them as functions, with the requisite number of arguments supplied in parentheses. For instance, the following lines will cause a door to open, pause, then close in one sector, while a moderate floor-waggle is produced in another for 10 seconds.

Door_Raise(20,128,128); Floor_Waggle(99,30,64,0,10);

The standard include file COMMON.ACS also defines a group of world variables that need to be included by all maps to ensure consistent indexing.

#DEFINE

The #define directive allows you to use an identifier to supply a constant expression in subsequent lines of ACS. This can make your later lines of script easier to read and easier to fine tune.

```
#define identifier constant-expression
```

The *constant-expression* can be any decimal, hexadecimal (prefixed 0x) or fixed-point number. ACS also supports *<radix>_<digits>* notation for people who like working in binary, octal, or other strange bases!

VARIABLES AND THEIR SCOPE

There is only one data type in ACS—a 4 byte integer. To declare a variable of this type, use the keyword int. You may also use the keyword str to indicate that you'll be using the variable as a string.

Here are some sample variable declarations:

```
str astring, anotherstr;
int myint;
```

NOTE: The int and str keywords don't assign any value to the variables—they merely declare them for later use.



A variable's scope is one of the following:

- World-scope: global variables, holding their value from map to map. World-scope variables are declared by using the keyword world before the declaration. Each global variable must be assigned an identifying number—see the example that follows shortly.
- **Map-scope:** variables which can be accessed only within the map in which they are declared. Map-scope is given to any variable declared outside of a script.
- **Script-scope:** variables declared within a script, which are confined to use within that script.

Listing 1 shows how three such variables might be declared.

Listing 1. Sample ACS variable declarations of various scopes.

```
world int 2:WorldTimer; //declare global variable 2, accessed through the name Timer
int LocalTimer; //declare another variable local to the current map
script 1 (void)
{
    int x, y; //declare variables x and y known only to script 1
    ...
}
```

NOTE: There is a limit of 128 strings permitted in any one map's scripts.

LANGUAGE KEYWORDS

The following is a complete list of ACS keywords. You must avoid these words when choosing names for your variables.

break	case	const
continue	default	define
do	else	goto
if	include	int
open	print	printbold
restart	script	special
str	suspend	switch
terminate	until	void
while	world	

SCRIPT DEFINITIONS

To define a script, use one of two script definition forms:

```
script constant-expression ( arglist ) { statement; ...}
```

or

```
script constant-expression OPEN { statement; ...}
```

The *arglist* is a list of arguments to be passed to the script, or the keyword void if none are used. The keyword OPEN indicates that the script is to be executed at the start of a map; no arguments can be passed to such scripts.

STATEMENTS

ACS supports several types of statements. Statements all end with a semicolon (;).

COMPOUND STATEMENTS

ACS allows compound statements with pairs of braces { } just like C.

```
{
   statement1;
   statement2;
   statement3;
}
```

DECLARATION STATEMENTS

Declaration statements declare variables and define their scope, as you have already seen.

ASSIGNMENT STATEMENTS

Assignment statements assign expressions to variables. ACS supports assignment operators that allow you to modify the variable by the assignment expression, rather than complete replacement. For example, the following line assigns the value 500 to the variable A:

```
A=500;
```

On the other hand, the next line doubles the current value of variable A:

A*=2;

ACS recognizes the following assignment operators:

= += -= *= /= %=

Like C, ACS also supports the use of ++ to increment a variable like this:

A++;

Similarly, -- can be used to decrement a variable.

SWITCH STATEMENTS

A switch statement enables an integral expression to control subsequent execution of a script. It uses the keywords switch, case, break, and default as in the following simple example:

```
switch (a)
   {
   case 1:
                   // when a = 1
       b = 1;
                   // .. this is executed,
       break;
                   // and this breaks out of the switch().
   case 2:
                   // when a = 2
       b = 8;
                   // .. this is executed,
                   // but there is no break, so it continues to the next
                   // case, even though a != 3.
   case 3:
                   // when a = 3
       b = 666;
                  // .. this is executed,
       break;
                  // and this breaks out of the switch().
   default:
                  // when none of the other cases match,
       b = 777; // .. this is executed.
   }
```

JUMP STATEMENTS

A jump statement passes control to another portion of the script. ACS provides three means of jumping: continue, break, and restart. These will be further explained shortly.

SELECTION STATEMENTS

Selection statements permit one statement to be selected in preference to another, as a result of some condition. The general form of this type of statement is:

```
if ( expression ) statement [else statement]
```

ITERATION STATEMENTS

ACS provides five forms of iteration:

```
while ( expression ) statement
until ( expression ) statement
do statement while ( expression ) ;
do statement until ( expression ) ;
for ( assignment-statement ; expression ; assignment-statement ) statement
```

Readers unfamiliar with C might not immediately see from the definition given here how the for statement operates. The three components of the construct used to control the iteration are as follows:

- The first assignment statement of the construct is used to initialize the control variable prior to the first iteration of the loop.
- The central *expression* is used to specify the condition under which the iteration continues to be carried out.
- The second *assignment statement* of the construct is used to modify the control variable after each iteration.

The following for statement could therefore be used to start each of scripts 1 to 10 executing in rapid succession:

for (x=1; x<10; x++) ACS_Execute(x, 0, 0, 0, 0, 0);

The continue, break, and restart keywords can be used within any iteration statement:

- continue forces all subsequent statements to be skipped on the current iteration, but if the control expression permits it, looping will continue.
- break terminates execution of the iteration statement.
- restart forces the entire iteration statement to be restarted.

FUNCTION STATEMENTS

A function statement calls either a Hexen special action code (detailed in the previous chapter), or an internal (Hexen) function. The special action codes can be called by name if SPECIAL.ACS has been included in the compilation. (This file is included automatically if COMMON.ACS is used.) Hexen's internal functions are detailed later in this chapter.

PRINT STATEMENTS

ACS allows text messages to be displayed on the screen to notify the player of various aspects of the way the game is progressing. There are two print statements:

```
print ( print-type : expression , ... ) ;
printbold ( print-type : expression , ... ) ;
```

where *print-type* is one of:

- s string
- d decimal
- c constant



CAUTION: Although, in general, case is not important in ACS scripts, the *print-type* identifiers must be given in lowercase; otherwise, the ACC compiler will return an error.

Both of these functions display their arguments along the top of the screen during play. The difference is that print displays any string arguments in lowercase, while printbold displays strings in uppercase.



CAUTION: Use only uppercase characters in strings passed to print and printbold. Any lowercase characters you use will display as various odd symbols.

You can use this function to pass hints and tips (or just explanatory messages) to the players. For example, if you don't think that the Earthquake effect is obvious from its action, you can replace the use of an Earthquake special on a trigger line with an ACS_Execute special, which calls the script in Listing 2.

Listing 2. Script to create an earthquake with warning message.

```
script 2 ( void )
{
    print ( s: "QUAKE!!!" );
    Earthquake (1, 128, 50, 100, 99);
}
```



NOTE: Some versions of SPECIAL.ACS have the Earthquake function entered as Radius_Quake instead.

The result of triggering this script is illustrated in Figure 1 (although you will need to shake the book violently to see the full effect!). Note the altered case for the message.

Figure 1. Additional notification of an earthquake.



CONTROL STATEMENTS

Finally, ACS provides two control statements:

suspend suspends the script terminate terminates the script

HEXEN'S INTERNAL FUNCTIONS

Hexen's internal functions can be used in ACS scripts just as functions in the C programming language. They can be used either as function statements to invoke an effect, or included in the expression part of an assignment statement if they return a value. Most of Hexen's internal functions return anint. Details of each of Hexen's internal functions follow.

SCRIPT CONTROL FUNCTIONS

```
void tagwait(int tag);
```

Suspends the current script until all sectors marked with tag are inactive.

```
void polywait(int po);
```

Suspends the current script until the PolyObj po is inactive.

```
void scriptwait(int script);
```

Suspends the current script until the specified script has terminated.

```
void delay(int tics);
```

Suspends the current script for the specified number of tics. (A tic is 1/35th of a second.)

SECTOR TEXTURE CHANGING FUNCTIONS

```
void changefloor(int tag, str flatname);
```

Changes the texture on the floor of all sectors marked with tag to the specified flatname.

```
void changeceiling(int tag, str flatname);
```

Changes the texture on the ceiling of all sectors marked with tag to the specified flatname.

LINE CHANGING FUNCTIONS

void clearlinespecial(void);

Clears the special code from the line that activated the script.

void setlinespecial(int line, int special, int arg1, int arg2, int arg3, int arg4, int arg5);

Sets the line special and arguments as specified on all lines identified by *line*. (This identification is provided by means of the line having a special code of type *Line_SetIdentification—see* Table 17 in "Hexen's Special Codes" on the CD-ROM.)

void setlinetexture(int line, int side, int position, str texturename);

Changes the texture used on a specified texture slot on all lines identified by *line*. DEFS.ACS has the following predefined macros for use with setlinetexture():

side:	SIDE_FRONT
	SIDE_BACK
position:	TEXTURE_TOP
	TEXTURE_MIDDLE
	TEXTURE_BOTTOM

For texturename, supply the name of the texture as it appears in a map editor. The name is not case-sensitive.

```
void setlineblocking(int line, int blocking);
```

Sets the Impassable flag on all lines identified by *line*. Predefined macros for the *blocking* parameter are:

ON OFF

SOUND CONTROL FUNCTIONS

Hexen is particularly rich in ways to control the sounds that players hear during the game.

```
void sectorsound(str name, int volume);
```

Plays the specified sound in the sector to the right of the line that activated the script. The *volume* parameter must be in the range 0 to 127.

```
void soundsequence(str name);
```

Plays a specified sound sequence in the sector to the right of the line that activated the script.

```
void thingsound(int tid, str name, int volume);
```

Plays a sound at all Things marked with tid. Again, volume must be the range 0 to 127.

```
void ambientsound(str name, int volume);
```

Plays an ambient sound. These work like ambient sounds in Heretic—they are heard by all players at the same volume.

The following sound functions use special predefined strings to identify the sounds that they produce. Their names are generally sufficient to identify them.

General Player Sounds

PlayerFighterGrunt PlayerMageGrunt PlayerClericFailedUse PlayerFighterPain PlayerMagePain PlayerClericFallingScream PlayerFighterNormalDeath PlayerMageNormalDeath PlayerClericCrazyDeath PlayerFighterExtreme1Death PlayerMageExtreme1Death PlayerClericExtreme2Death PlayerFighterExtreme3Death PlayerMageExtreme3Death PlayerClericBurnDeath PlayerLand PlayerFallingSplat PickupItem PickupPiece PickupArtifact PuzzleSuccess PuzzleFailCleric FighterGrunt ClericCStaffFire FighterPunchHitThing MageLightningFire ClericCStaffExplode FighterAxeHitThing MageLightningContinuous ClericFlameExplode FighterHammerHitThing MageShardsFire HolySymbolFire FighterHammerContinuous MageStaffFire SpiritAttack FighterSwordFire FighterSwordExplode

PlayerClericGrunt PlayerFighterFailedUse PlayerMageFailedUse PlayerClericPain PlayerFighterFallingScream PlayerMageFallingScream PlayerClericNormalDeath PlayerFighterCrazyDeath PlayerMageCrazyDeath PlayerClericExtreme1Death PlayerFighterExtreme2Death PlayerMageExtreme2Death PlayerClericExtreme3Death PlayerFighterBurnDeath PlayerMageBurnDeath PlayerPoisonCough PickupWeapon PickupKey WeaponBuild UseArtifact PuzzleFailFighter PuzzleFailMage FighterPunchMiss MageWandFire ClericCStaffHitThing FighterPunchHitWall MageLightningZap ClericFlameFire FighterHammerMiss MageLightningReady ClericFlameCircle FighterHammerHitWall MageShardsExplode SpiritActive FighterHammerExplode MageStaffExplode SpiritDie

Monster Sounds

EttinSight	EttinActive
EttinPain	EttinAttack
EttinDeath	PuppyBeat (Ettin being poisoned)
FireDemonSpawn	FireDemonActive
FireDemonPain	FireDemonDeath
FireDemonAttack	FireDemonMissileHit
CentaurSight	CentaurActive
CentaurPain	CentaurAttack
CentaurDeath	CentaurLeaderAttack
CentaurMissileExplode	SerpentSight
SerpentActive	SerpentPain
SerpentAttack	SerpentMeleeHit
SerpentDeath	SerpentBirth
SerpentFXContinuous	SerpentFXHit
DemonSight	DemonActive
DemonPain	DemonAttack
DemonDeath	DemonMissileFire
DemonMissileExplode	WraithSight
WraithActive	WraithPain
WraithAttack	WraithDeath
WraithMissileFire	WraithMissileExplode
IceGuySight	IceGuyActive
IceGuyAttack	IceGuyMissileExplode
BishopSight	BishopActive
BishopPain	BishopAttack
BishopDeath	BishopBlur
BishopMissileExplode	DragonSight
DragonActive	DragonWingflap
DragonAttack	DragonPain
DragonDeath	DragonFireballExplode
SorcererSight	SorcererActive
SorcererPain	SorcererDeathScream
SorcererHeadScream	SorcererSpellCast
SorcererBishopSpawn	SorcererBallWoosh
SorcererBallPop	SorcererBallBounce
SorcererBallExplode	SorcererBigBallExplode
KoraxSight	KoraxActive
KoraxStep	KoraxAttack
KoraxCommand	KoraxPain
KoraxDeath	

Other Active Sounds

PigActive1 (snort)	PigActive2 (squeal)
PigPain	PigAttack
PigDeath	MaulatorSight
MaulatorActive	MaulatorPain
MaulatorHamSwing	MaulatorHamHit
MaulatorMissileHit	MaulatorDeath

General Sound Effects

PoisonShroomPain ThrustSpikeRaise FlechetteBounce FreezeDeath MysticIncant GlassShatter TreeBreak BlastRadius Ignite Respawn StartupTick MenuMove Switch2 (switch operating) PlatformStartMetal StoneMove DoorCreak DoorLocked DoorCloseMetal DoorCloseHeavy LavaMove EarthStartMove LavaSizzle ValveTurn BellRing BatScream Earthquake ClockTick

PoisonShroomDeath ThrustSpikeLower FlechetteExplode FreezeShatter PotteryExplode SuitofArmorBreak TreeExplode Fireball Teleport SwitchOtherLevel Chat Switch1 (lock tumblers) PlatformStart PlatformStop MetalMove Door0pen DoorOpenMetal DoorCloseLight WaterMove IceStartMove WaterSplash SludgeGloop RopePull FlyBuzz Drip ThunderCrash

Ambient Sound Effects

Wind	Wind blowing
Ambient1	Wind rattle
Ambient2	Insect or snake hiss
Ambient3	Grasshopper song
Ambient4	Frog song
Ambient5	Cricket chirrup
Ambient6	Owl hoot
Ambient7	Bird cry
Ambient8	Slithering watery sound
Ambient9	Gloop
Ambient10	Water dripping
Ambient11	Rocks falling
Ambient12	Chains rattling
Ambient13	Gong sounding
Ambient14	Metal sheet sound
Ambient15	Rusty metal sheet sound
The Voice of Korax	
KoraxVoiceGreetings	"Greetings, mortal!"
KoraxVoiceReady	"Are you ready to die?"
KoraxVoiceBlood	"My servants can smell your blood, human."
KoraxVoiceGame	"You have played this game too long, mortal."
KoraxVoiceBoard	"I think I shall remove you from the board!"
KoraxVoiceWorship	"Worship me, and I may yet be merciful."
KoraxVoiceMaybe	"Then again, maybe not!"

MISCELLANEOUS FUNCTIONS

int thingcount(int type, int tid);

KoraxVoiceStrong

KoraxVoiceFace

Returns a count of Things in the Hexen world. Use the Thing type definitions in DEFS.ACS for type. Either (or both) type and tid can be set to 0 to force the counting to ignore that information, as in the following examples:

"Are you strong enough..."

"...to face your own masters?"

```
c = thingcount(T_ETTIN, 28); // Count all Ettins that are marked with TID 28
c = thingcount(T_ETTIN, 0); // Count all Ettins, irrespective of TID
c = thingcount(0, 28); // Count all things with TID 28, irrespective of type
int random(int low, int high);
```

Returns a random number between *low* and *high* inclusive. The values for *low* and *high* must be in the range 0 to 255.

int lineside(void);

Returns the side of the line that the script was activated from. The macros LINE_FRONT and LINE_BACK, defined in DEFS.ACS, provide the appropriate values to test for. This allows lines to behave differently, depending upon which way a player moves across them.

int playercount(void);

Returns the number of active players.

int gametype(void);

Returns the type of game being played. Again, DEFS.ACS defines appropriate macros:

GAME_SINGLE_PLAYER

GAME_NET_COOPERATIVE

GAME_NET_DEATHMATCH

int gameskill(void);

Returns the game's current difficulty setting:

SKILL_VERY_EASY

SKILL_EASY

SKILL_NORMAL

SKILL_HARD

SKILL_VERY_HARD

int timer(void);

Returns the time spent on the current level, in tics. (Again, a tic is 1/35 of a second.)

EXIT: MOPPING UP AND MOVING ON

In this chapter, you were presented with the details of Hexen's Action Control Scripting language and given some indication of its capabilities. This chapter concludes the round-up of vital hacking information in the DOOM Data Dossier that started in the book. You should now have all the information at your fingertips to produce virtually any WAD you desire. From now on, it's all down to your imagination, creativity, and dedication to the task!