ACC++ Manual

Everything you could ever want to know about ACS

BitOwl

Table of Contents

Introduction

Introduction to the Manual

This manual is going to be written as I design and write the ACC++ compiler. My hopes are that the manual will describe anything and everything related to ACS. Although the ACC++ project is primarily focused around utilizing the features of ZDoom, included will be documentation on the original byte code format as well as any other extensions that I may stumble upon.

Seeing that I plan on writing this as I go, the beginning will focus mostly on the byte code of ACS. From there I will describe the features of the ACS and ACS++ languages along with any differences from the implementation of ACC. I will not describe the functionality of each individual action special as they are all processed by the lspec instructions. A good reference for the functionality of these action specials would be the ZDoom wiki.[1] Although I will briefly cover the byte code for each built in instruction I will still expect some external reference for specific details.[2]

Introduction to ACS and ACS++

ACS is a scripting language that was designed by Raven Software for use within Hexen. ACS provides a form of C style scripting which vastly expands the capabilities of the Doom engine. The source to the compiler, ACC, has been released by Raven software under a restrictive license. This code base has been used to create the compiler used by ZDoom. ZDoom uses a greatly expanded version of ACS which can be described as the de facto implementation due to its popularity.

Some of ZDoom's extensions, namely arrays, provided the base for which ACS++ can operate on. Specifically global arrays have no definite bounds, which is a property that will be used by ACC++ to allow a form of pointers, and by extension OOP to be possible. Other features implemented by ZDoom will also play a major role within ACS++, such as functions and libraries.

The ACC++ project was started on June 3rd, 2010 after much anticipation from the Doom community (although planning was started as far back as February of 2010). It is licensed under the 3-clause BSD license in hopes that it can be used in various Doom projects, such as within ZDoom itself. ACC++ is a drop in replacement for ACC and will be backwards compatible with existing implementations of the ACS virtual machine, although certain ZDoom extensions will be needed in order to use the ACS++ language.

As a replacement for ACC, ACC++ does not plan on extending the byte code format where possible, and will not specifically allow anything new to be created. Although the availability of object oriented concepts should make some things that would be otherwise difficult to manage a bit easier. In addition ACC++ will not provide complete compatibility with ACS (think C++ vs. C), but should allow existing code to compile with some minor changes.

1 Action special reference can be found at the following location: http://zdoom.org/wiki/Action_specials

2 Reference for the "built in" functions can also be found on the ZDoom wiki: http://zdoom.org/wiki/Built-in_ACS_functions

Byte Code

Data Types and Other Numbers

Before reaching into the detailed specification for the ACS byte code format, we will define our data types. Excepting for strings, all data types will be stored in little endian byte order (least significant first). Strings will be stored as they are seen ending in a '\0' byte unless otherwise noted.

For the purposes of this textbook signed values should be assumed unless otherwise noted. The data type *int* will be used to represent a 32-bit integer, *short* for 16-bit, and *char* for 8-bit (with a string being an array of chars). The floating point type *float* will be 32-bit and *double* for 64-bit.

Script Types

In order to handle various events on the map scripts are given one of the following types. These will be referred to as *ScriptType* within this document.

0.	CLOSED	A normal script with no special trigger.
1.	OPEN	Script that is executed at the start of a game (world). It must be called only once per map, UNLOADING can be used for returns.
2.	RESPAWN	Script that is executed by the player on respawn.
3.	DEATH	Script that is executed by the player when they are killed.
4.	ENTER	Script that is executed by the player when they enter the game. If spectating is supported then this should only be when the player enters the game physically. Should only be executed once per player.
5.	PICKUP[*]	Executes whenever a flag (for any team) is picked up in CTF.
6.	BLUERETURN[*]	Executed when the blue team's flag is returned in CTF.
7.	REDRETURN[*]	Executed when the red team's flag is returned in CTF.
8.	WHITERETURN[*]	Executed with the white flag is returned in one flag CTF.
		Types 9 through 11 are undefined.
12.	LIGHTNING	Executes whenever lightning strikes on the map.
13.	UNLOADING	Executed by the world before the level is finished. It should be executed until the first delay, at which point execution may continue only if the level is returned to.
14.	DISCONNECT	Should take one argument, which will be filled with the player number of the disconnecting player. Note that in ports that allow spectating this should also happen if the player turns into a spectator.
15.	RETURN	Like ENTER, but executed when the player returns to the map.

[*] These were defined for Skulltag's capture the flag game modes. They can be considered deprecated.

Script Flags

In addition to the script type a script may have flags associated which change how the script is executed. Within this document these will be referred to as *ScriptFlags*.

1.	NET	Allows the player to "puke" the script. (That is execute from the console or by button press.)[3]
2.	CLIENTSIDE	Causes the script to be executed by the client in a client server model. This implies that the server does not need to be informed of any changes to the script. Also, any script not carrying this flag is assumed to be server side only.

Magic Limits

Since ZDoom has changed many of the hard limits I feel it's worth documenting these changes. Here is a table of some of the magic numbers with their old and new values.

Limitation	Hexen	ZDoom
Scripts	64	1000
Map Variables	32	128
Script Variables	10	20[4]
World Variables	64	256
Strings	128	32768
Global Variables	N/A	64
Functions	N/A	256
Translations	N/A	32

Variable Scope and Arrays

Due to the various references to scope throughout the byte code documentation it is worth taking some time here to note what each scope means. In general there are 4 scopes in ACS, local, map, world, and global. The last one, global, is an addition by ZDoom, but will be an important role in ACS++.

The local scope is simply all of the variables within a script or function. These are automatically assigned indexes by the compiler. Each script or function should have it's own local scope.

The map scope are those global to a particular behavior lump. It may also include variables pulled in from a library. These, like those in the local scope, are also automatically assigned their indexes. One major difference between map variables and local variables is that map variables can be an array. Exclusive to the map scope arrays is the ability to be initialized

3 At one point in time this flag also implied CLIENTSIDE in Skulltag, however due to incompatibility issues this has been changed to only mean "can be puked."

4 Can be increased with SVCT chunks.

during their definition.

The world scope contains variables that are carried over through an entire hub. When not in a hub they are functionally identical to map variables. However due to the ability to be carried over maps they must be manually assigned their index and can not be initialized during defintion.

The last scope, introduced by ZDoom, is global. These variables are carried during game play to all maps until a new game starts. These are useful for tracking stats between levels and will be the basis for the memory heap in ACS++. Otherwise they are very similar to world variables.

Some things to note about world and global arrays. First is that they do not share their indexes with world and global variables. ZDoom defines a limit of 256 world variables and arrays, and 64 global variables and arrays. The second important feature is that they world and global arrays have no defined size. An ACS++ conforming VM must have the ability to dynamically resize these arrays.

Detecting Byte Code Format

As of this writing there are three known byte code formats. The first we'll be covering is the Hexen byte code format, henceforth known as ACS0, which can be identified by the 4-byte header of "ACS\0". ZDoom introduces two enhanced byte code formats. The first being ACSE, and the latter being ACSe.

Although not required, ZDoom's ACC generally generates an empty ACS0 script at the beginning of the file and thus the detection of the format is not entirely straightforward.

Type	Name	Description
char[4]	Identifier	May be "ACS\0", "ACSE", or "ACSe"
int	DirOffset	

If the *Identifier* is ACSE or ACSe our work is done, otherwise we need to check the *char[4]* at the *DirOffset-4*. If this is ACSE or ACSe then we need to switch to the respective format. In that case we will need to get our new *DirOffset* from *DirOffset-8*.

ACS0 Script Directory And Byte Code

At *DirOffset* the first *int* represents the number of scripts within the lump. The number of scripts is followed by that many pointers. These pointers are in the following format.

Type	Name	Description
int	Number	The actual number is this int mod 1000. This number divided by 1000 will represent the *ScriptType*.
int	Address	Position in which the script starts.
int	ArgCount	The number of arguments the script takes. Due to limitations this should be no larger than 3.

After the script pointers is an *int* for the number of strings in the strings table. This will be followed by the same amount of *int*s referencing the position within the lump at which the string starts. All strings are standard NULL terminated *char* arrays.

An ACS0 script is relatively straightforward. It consists of an *int* for the pcode followed by additional *ints* for any parameters the pcode may take. This is repeated until *PCD_TERMINATE* is reached. See appendix B for a list of pcodes and their functions.

ZDoom Enhanced Formats

ZDoom offers two enhanced byte code formats. Both ACSE and the newer ACSe are chunk based. As such the *DirOffset* from the header now indicates the offset to the start of the chunk data rather than an actual directory. Below is the format of the chunk header.

Type	Name	Description
char[4]	Identifier	
int	Size	

None of the chunks are required, however some of the chunks can be assumed that there are only one instance of them. In this documentation you may assume that you should only find one of each chunk unless otherwise specified.

ARAY, AINI, and AIMP Chunks

The ARAY chunk holds information about the map arrays defined in the script. The number of arrays can be found by dividing the chunk size by 8. The format for each array is an *int* for the array's number followed by an *int* for the size of the array.

An AINI chunk initializes one of the arrays defined in the ARAY chunk. There can be as many of these chunks as there are map arrays defined in the ARAY chunk. The format is an *int* followed by one *int* for each element in the array as defined in ARAY.

The AIMP chunk is used to list information on the arrays that have been imported from a library. It first has an *int* for the number of arrays that have been imported which will be followed by one of the following structures for each array.

Type	Name	Description
int	Number	
int	Size	Expected size.
char[]	Name	Null terminated string.

ASTR and MSTR Chunks

These chunks are used in libraries to notify of variables and arrays that hold string values. They will have to be tagged at run time with an identifier. This identifier is stored in the upper

16-bits of the string's index just as the tagstring instruction does.

The MSTR chunk is for map variables. The number of variables referenced can be found by dividing the chunk size by 4 and will be followed by that many *ints* referencing the map variables by their index.

The ASTR chunk does the same for map arrays. Same format is MSTR except the variable index refers to an array. Note that you will have to flag all the strings in the array based on the size found in the AIMP chunk.

LOAD Chunk

This chunk provides the name of a library in which to load. The chunk is simply a list of null terminated string with the library name. As of this writing these should not be more than 8 characters. The only way to determine how many names there are is by checking to see if the chunk size has been reached.

FUNC and FNAM Chunks

The FUNC chunk holds information about the various functions that are in the script. The number of functions can be determined by dividing the chunk size by 8. For each function the following structure is used.

Type	Name	Description
char	NumArgs	
char	NumVars	
char	HasReturn	Actually a boolean.
char		Always 0.
int	Address	Lump offset to the start of the function's code.

The FNAM chunk is a string table which indexes the names of each function in the lump. For information on the formatting of the string tables see the STRL and STRE chunks documentation.

MEXP, MINI, and MIMP Chunks

The MINI and MIMP chunks are similar to the array counterparts AINI and AIMP. An MINI chunk initializes the map variables. The format is an *int* for the number of map variables initialized followed by one *int* for each variable's value.

The MIMP chunk is used to list information on the map variables that have been imported from a library. This chunk is just a list of the following structures, so you will need to check against the chunk size to determine the number of variables imported.

Type	Name	Description
int	Number	
char[]	Name	Null terminated string.

The MEXP chunk is used in the lump for a library to store string table with the names of the map variables. See the STRL and STRE chunks for more details.

SPTR, SFLG, and SVCT Chunks

The SFLG chunk is an additional chunk used to define the *ScriptFlags* for a particular script. The number of scripts referenced can be found by dividing the chunk size by 4. It will be followed by that many of the following structures.

Type	Name	Description
short	ScriptNumber	
short	ScriptFlags	See *ScriptFlags* reference in Data Types and Other Numbers.

The SVCT flag changes the limit on the number of local variables a script can use (the default limit is 20). The format of this chunk is the same as SFLG except the *ScriptFlags* would represent the number of local variables available.

The SPTR chunk is the replacement for the script directory. It also is what makes the difference between the ACSe and ACSE formats. For the sake of chronological order, I'll be documenting the ACSe version first.

For ACSe the number of scripts defined by the SPTR chunk is the chunk size divided by 12. It will be followed by that many of the following structure.

Type	Name	Description
short	ScriptNumber	
short	ScriptType	See *ScriptType* reference in Data Types and Other Numbers.
int	Address	Relative to the entire lump.
int	ArgCount	

For ACSE the number of script is defined by the size of the chunk divided by 8. The structures will be in the following format.

Type	Name	Description
short	ScriptNumber	
char	ScriptType	See *ScriptType* reference in Data Types and Other Numbers.
char	ArgCount	
int	Address	Relative to the entire lump.

STRL and STRE Chunks

These are the string table chunks. STRL is a normal string table and STRE is an encrypted string table (encryption is enabled through ACC's *#encryptstrings* option). The format of the standard string tables are an *int* that is wasted, an *int* for the number of strings in the table, followed by another wasted *int*. This will be followed by an *int* for each string in the table which represents the offset to that string in the chunk (note that these will also be part of the keys during encryption).

The encryption uses a fairly simple algorithm in which encryption and unencryption is done through the same process. The encryption key is the offset in the chunk to the string multiplied by 157,135. With the key, loop through each character taking the exclusive or of the character with the *key+x* where x is the character number divided by two.

A note about other string table chunks (FNAM, and MEXP), their format does differ slightly in that the two wasted *ints* do not exist.

Appendix A: Script Optimizations

Solving Runaway Scripts

At least in ZDoom, an instruction limit is imposed for every tic. This limit is 500,000 instructions. In general you should not reach this limit, however if you do here are a few tips on how to avoid it. Keep in mind that most of the time when this limit is reached it is because of loops or recursive functions. Therefor even if it may seem like these techniques only save an instruction or two you might end up saving more than you think.

Avoid Providing Optional Arguments

Whenever possible you should avoid providing the optional arguments. Unless you are using a constant expression, each argument produces at least one push instruction which counts towards your limit. For example, if you use the *ACS_Execute* function and you don't need to pass in any arguments only provide the fist two. See the following code:

```
#include "zcommon.acs"
script 1 (void)
{
   // This call will use 5 push instructions followed by an lspec5.
   ACS_Execute(2, 0, 0, 0, 0);

   // This functionally identical call will use 2 push instructions and lspec2.
   ACS_Execute(2, 0);
}
```

Use "const:" Where Possible

Probably the simplest thing to do provided none of your arguments are expressions. Simply place "*const:*" before the first argument on your action special calls and avoid using push instructions completely. Observe the following:

```
#include "zcommon.acs"
script 1 (void)
{
   // 6 instructions
   ACS_Execute(2, 0, 0, 0, 0);

   // 1 instruction
   ACS_Execute(const:2, 0, 0, 0, 0);
}
```

In addition to all line specials, you can use *const* on any of the following functions: Delay, Random, ThingCount, TagWait, PolyWait, ChangeFloor, ChangeCeiling, ScriptWait, SetGravity, SetAirControl, GiveInventory, TakeInventory, CheckInventory, Spawn, SpawnSpot, SetMusic, LocalSetMusic, and SetFont.

This can not be combined with leaving out optional arguments for the purposes of saving "cycles," however you may do so in order to save space on the compiled byte code.

Appendix B: PCode Reference

PCode Table

In all there are 352 defined PCodes. The extension column states what engine introduced the PCode. Required indicates if it is needed for ACS++. In addition all Hexen PCodes are listed as required. “St. Of.” (stack offset) indicates the expected shift in stack size. The codes highlighted in gray are named however are not, and to my knowledge never have been, defined programmatically.

For most functions I'll just say “See FunctionName(<num parameters>[, <max parameters>]).” The number of parameters is a constant in the byte code, so the number of optional parameters is merely provided here for reference purposes. To learn more about these functions I recommend checking out the page on the ZDoom wiki. (http://zdoom.org/wiki/Built-in_ACS_functions)

It is important to note that for function calls in ACS, the first item pushed onto the stack is the first argument, the next would be the second argument and so forth. Likewise when performing operations the first item pushed onto the stack is the operand to the left of the operator and the second item would be the right operand. That said when I describe which stack item to use for what I will be using “1st” to mean the top of the stack and not the first item pushed.

#	PCode	Args	St.Of.	Ext	Req	Description
0	nop	0	0	Hexen	X	Does nothing
1	terminate	0	0	Hexen	X	Finishes the current script.
2	suspend	0	0	Hexen	X	Holds the execution of the script until it is executed again, at which point it will resume.
3	pushnumber	1	1	Hexen	X	Pushes an *int* onto the stack.
4	lspec1	1	-1	Hexen	X	Executes the line special using removing one argument from the stack.
5	lspec2	1	-2	Hexen	X	Same as lspec1 only uses 2 args from the stack.
6	lspec3	1	-3	Hexen	X	Same as lspec1 only uses 3 args from the stack.
7	lspec4	1	-4	Hexen	X	Same as lspec1 only uses 4 args from the stack.
8	lspec5	1	-5	Hexen	X	Same as lspec1 only uses 5 args from the stack.
9	lspec1direct	2	0	Hexen	X	Executes the line special in the first arg with one constant parameter.
10	lspec2direct	3	0	Hexen	X	Same as lspec1direct only with 2 arguments.
11	lspec3direct	4	0	Hexen	X	Same as lspec1direct only with 3 arguments.
12	lspec4direct	5	0	Hexen	X	Same as lspec1direct only with 4 arguments.
13	lspec5direct	6	0	Hexen	X	Same as lspec1direct only with 5 arguments.
14	add	0	-1	Hexen	X	Adds the last two items from the stack, pushing the result.

#	PCode	Args	St.Of.	Ext	Req	Description
15	subtract	0	-1	Hexen	X	Subtracts the last two items from the stack pushing the result.
16	multiply	0	-1	Hexen	X	Multiplies the last two items from the stack pushing the result.
17	divide	0	-1	Hexen	X	Divides the last two items from the stack pushing the result. Errors on division by zero.
18	modulus	0	-1	Hexen	X	Calculates the remainder of division of the last two items on the stack, pushing the result. Errors on modulus by zero.
19	eq	0	-1	Hexen	X	Pushes 1 to the stack if the last two items are equivalent, otherwise pushes 0.
20	ne	0	-1	Hexen	X	Pushes 1 to the stack if the last two items are not equivalent, otherwise pushes 0.
21	lt	0	-1	Hexen	X	Pushes 1 to the stack if the 2nd item in the stack is less than the 1st, otherwise pushes 0.
22	gt	0	-1	Hexen	X	Pushes 1 to the stack if the 2nd item in the stack is greater than the 1st, otherwise pushes 0.
23	le	0	-1	Hexen	X	Pushes 1 to the stack if the 2nd item in the stack is less or equal to the 1st, otherwise pushes 0.
24	ge	0	-1	Hexen	X	Pushes 1 to the stack if the 2nd item in the stack is greater than or equal to 1st, otherwise pushes 0.
25	assignscriptvar	1	-1	Hexen	X	Assigns the value on the stack to the specified local variable.
26	assignmapvar	1	-1	Hexen	X	Assigns the value on the stack to the specified map variable.
27	assignworldvar	1	-1	Hexen	X	Assigns the value on the stack to the specified world variable.
28	pushscriptvar	1	1	Hexen	X	Pushes the value of the local variable onto the stack.
29	pushmapvar	1	1	Hexen	X	Pushes the value of the map variable onto the stack.
30	pushworldvar	1	1	Hexen	X	Pushes the value of the world variable onto the stack.
31	addscriptvar	1	-1	Hexen	X	Adds the stack to the specified local variable.
32	addmapvar	1	-1	Hexen	X	Adds the stack to the specified map variable.
33	addworldvar	1	-1	Hexen	X	Adds the stack to the specified world variable.
34	subscriptvar	1	-1	Hexen	X	Subtracts the stack from the specified local variable.
35	submapvar	1	-1	Hexen	X	Subtracts the stack from the specified map variable.
36	subworldvar	1	-1	Hexen	X	Subtracts the stack from the specified world variable.
37	mulscriptvar	1	-1	Hexen	X	Multiplies the stack to the specified local variable.
38	mulmapvar	1	-1	Hexen	X	Multiplies the stack to the specified map variable.
39	mulworldvar	1	-1	Hexen	X	Multiplies the stack to the specified world variable.

#	PCode	Args	St.Of.	Ext	Req	Description
40	divscriptvar	1	-1	Hexen	X	Divides the stack from the specified local variable. Errors on division by zero.
41	divmapvar	1	-1	Hexen	X	Divides the stack from the specified map variable. Errors on division by zero.
42	divworldvar	1	-1	Hexen	X	Divides the stack from the specified world variable. Errors on division by zero.
43	modscriptvar	1	-1	Hexen	X	Takes modulus of the local variable by the stack.
44	modmapvar	1	-1	Hexen	X	Takes modulus of the map variable by the stack.
45	modworldvar	1	-1	Hexen	X	Takes modulus of the world variable by the stack.
46	incscriptvar	1	0	Hexen	X	Increments the local variable by one.
47	incmapvar	1	0	Hexen	X	Increments the map variable by one.
48	incworldvar	1	0	Hexen	X	Increments the world variable by one.
49	decscriptvar	1	0	Hexen	X	Decrements the local variable by one.
50	decmapvar	1	0	Hexen	X	Decrements the map variable by one.
51	decworldvar	1	0	Hexen	X	Decrements the world variable by one.
52	goto	1	0	Hexen	X	Sets the script execution to the offset specified.
53	ifgoto	1	-1	Hexen	X	Same as goto except it first checks to see if the stack doesn't equal zero.
54	drop	0	-1	Hexen	X	Removes one value from the stack.[5]
55	delay	0	-1	Hexen	X	Causes the script to wait for the number of tics specified on the stack. See also Delay(2).
56	delaydirect	1	0	Hexen	X	Same as delay only with constant parameters.
57	random	0	-1	Hexen	X	Picks a random number inclusively between the last two items on the stack. See also Random(2).
58	randomdirect	2	1	Hexen	X	Same as random only uses constant parameters.
59	thingcount	0	-1	Hexen	X	See ThingCount(2)
60	thingcountdirect	2	1	Hexen	X	Same as thingcount, uses constant parameters.
61	tagwait	0	-1	Hexen	X	Waits for the tag in the stack to finish moving. See also TagWait(1).
62	tagwaitdirect	1	0	Hexen	X	Same as tagwait, uses constant parameters.
63	polywait	0	-1	Hexen	X	Waits for the polyobj number in the stack to finish moving. See PolyWait(1).
64	polywaitdirect	1	0	Hexen	X	Same as polywait, uses constant parameters.
65	changefloor	0	-2	Hexen	X	See ChangeFloor(2).
66	changefloordirect	2	0	Hexen	X	Same as changefloor, uses constant parameters.
67	changeceiling	0	-2	Hexen	X	See ChangeCeiling(2).

5 In ZDoom this is handled identically to the PCode setresultvalue.

#	PCode	Args	St.Of.	Ext	Req	Description
68	changeceilingdirect	2	0	Hexen	X	Same as changeceiling, uses constant parameters.
69	restart	0	0	Hexen	X	Sets the script execution offset back to the beginning of the script.
70	andlogical	0	-1	Hexen	X	Pushes 1 if the last two items on the stack are true, otherwise pushes 0.
71	orlogical	0	-1	Hexen	X	Pushes 1 if either of the last two items on the stack are true, otherwise pushes 0.
72	andbitwise	0	-1	Hexen	X	Performs a bitwise and operation on the last two items on the stack, pushing the result.
73	orbitwise	0	-1	Hexen	X	Performs a bitwise or optionation on the last two items on the stack, pushing the result.
74	eorbitwise	0	-1	Hexen	X	Performs a bitwise exclusive or optionation on the last two items on the stack, pushing the result.
75	negatelogical	0	0	Hexen	X	Switches the last item on the stack from true to false or vice versa.
76	lshift	0	-1	Hexen	X	Performs a left shift operation on the last two items on the stack, pushing the result.
77	rshift	0	-1	Hexen	X	Performs a right shift operation on the last two items on the stack, pushing the result.
78	unaryminus	0	0	Hexen	X	Changes the last item on the stack from positive to negative or vice versa.
79	ifnotgoto	1	-1	Hexen	X	Sets the script execution to the offset specified only if the stack is not true.
80	lineside	0	1	Hexen	X	See LineSize(0).
81	scriptwait	0	-1	Hexen	X	See ScriptWait(1).
82	scriptwaitdirect	1	0	Hexen	X	Same as scriptwait, uses constant parameters.
83	clearlinespecial	0	0	Hexen	X	See ClearLineSpecial(0).
84	casegoto	2	-1	Hexen	X	Sets the execution to the 2nd parameter if the stack is equal to the 1st.
85	beginprint	0	0	Hexen	X	Starts collecting information for the print or printbold functions.
86	endprint	0	0	Hexen	X	Finishes a print() call.
87	printstring	0	-1	Hexen	X	Adds the string referenced by the stack to the print.
88	printnumber	0	-1	Hexen	X	Adds the number referenced by the stack to the print.
89	printcharacter	0	-1	Hexen	X	Adds the character referenced by the stack to the print.
90	playercount	0	1	Hexen	X	See PlayerCount(0).
91	gametype	0	1	Hexen	X	See GameType(0).
92	gameskill	0	1	Hexen	X	See GameSkill(0).
93	timer	0	1	Hexen	X	See Timer(0).

#	PCode	Args	St.Of.	Ext	Req	Description
94	sectorsound	0	-2	Hexen	X	See SectorSound(2).
95	ambientsound	0	-2	Hexen	X	See AmbientSound(2).
96	soundsequence	0	-1	Hexen	X	See SoundSequence(1).
97	setlinetexture	0	-4	Hexen	X	See SetLineTexture(4).
98	setlineblocking	0	-2	Hexen	X	See SetLineBlocking(2).[6]
99	setlinespecial	0	-7	Hexen	X	See SetLineSpecial(2,7).
100	thingsound	0	-3	Hexen	X	See ThingSound(3).
101	endprintbold	0	0	Hexen	X	Finishes the "bold" print statement. All players should receive this message.
102	activatorsound	0	-2	ZDoom		See ActivatorSound(2).
103	localambientsound	0	-2	ZDoom		See LocalAmbientSound(2).
104	setlinemonsterblocking	0	-2	ZDoom		See SetLineMonsterBlocking(2).
105	playerblueskull	0	1	Skulltag		Pushes -1 to the stack.
106	playerredskull	0	1	Skulltag		Pushes -1 to the stack.
107	playeryellowskull	0	1	Skulltag		Pushes -1 to the stack.
108	playermasterskull			Skulltag		*Completely undefined at this time.*
109	playerbluecard	0	1	Skulltag		Pushes -1 to the stack.
110	playerredcard	0	1	Skulltag		Pushes -1 to the stack.
111	playeryellowcard	0	1	Skulltag		Pushes -1 to the stack.
112	playermastercard			Skulltag		*Completely undefined at this time.*
113	playerblackskull			Skulltag		*Completely undefined at this time.*
114	playersilverskull			Skulltag		*Completely undefined at this time.*
115	playergoldskull			Skulltag		*Completely undefined at this time.*
116	playerblackcard			Skulltag		*Completely undefined at this time.*
117	playersilvercard			Skulltag		*Completely undefined at this time.*
118	playeronteam			Skulltag		*Completely undefined at this time.*
119	playerteam	0	1	Skulltag		See PlayerTeam(0).
120	playerhealth	0	1	Skulltag		See PlayerHealth(0).
121	playerarmorpoints	0	1	Skulltag		See PlayerArmorPoints(0).
122	playerfrags	0	1	Skulltag		See PlayerFrags(0).
123	playerexpert			Skulltag		*Completely undefined at this time.*
124	blueteamcount	0	1	Skulltag		See BlueCount(0).
125	redteamcount	0	1	Skulltag		See RedCount(0).

6 In Hexen it may be assumed that the second argument to this function is a boolean, however other ports have extended this to include multiple flags.

#	PCode	Args	St.Of.	Ext	Req	Description
126	blueteamscore	0	1	Skulltag		See BlueScore(0).
127	redteamscore	0	1	Skulltag		See RedScore(0).
128	isoneflagctf	0	1	Skulltag		See IsOneFlagCTF(0).
129	getinvasionwave	0	1	Skulltag		See GetInvasionWave(0).
130	getinvasionstate	0	1	Skulltag		See GetInvasionState(0).
131	printname	0	-1	Skulltag		Adds the name of the player referenced by the number on the stack to the current print.
132	musicchange	0	-2	Skulltag		See Music_Change(2).
133	consolecommanddirect	3	0	Skulltag		Takes the specified command and executes it.[7] Last two parameters are unused.
134	consolecommand	0	-3	Skulltag		Same as consolecommanddirect, except it reads from the stack.
135	singleplayer	0	1	Skulltag		Pushes 1 to the stack if playing in single player otherwise pushes 0.
136	fixedmul	0	-1	ZDoom	X	Multiplies the last two fixed point numbers on the stack and pushes the result.
137	fixeddiv	0	-1	ZDoom	X	Divides the last two fixed point numbers on the stack and pushes the result.
138	setgravity	0	-1	ZDoom		See SetGravity(1).
139	setgravitydirect	1	0	ZDoom		Same as setgravity, uses constant parameters.
140	setaircontrol	0	-1	ZDoom		See SetAirControl(1).
141	setaircontroldirect	1	0	ZDoom		Same as setaircontrol, uses constant parameters.
142	clearinventory	0	0	ZDoom		See ClearInventory(0).
143	giveinventory	0	-2	ZDoom		See GiveInventory(2).
144	giveinventorydirect	2	0	ZDoom		Same as giveinventory, uses constant parameters.
145	takeinventory	0	-2	ZDoom		See TakeInventory(2).
146	takeinventorydirect	2	0	ZDoom		Same as takeinventory, uses constant parameters.
147	checkinventory	0	0	ZDoom		See CheckInventory(1).
148	checkinventorydirect	1	0	ZDoom		Same as checkinventory, uses constant parameters.
149	spawn	0	-5	ZDoom		See Spawn(4,6). Pushes 1 on success 0 on failure.
150	spawndirect	6	0	ZDoom		Same as spawn, uses constant parameters.
151	spawnspot	0	-3	ZDoom		See SpawnSpot(2,4). Pushes 1 on success 0 on failure.
152	spawnspotdirect	4	0	ZDoom		Same as spawnspot, uses constant parameters.
153	setmusic	0	-3	ZDoom		See SetMusic(1,3).
154	setmusicdirect	3	0	ZDoom		Same as setmusic, uses constant parameters.

7 It is worth noting here that consolecommand can be a serious security issue. As such I would advise against implementing it.

#	PCode	Args	St.Of.	Ext	Req	Description
155	localsetmusic	0	-3	ZDoom		See LocalSetMusic(1,3).
156	localsetmusicdirect	3	0	ZDoom		Same as localsetmusic, uses constant parameters.
157	printfixed	0	-1	ZDoom		Adds the fixed point number on the stack to the print.
158	printlocalized	0	-1	ZDoom		Adds the localized string referenced by the stack to the print command.
159	morehudmessage	0	0	ZDoom		Effectively ends the print portion of hudmessage. It does not print the message.
160	opthudmessage	0	0	ZDoom		Sets the position of hudmessage arguments in the stack to the current stack pointer. Finishing the parameter section of hudmessage. See hudmessage on the ZDoom wiki for detailed information on the parameters.
161	endhudmessage	0	*x*	ZDoom		Finishes an hudmessage command. Removing all parameters (terminated by opthudmessage) from the stack.
162	endhudmessagebold	0	*x*	ZDoom		Same as endhudmessage only prints the message to all players.
163	setstyle			ZDoom		*Completely undefined at this time.*
164	setstyledirect			ZDoom		*Completely undefined at this time.*
165	setfont	0	-1	ZDoom		See SetFont(1).
166	setfontdirect	1	0	ZDoom		Same as setfont, uses constant parameters.
167	pushbyte	1	0	ZDoom		Pushes a *char* onto the stack.
168	lspec1directb	2	0	ZDoom		Same as lspec1direct except all parameters are *chars*.
169	lspec2directb	3	0	ZDoom		Same as lspec2direct except all parameters are *chars*.
170	lspec3directb	4	0	ZDoom		Same as lspec3direct except all parameters are *chars*.
171	lspec4directb	5	0	ZDoom		Same as lspec4direct except all parameters are *chars*.
172	lspec5directb	6	0	ZDoom		Same as lspec5direct except all parameters are *chars*.
173	delaydirectb	1	0	ZDoom		Same as delaydirect except it takes a *char*.
174	randomdirectb	2	0	ZDoom		Same as randomdirect except all parameters are *chars*.
175	pushbytes	*n+1*	*n*	ZDoom		Pushes *n chars* to the stack where *n* is the first *char.*
176	push2bytes	2	2	ZDoom		Pushes 2 *chars* to the stack.
177	push3bytes	3	3	ZDoom		Pushes 3 *chars* to the stack.
178	push4bytes	4	4	ZDoom		Pushes 4 *chars* to the stack.
179	push5bytes	5	5	ZDoom		Pushes 5 *chars* to the stack.
180	setthingspecial	0	-7	ZDoom		See SetThingSpecial(2,7).
181	assignglobalvar	1	-1	ZDoom	X	Assigns the value on the stack to the specified global variable.

#	PCode	Args	St.Of.	Ext	Req	Description
182	pushglobalvar	1	1	ZDoom	X	Pushes the value of the specified global variable onto the stack.
183	addglobalvar	1	-1	ZDoom	X	Adds the value of the stack to the specified global var.
184	subglobalvar	1	-1	ZDoom	X	Substracts the value of the stack from the specified global variable.
185	mulglobalvar	1	-1	ZDoom	X	Multiplies the global variable by the value on the stack.
186	divglobalvar	1	-1	ZDoom	X	Divides the global variable by the value on the stack. Errors on division by zero.
187	modglobalvar	1	-1	ZDoom	X	Does the modulus of the global variable with the value on the stack. Errors on modulus by zero.
188	incglobalvar	1	0	ZDoom	X	Increments the global variable by one.
189	decglobalvar	1	0	ZDoom	X	Decrements the global variable by one.
190	fadeto	0	-5	ZDoom		See FadeTo(5).
191	faderange	0	-9	ZDoom		See FadeRange(9).
192	cancelfade	0	0	ZDoom		See CancelFade(0).
193	playmovie	0	0	ZDoom		See PlayMovie(1).
194	setfloortrigger	0	-8	ZDoom		See SetFloorTrigger(3,8).
195	setceilingtrigger	0	-8	ZDoom		See SetCeilingTrigger(3,8).
196	getactorx	0	1	ZDoom		See GetActorX(1).
197	getactory	0	1	ZDoom		See GetActorY(1).
198	getactorz	0	1	ZDoom		See GetActorZ(1).
199	starttranslation	0	-1	ZDoom		Begins the translation definition for the translation specified on the stack. See CreateTranslation(2).
200	translationrange1	0	-4	ZDoom		Specifies a translation between palette indexes. Stack represents 1^{st}:2^{nd}=3^{rd}:4^{th}. See CreateTranslation(2).
201	translationrange2	0	-8	ZDoom		Specifies a translation from palette indexes to RGB values. Stack represents 1^{st}:2^{nd}=[3^{rd},4^{th},5^{th}]:[6^{th},7^{th},8^{th}]. See CreateTranslation(2).
202	endtranslation	0	0	ZDoom		Finishes a call to CreateTranslation(2).
203	call	1	1	ZDoom	X	Calls the function specified by index.
204	calldiscard	1	0	ZDoom	X	Same as call, except the value returned will not be pushed to the stack. (Alternatively dropped.)
205	returnvoid	0	1/0	ZDoom	X	Returns from a function pushing 0 to the stack unless called by calldiscard.
206	returnval	0	1/0	ZDoom	X	Returns from a function keeping the top value of the stack unless called by calldiscard.
207	pushmaparray	1	0	ZDoom	X	Pushes the value of the specified map array at the index referenced by the stack.

#	PCode	Args	St.Of.	Ext	Req	Description
208	assignmaparray	1	-2	ZDoom	X	Assigns the value of 1st stack item to the specified map array at the index referenced by the 2nd item.
209	addmaparray	1	-2	ZDoom	X	Adds the value of the 1st stack item to the specified map array at the index of the 2nd stack item.
210	submaparray	1	-2	ZDoom	X	Subtracts the value of the 1st stack item to the specified map array at the index of the 2nd stack item.
211	mulmaparray	1	-2	ZDoom	X	Multiplies the specified map array's index referenced by the 2nd item on the stack by the 1st.
212	divmaparray	1	-2	ZDoom	X	Divides the specified map array's index referenced by the 2nd item on the stack by the 1st. Errors on division by zero.
213	modmaparray	1	-2	ZDoom	X	Performs the modulus of the specified map array's index referenced by the 2nd item on the stack by the 1st item on the stack. Errors on modulus by zero.
214	incmaparray	1	-1	ZDoom	X	Increments the specified map array at the index referenced by the stack by one.
215	decmaparray	1	-1	ZDoom	X	Decrements the specified map array at the index referenced by the stack by one.
216	dup	0	1	ZDoom		Pushes the value of the stack to the stack.
217	swap	0	0	ZDoom		Switches the last two values on the stack.
218	writetoini			ZDoom		*Completely undefined at this time.*
219	getfromini			ZDoom		*Completely undefined at this time.*
220	sin	0	0	ZDoom		See Sin(1).
221	cos	0	0	ZDoom		See Cos(1).
222	vectorangle	0	-1	ZDoom		See VectorAngle(2).
223	checkweapon	0	0	ZDoom		See CheckWeapon(1).
224	setweapon	0	0	ZDoom		See SetWeapon(1).
225	tagstring	0	0	ZDoom	X	Puts the id of the library into upper 16-bits of the stack.
226	pushworldarray	1	0	ZDoom	X	Pushes the value of the specified world array at the index referenced by the stack.
227	assignworldarray	1	-2	ZDoom	X	Assigns the value of 1st stack item to the specified world array at the index referenced by the 2nd item.
228	addworldarray	1	-2	ZDoom	X	Adds the value of the 1st stack item to the specified world array at the index of the 2nd stack item.
229	subworldarray	1	-2	ZDoom	X	Subtracts the value of the 1st stack item to the specified world array at the index of the 2nd stack item.
230	mulworldarray	1	-2	ZDoom	X	Multiplies the specified world array's index referenced by the 2nd item on the stack by the 1st.

#	PCode	Args	St.Of.	Ext	Req	Description
231	divworldarray	1	-2	ZDoom	X	Divides the specified world array's index referenced by the 2nd item on the stack by the 1st. Errors on division by zero.
232	modworldarray	1	-2	ZDoom	X	Performs the modulus of the specified world array's index referenced by the 2nd item on the stack by the 1st item on the stack. Errors on modulus by zero.
233	incworldarray	1	-1	ZDoom	X	Increments the specified world array at the index referenced by the stack by one.
234	decworldarray	1	-1	ZDoom	X	Decrements the specified world array at the index referenced by the stack by one.
235	pushglobalarray	1	0	ZDoom	X	Pushes the value of the specified global array at the index referenced by the stack.
236	assignglobalarray	1	-2	ZDoom	X	Assigns the value of 1st stack item to the specified global array at the index referenced by the 2nd item.
237	addglobalarray	1	-2	ZDoom	X	Adds the value of the 1st stack item to the specified global array at the index of the 2nd stack item.
238	subglobalarray	1	-2	ZDoom	X	Subtracts the value of the 1st stack item to the specified global array at the index of the 2nd stack item.
239	mulglobalarray	1	-2	ZDoom	X	Multiplies the specified global array's index referenced by the 2nd item on the stack by the 1st.
240	divglobalarray	1	-2	ZDoom	X	Divides the specified global array's index referenced by the 2nd item on the stack by the 1st. Errors on division by zero.
241	modglobalarray	1	-2	ZDoom	X	Performs the modulus of the specified global array's index referenced by the 2nd item on the stack by the 1st item on the stack. Errors on modulus by zero.
242	incglobalarray	1	-1	ZDoom	X	Increments the specified global array at the index referenced by the stack by one.
243	decglobalarray	1	-1	ZDoom	X	Decrements the specified global array at the index referenced by the stack by one.
244	setmarineweapon	0	-2	ZDoom		See SetMarineWeapon(2).
245	setactorproperty	0	-3	ZDoom		See SetActorProperty(3).
246	getactorproperty	0	-1	ZDoom		See GetActorProperty(2).
247	playernumber	0	1	ZDoom		See PlayerNumber(0).
248	activatortid	0	1	ZDoom		See ActivatorTID(0).
249	setmarinesprite	0	-2	ZDoom		See SetMarineSprite(2).
250	getscreenwidth	0	1	ZDoom		See GetScreenWidth(0).
251	getscreenheight	0	1	ZDoom		See GetScreenHeight(0).
252	thingprojectile2	0	-7	ZDoom		See Thing_Projectile2(7).
253	strlen	0	0	ZDoom	X	See StrLen(1).

#	PCode	Args	St.Of.	Ext	Req	Description
254	sethudsize	0	-3	ZDoom		See SetHudSize(3).
255	getcvar	0	0	ZDoom		See GetCVar(1).
256	casegotosorted	*n+1*	-1	ZDoom	X	First ensure that your script data pointer is 4 byte aligned. The first *short* indicates the number of cases. The cases should be sorted in such a way that a binary search may be performed. The case to jump to is the one where the case matches the stack.
257	setresultvalue	0	-1	ZDoom	X	Sets the result of the script to the stack.
258	getlinerowoffset	0	1	ZDoom		See GetLineRowOffset(0).
259	getactorfloorz	0	0	ZDoom		See GetActorFloorZ(0).
260	getactorangle	0	0	ZDoom		See GetActorAngle(0).
261	getsectorfloorz	0	-2	ZDoom		See GetSectorFloorZ(3).
262	getsectorceilingz	0	-2	ZDoom		See GetSectorCeilingZ(3).
263	lspec5result	0	-4	ZDoom	X	Same as lspec5 except the result of the special is pushed onto the stack.
264	getsigilpieces	0	1	ZDoom		See GetSigilPieces(0).
265	getlevelinfo	0	0	ZDoom		See GetLevelInfo(1).
266	changesky	0	-2	ZDoom		See ChangeSky(2).
267	playeringame	0	0	ZDoom		See PlayerInGame(1).
268	playerisbot	0	0	ZDoom		See PlayerIsBot(1).
269	setcameratotexture	0	-3	ZDoom		See SetCameraToTexture(3).
270	endlog	0	0	ZDoom		Ends a log call.
271	getammocapacity	0	0	ZDoom		See GetAmmoCapacity(1).
272	setammocapacity	0	-2	ZDoom		See SetAmmoCapacity(2).
273	printmapchararray	0	-2	ZDoom		Adds the string represented by the character array in the map array specified by the 1st stack item at the offset in the 2nd stack item.
274	printworldchararray	0	-2	ZDoom		Adds the string represented by the character array in the world array specified by the 1st stack item at the offset in the 2nd stack item.
275	printglobalchararray	0	-2	ZDoom		Adds the string represented by the character array in the global array specified by the 1st stack item at the offset in the 2nd stack item.
276	setactorangle	0	-2	ZDoom		See SetActorAngle(2).
277	grabinput			ZDoom		*Completely undefined at this time.*
278	setmousepointer			ZDoom		*Completely undefined at this time.*
279	movemousepointer			ZDoom		*Completely undefined at this time.*
280	spawnprojectile	0	-7	ZDoom		See SpawnProjectile(7).

#	PCode	Args	St.Of.	Ext	Req	Description
281	getsectorlightlevel	0	0	ZDoom		See GetSectorLightLevel(1).
282	getactorceilingz	0	0	ZDoom		See GetActorCeilingZ(1).
283	getactorpositionz			ZDoom		*Completely undefined at this time.*
284	clearactorinventory	0	-1	ZDoom		See ClearActorInventory(1).
285	giveactorinventory	0	-3	ZDoom		See GiveActorInventory(3).
286	takeactorinventory	0	-3	ZDoom		See TakeActorInventory(3).
287	checkactorinventory	0	-1	ZDoom		See CheckActorInventory(2).
288	thingcountname	0	-1	ZDoom		See ThingCountName(2).
289	spawnspotfacing	0	-2	ZDoom		See SpawnSpotFacing(3).
290	playerclass	0	0	ZDoom		See PlayerClass(1).
291	andscriptvar	1	-1	ZDoom	X	Performs the bitwise and operation with the specified local variable and the stack.
292	andmapvar	1	-1	ZDoom	X	Performs the bitwise and operation with the specified map variable and the stack.
293	andworldvar	1	-1	ZDoom	X	Performs the bitwise and operation with the specified world variable and the stack.
294	andglobalvar	1	-1	ZDoom	X	Performs the bitwise and operation with the specified global variable and the stack.
295	andmaparray	1	-2	ZDoom	X	Performs the bitwise and operation with the specified map array at the index of the 2nd stack item and the 1st stack item.
296	andworldarray	1	-2	ZDoom	X	Performs the bitwise and operation with the specified world array at the index of the 2nd stack item and the 1st stack item.
297	andglobalarray	1	-2	ZDoom	X	Performs the bitwise and operation with the specified global array at the index of the 2nd stack item and the 1st stack item.
298	eorscriptvar	1	-1	ZDoom	X	Performs the bitwise exclusive or operation with the specified local variable and the stack.
299	eormapvar	1	-1	ZDoom	X	Performs the bitwise exclusive or operation with the specified map variable and the stack.
300	eorworldvar	1	-1	ZDoom	X	Performs the bitwise exclusive or operation with the specified world variable and the stack.
301	eorglobalvar	1	-1	ZDoom	X	Performs the bitwise exclusive or operation with the specified global variable and the stack.
302	eormaparray	1	-2	ZDoom	X	Performs the bitwise excluse or operation with the specified map array at the index of the 2nd stack item and the 1st stack item.
303	eorworldarray	1	-2	ZDoom	X	Performs the bitwise excluse or operation with the specified world array at the index of the 2nd stack item and the 1st stack item.

#	PCode	Args	St.Of.	Ext	Req	Description
304	eorglobalarray	1	-2	ZDoom	X	Performs the bitwise excluse or operation with the specified global array at the index of the 2nd stack item and the 1st stack item.
305	orscriptvar	1	-1	ZDoom	X	Performs the bitwise or operation with the specified local variable and the stack.
306	ormapvar	1	-1	ZDoom	X	Performs the bitwise or operation with the specified map variable and the stack.
307	orworldvar	1	-1	ZDoom	X	Performs the bitwise or operation with the specified world variable and the stack.
308	orglobalvar	1	-1	ZDoom	X	Performs the bitwise or operation with the specified global variable and the stack.
309	ormaparray	1	-2	ZDoom	X	Performs the bitwise or operation with the specified map array at the index of the 2nd stack item and the 1st stack item.
310	orworldarray	1	-2	ZDoom	X	Performs the bitwise or operation with the specified world array at the index of the 2nd stack item and the 1st stack item.
311	orglobalarray	1	-2	ZDoom	X	Performs the bitwise or operation with the specified global array at the index of the 2nd stack item and the 1st stack item.
312	lsscriptvar	1	-1	ZDoom	X	Left shifts the specified local variable by the stack.
313	lsmapvar	1	-1	ZDoom	X	Left shifts the specified map variable by the stack.
314	lsworldvar	1	-1	ZDoom	X	Left shifts the specified world variable by the stack.
315	lsglobalvar	1	-1	ZDoom	X	Left shifts the specified global variable by the stack.
316	lsmaparray	1	-2	ZDoom	X	Left shifts the specified map array at the index referenced by the 2nd stack item by the 1st item.
317	lsworldarray	1	-2	ZDoom	X	Left shifts the specified world array at the index referenced by the 2nd stack item by the 1st item.
318	lsglobalarray	1	-2	ZDoom	X	Left shifts the specified global array at the index referenced by the 2nd stack item by the 1st item.
319	rsscriptvar	1	-1	ZDoom	X	Right shifts the specified local variable by the stack.
320	rsmapvar	1	-1	ZDoom	X	Right shifts the specified map variable by the stack.
321	rsworldvar	1	-1	ZDoom	X	Right shifts the specified world variable by the stack.
322	rsglobalvar	1	-1	ZDoom	X	Right shifts the specified global variable by the stack.
323	rsmaparray	1	-2	ZDoom	X	Right shifts the specified map array at the index referenced by the 2nd stack item by the 1st item.
324	rsworldarray	1	-2	ZDoom	X	Right shifts the specified local array at the index referenced by the 2nd stack item by the 1st item.
325	rsglobalarray	1	-2	ZDoom	X	Right shifts the specified global array at the index referenced by the 2nd stack item by the 1st item.
326	getplayerinfo	0	-1	ZDoom		See GetPlayerInfo(2).

#	PCode	Args	St.Of.	Ext	Req	Description
327	changelevel	0	-4	ZDoom		See ChangeLevel(3,4).
328	sectordamage	0	-5	ZDoom		See SectorDamage(5).
329	replacetextures	0	-3	ZDoom		See ReplaceTextures(2,3).
330	negatebinary	0	0	ZDoom	X	Performs a binary negation on the stack.
331	getactorpitch	0	0	ZDoom		See GetActorPitch(1).
332	setactorpitch	0	-2	ZDoom		See SetActorPitch(2).
333	printbind	0	-1	ZDoom		Adds the name of the key in which the action referenced by the stack is bound to the print.
334	setactorstate	0	-2	ZDoom		See SetActorState(2,3).
335	thingdamage2	0	-2	ZDoom		See Thing_Damage2(3).
336	useinventory	0	0	ZDoom		See UseInventory(1).
337	useactorinventory	0	-1	ZDoom		See UseActorInventory(2).
338	checkactorceilingtexture	0	-1	ZDoom		See CheckActorCeilingTexture(2).
339	checkactorfloortexture	0	-1	ZDoom		See CheckActorFloorTexture(2).
340	getactorlightlevel	0	0	ZDoom		See GetActorLightLevel(1).
341	setmugshotstate	0	-1	ZDoom		See SetMugShotState(1).
342	thingcountsector	0	-2	ZDoom		See ThingCountSector(3).
343	thingcountnamesector	0	-2	ZDoom		See ThingCountNameSector(3).
344	checkplayercamera	0	0	ZDoom		See CheckPlayerCamera(1).
345	morphactor	0	-6	ZDoom		See MorphActor(1,7).
346	unmorphactor	0	-1	ZDoom		See UnMorphActor(1,2).
347	getplayerinput	0	-1	ZDoom		See GetPlayerInput(2).
348	classifyactor	0	0	ZDoom		See ClassifyActor(1).
349	printbinary	0	-1	ZDoom		Adds the binary number to the print.
350	printhex	0	-1	ZDoom		Adds the hexadecimal number to the print.
351	callfunc	2	*-n+1*	ZDoom	X	Calls an alternative type of built in function. The first *char* following the PCode is the number of arguments. The next *short* is the function index (see below). After all of the arguments are removed from the stack, the return value is pushed to the stack. If this is used in ACS0 the first two parameters are also *ints*.

Additional Functions

The callfunc PCode references a table of functions. These functions are handled similar to the normal built in functions, however they are more flexible (argument count is not a constant for example). The PCode always pushes a return value onto the stack so it must be dropped if it is to be ignored. Continue to refer to the ZDoom wiki's built in ACS function documentation for details on their exact operations.

#	Function	Args	Ret	Req
1	GetLineUDMFInt	2	int	
2	GetLineUDMFFixed	2	fixed	
3	GetThingUDMFInt	2	int	
4	GetThingUDMFFixed	2	fixed	
5	GetSectorUDMFInt	2	int	
6	GetSectorUDMFFixed	2	fixed	
7	GetSideUDMFInt	3	int	
8	GetSideUDMFFixed	3	fixed	
9	GetActorVelX	1	int	
10	GetActorVelY	1	int	
11	GetActorVelZ	1	int	
12	SetActivator	1	bool	
13	SetActivatorToTarget	1	bool	
14	GetActorViewHeight	1	int	
15	GetChar	2	char	X
16	GetAirSupply	1	int	
17	SetAirSupply	2	bool	
18	SetSkyScrollSpeed	2	bool	
19	GetArmorType	2	int	
20	SpawnSpotForced	4	bool	
21	SpawnSpotFacingForced	3	bool	
22	CheckActorProperty	3	bool	
23	SetActorVelocity	6	void	
24	SetUserVariable	3	int	
25	GetUserVariable	2	int	
26	Radius_Quake2	6	void	
27	CheckActorClass	2	bool	
28	SetUserArray	4	int	
29	GetUserArray	3	int	
30	SoundSequenceOnActor	2	void	
31	SoundSequenceOnSector	3	void	
32	SoundSequenceOnPolyobj	2	void	