ACS

Introduction

Scripting with Vavoom is just like scripting for Hexen, and Hexen's acs utility is even used to compile scripts. The only significant difference is that scripts should include **vcommon.acs** instead of "common.acs" so that they can get definitions appropriate for Vavoom instead of Hexen. This file just includes three others:

- vspecial.acs
- vdefs.acs
- vwvars.acs

The Hexen Script Language is called "Action Code Script", or "ACS".

Each map has an ACS file that contains the scripts specific to that map. The scripts within it are identified using numbers that the general special ACS_Execute uses.

A script itself can call the ACS_Execute special, which will spawn (start) another script that will run concurrently (at the same time) with the rest of the scripts.

A script can also be declared as "OPEN", which will make it run automatically upon entering the map. This is used for perpetual type effects, level initialization, etc.

The compiler takes the ACS file and produces and object file that is the last lump in the map WAD (BEHAVIOR).

To create a compiled ACS file from a text script from DOS type: C:\HEXEN> ACS filename [enter]

Script Shared Structure

Map scripts should start with **#include "vcommon.acs"**, which is just...

```
#include "vspecial.acs"
#include "vdefs.acs"
#include "vwvars.acs"
```

- The file vspecial.acs defines all the general specials. These are used within scripts just like function calls.
- The file vdefs.acs defines a bunch of constants that are used by the scripts.
- The file vwvars.acs defines all the world variables. It needs to be included by all maps so they use consistent indexing.

Variables and their Scope

There is only one data type ACS, a 4 byte integer. Use the keyword **int** to declare an integer variable. You may also use the keyword **str**, it's used to indicate that you'll be using the variable as a string. The compiler doesn't use string pointers, it uses string handles, which are just integers.

Declaring a variable

There are two "types" of variables:

1. str
2. int

A third type of informal variable is a fixed-point number. This type is the same as int (and is identified as an int), except

that half the number is treated as a fractional value. To convert from an int to a fixed, shift the number left by 16 (<< 16). To convert from a fixed to an int, shift the number right by 16 (>> 16). If you use a decimal point in a number (e.g. 35.0 instead of 35), acc will automatically create the correct fixed point value for you. ACS includes some commands specifically for performing fixed point math. examples:

```
str mystring;
int myint;
```

or:

```
str texture, sound;
int i, tid;
```

Note: You can't assign a variable in its declaration; you must give it a value in a different expression.

The SCOPE of a variable is one of the following:

1. World-scope
2. Map-scope
3. Script-scope

World-scope

World-scope variables are global, and can be accessed in any map. Hexen maintains 64 permanent globals, numbered 0-63. You must assign one of the globals a name in order to access it, like this:

```
world int 5:Grunt;
```

This tells Hexen to reference world global number 5 whenever it encounters the name "Grunt".

Map-scope

Map-scope variables are local to the current map. They must be declared outside of any script code, but without the world keyword. These variables can't be accessed in any other map.

Script-scope

Script-scope variables are local to the current script – they can't be accessed by any other script or map.

Here's some code that shows the declaration of all three scopes:

```
world int 3:DungeonAccess; // World-scope
int mapTimer; // Map-scope

script 4 (void)
{
    int x, y; // Script-scope
    ...
}
```

Syntax

Keywords

The following identifiers are reserved for use as keywords, and may not be used otherwise:

- 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

Comments

Comments are ignored by the script compiler. There are two forms:

1. /*...your comment... */
   - All information between the first /* and last */ is ignored. The leading /* and trailing */ are required.
2. // your comment
   - All information past the // on the current line is ignored

Examples:

```
/*
   This is a comment.
*/
```

```
int a; // And this is a comment
```

Variable definitions

World-variable definitions

world int <constant-expression> : <identifier> ;

world int <constant-expression> : <identifier> , ... ;

Map-variable definitions

Declares a variable local to the current map.

int <identifier> ;
str <identifier> ;
int <identifier> , ... ;

Include Directive

Includes the source of the specified file and compiles it. This acts the same as if you have “included” the source in the file it resides in. Use this to make a common reference set of code you use often.

#include <string-literal>

The supplied required includes shown earlier are an illustration:

```
#include "specials.acs"
#include "defs.acs"
#include "wvars.acs"
```

Define Directive

Replaces an identifier with a constant expression.

```
#define <identifier> <constant-expression>
```

Whenever “identifier” is used in the source, the “constant-expression” is substituted. This is similar to a macro or keyboard short-cut.

Constant Expressions

<integer-constant>:

decimal (200)
hexadecimal (0x00a0, 0x00A0)

fixedpoint(32.0, 0.5, 103.329)

any radix <radix>_digits :

binary (2_01001010)
octal (8_072310)
decimal (10_50025)

hexadecimal (16_00a03f2)

String Literals

<string-literal>:

“string”

Example: “Hello there”

Script Definitions

To define a script:

<script-definition>:

script <constant-expression> (<arglist>) { <statement> }

script <constant-expression> OPEN { <statement> }

For example:

```
script 10 (void) { ... }
script 5 OPEN { ... }
```

Note: OPEN scripts do not take arguments.

Statements

<statement>:

<declaration-statement> <assignment-statement> <compound-statement> <switch-statement> <jump-statement> <selection-statement> <iteration-statement> <function-statement> <linespecial-statement> <print-statement> <control-statement>

Declaration Statements

Declaration statements create script variables.

<declaration-statement>:
int <variable> ;
int <variable> , <variable> , ... ;

Assignment Statements

Assigns an expression to a variable.

<assignment-statement>:

<variable> <assignment-operator> <expression> ; <assignment-operator>:

=
+=
-=
*=
/=
%=

Note: An assignment of the form V <op>= E is equivalent to V = V <op> E.

For example:

```
A += 5;
```

is the same as

```
A = A + 5;
```

Compound Statements

<compound-statement>:

{ <statement-list> }

<statement-list>:

<statement>

<statement-list> <statement>

Switch Statements

A switch statement evaluates an integral expression and passes control to the code following the matched case.

<switch-statement>:
switch (<expression>) { <labeled-statement-list> }

<labeled-statement>:
case <constant-expression> : <statement>

default : <statement>

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.
}
```

Note for C users: While C only allows integral expressions in a switch statement, ACS allows full expressions such as “a + 10”.

Jump Statements

A jump statement passes control to another portion of the script.

<jump-statement>:
continue;
break;
restart;

Iteration Statements

<iteration-statement>:
while (<expression>) <statement>
until (<expression>) <statement>

do <statement> while (<expression>) ;
do <statement> until (<expression>) ;
for (<assignment-statement> ; <expression> ; <assignment-statement>) <statement>

The continue, break and restart keywords can be used in an iteration statement:

- the continue keyword jumps to the end of the last <statement> in the iteration-statement. The loop continues.
- the break keyword jumps right out of the iteration-statement.

Function Statements

A function statement calls a Hexen internal-function, or a Hexen linespecial-function.

<function-statement>:

<internal-function> | <linespecial-statement>
<internal-function>:

<identifier> (<expression>, ...);

<identifier> (const : <constant-expression> , ...) ;

<linespecial-statement>:

<linespecial> (<expression> , ...) ;

<linespecial> (const : <constant-expression> , ...) ;

Print Statements

<print-statement>:
print (<print-type> : <expression> , ...) ; printbold (<print-type> : <expression> , ...) ;

<print-type>:s (string)
d (decimal)
c (constant)

Selection Statements

<selection-statement>:

if (<expression>) <statement>

if (<expression>) <statement> else <statement>

Control Statements

<control-statement>:
suspend; // suspends the script

terminate; // terminates the script