# VavoomC

**VavoomC** is the game programming language of the Vavoom engine, also referred to as progs. It is used to implement game logic and user interface.

## Variables

### Simple Variables

Here are some examples of instance variable declarations in VavoomC:

```
int a; // Declare an integer variable named "a".
float f; // Declare a floating-point variable named "f".
int Table[64]; // Declare an array of 64 integers named "Table".
int *p; // Declare a pointer to int named "p".
string PlayerName; // Declare a string pointer.
Actor Other; // Declare a variable referencing an actor.
```

Variables can appear in three kinds of places in VavoomC: global variables, which are accessible from everywhere. Instance variables, which apply to an entire object, appear immediately after the class declarations. Local variables appear within a function, and are only active while that function executes.

Here are the basic variable types supported in VavoomC:

- int: A 32-bit integer value.
- bool: A boolean value: either “true” or “false”.
- float: A 32-bit floating point number.
- string: A string of characters.
- name: The name of an item in Vavoom (such as the name of a function, state, class, etc). Names are stored as a 16-bit index into the global name table. Names correspond to simple strings of 1-31 characters. Names are not like strings: strings can be modified dynamically, but names can only take on predefined name values.
- classid: Represents a class. The actual value is a reference to an object describing class, but currently you can’t use it as a Class object.
- Pointers.
- Class references: A variable that refers to another object or actor in the world. Object and actor references are very powerful tools, because they enable you to access the variables and functions of another actor. Object references may also contain a special value called “none”, which is the equivalent of the C “NULL” pointer: it says “this variable doesn’t refer to any object”.
- Structs: Same as C structures.

### Pointers

You can declare a pointer variable like this:

```
int *p; // A pointer to integer.
```

The variable “p” above is a pointer to an integer. Pointers to structs can refer to any object that belongs to a subclass of this struct.

There’s a special pointer constant NULL which points to “nowhere”.

Pointers of type void* also are handled specially – any pointer can be assigned to void* and void* can be assigned to any pointer.

Pointers to classes are not allowed.

### Reference variables

You can declare a variable that refers to an object like this:

```
Actor A; // An actor reference.
```

The variable "A" above is a reference to an object in the Actor class. Such a variable can refer to any object that belongs to a subclass of Actor.

When you have a variable that refers to an actor, you can access that actor's variables, and call its functions. Variables that refer to actors always either refer to a valid actor (any actor that actually exists in the level), or they contain the value "none". none is equivalent to the C/C++ "NULL" pointer. Note that an object or actor reference "points to" another actor or object, it doesn't "contain" an actor or object. The C equivalent of an actor reference is a pointer to an object.

### Arrays

Arrays are declared using the following syntax:

```
int MyArray[20]; // Declares an array of 20 ints.
```

VavoomC supports multidimensional arrays.

## Enumerations

Enumerations exist in VavoomC as a convenient way to declare a bunch of keywords.

Here is a sample code that declares enumerations.

```
// Declare an enumeration, with three values.
enum
{
    CO_Red,
    CO_Green,
    CO_Blue
};
```

## Structs

A VavoomC struct is a way of cramming a bunch of variables together into a new kind of super-variable called a struct. VavoomC structs are just like C structs, in that they can contain any simple variables or arrays.

You can declare a struct as follows:

```
// A structure describing a plane
struct TPlane
{
    TVec normal;
    float dist;
    int __type;
    int __signbits;
    int __reserved1;
    int __reserved2;
};
```

In VavoomC structures can have a parent structure, just like classes. For example:

```
struct sec_plane_t:TPlane
{
    float minz;
    float maxz;
    int pic;
    int __base_pic;
    float xoffs;
    float yoffs;
    int flags;
    int translucency;
};
```

Once you declare a struct, you are ready to start declaring specific variables of that struct type:

```
// Declare a pointer variable of type TPlane.
sec_plane_t *floor;
```

To access a component of a struct, use code like the following.

```
void MyFunction()
{
    // Scroll texture
    floor->xoffs += 8.0;
    floor->yoffs += 4.0;
    // Pass floor to a function.
    SomeFunction(floor);
}
```

### Structure and class prototypes

VavoomC Compiler is a one-pass compiler, that means that if you want to create a pointer to a struct it must be already declared or prototyped. Struct prototype looks like this:

```
struct sec_plane_t;
```

## Classes

A class is declared like this:

```
class MyClass : MyParentClass
    // Class specifiers.
    ;

// Declaration of class variables and functions goes here

defaultproperties
{
    //  Initialisation of properties goes here.
}
```

Here I am declaring a new class named “MyClass”, which inherets the functionality of “MyParentClass”.

Object is the parent class of all objects in Vavoom. Object is an abstract base class, in that it doesn’t do anything useful.

Each class inherets all of the variables and functions from its parent class. It can then add new variable declarations, add new functions (or override the existing functions).

The class declaration can take several optional specifiers that affect the class:

- native: Says "this class uses behind-the-scenes C++ support". Vavoom expects native classes to contain a C++ implementation in the EXE file.
- abstract: Declares the class as an "abstract base class". This prevents the user from spawning actors of this class, because the class isn't meaningful on its own.
- __mobjinfo__(a): Sets the ID number used in editor.
- __scriptid__(a): Sets the ID number used in Thing_Spawn* action specials.

### Class prototypes

VavoomC Compiler is a one-pass compiler, that means that if you want to create a reference to a class, it must be already declared or prototyped. Prototypes look like this:

```
class MyClass;
```

## Expressions

### Constants

In VavoomC, you can specify constant values of nearly all data types:

- Integer constants are specified with simple numbers, for example: 123
- If you must specify an integer constant in hexidecimal format, use i.e.: 0x123
- Floating point constants are specified with decimal numbers like: 456.789
- String constants must be enclosed in double quotes, for example: "MyString"
- Name constants must be enclosed in single quotes, for example 'MyName'
- Vector constants contain X, Y, and Z values like this: vector(1.0,2.0,4.0)
- The "NULL" constant points to "nothing".
- The "none" constant refers to "no object".
- The "self" constant refers to "this object", i.e. the object whose script is executing.

### Expressions

To assign a value to a variable, use "=" like this:

```
void Test(void)
{
    int i;
    float f;
    string s;
    name n;
    TVec v, q;

    i = 10; // Assign a value to integer variable i.
    f = 2.7; // Assign a value to floating-point variable f.
    s = "Hello!"; // Assign a value to string variable s.
    n = 'John'; // Assign a value to name variable n.
    v = q; // Copy value of vector q to v.
}
```

VavoomC is a strongly typed language, that means that attempts to assign a value of incompatible type will result in compiler error.

### Built-in operators and their precedence

VavoomC provides a wide variety of C/C++/Java-style operators for such operations as adding numbers together, comaring values, and incrementing variables. Note that all of the operators have the same precedence as they do in C.

| Operator | Types it applies to | Meaning |
|---|---|---|
| *= | int, float, vector | Multiply and assign |
| /= | int, float, vector | Divide and assign |
| \|\| | bool | Logical or |
| && | bool | Logical and |
| & | int | Bitwise and |
| \| | int | Bitwise or |
| ^ | int | Bitwise exlusive or |
| != | All | Compare for inequality |
| == | All | Compare for equality |
| < | int, float | Less than |
| > | int, float | Greater than |
| <= | int, float | Less than or equal to |
| >= | int, float | Greater than or equal to |
| << | int | Left shift |
| >> | int | Right shift |
| % | int | Modulo (remainder after division) |
| * | int, float, vector | Multiply |
| / | int, float, vector | Divide |

The above table lists the operators in order of precedence (with operators of the same precedence grouped together). When you type in a complex expression like "1*2+3*4", VavoomC automatically groups the operators by precedence. Since multiplication has a higher precedence than addition, the expression is evaluated as "(1*2)+(3*4)".

The "&&" (logical and) and "||" (logical or) operators are short-circuited: if the result of the expression can be determined solely from the first expression (for example, if the first argument of && is false), the second expression is not evaluated.

In addition, VavoomC supports the following unary operators:

- ! (bool) Logical not.
- – (int, float) negation.
- ~ (int) bitwise negation.
- ++, — Decrement (either before or after a variable).

## Functions

### Declaring Functions

VavoomC supports only class member functions.

In VavoomC, you can declare new functions and write new versions of existing functions. Functions can take one or more parameters, and can optionally return a value. The parameter and return value type mus be of size 4 (i.e. integers, floats, pointers, references) or vectors. Some functions are implemented in C++, they are called builtin functions.

Here are some simple function declarations:

```
class MyClass : MyParentClass;

int MyVariable;

int GetMyVariable()
{
   return MyVariable;
}

defaultproperties
{
}
```

When a function is called, the code within the brackets is executed. Inside the function, you can declare local variables, and execute any VavoomC code. The optional "return" keyword causes the function to immediately return a value.

Warning: Local variables you declare in a function are not initialised.

Function calls can be recursive. For example, the following function computes the factorial of a number:

```
// Function to compute the factorial of a number.
int Factorial(int Number)
{
   if (Number <= 0)
      return 1;
   else
      return Number * Factorial(Number - 1);
}
```

### Function overriding

"Function overriding" refers to writing a new version of a function in a subclass.

To override a function, just cut and paste the function definition from the parent class into your new class. For example, for OnMapSpawn, you could add this to your Demon class.

```
// New Demon class version of the OnMapSpawn function.
void OnMapSpawn(mthing_t *mthing)
{
   // If monsters are disabled, then destroy this actor immediately
   if (nomonsters)
   {
      RemoveMobjThinker(this);
      return;
   }
   // Call parent class version of OnMapSpawn
   ::OnMapSpawn(mthing);
}
```

Function overriding is the key to creating new VavoomC classes efficiently. You can create a new class that expands on an existing class. Then, all you need to do is override the functions which you want to be handled differently. This enables you to create new kinds of objects without writing gigantic amounts of code.

### Advanced function specifiers

- native: You can declare VavoomC functions as “native”, which means that the function is callable from VavoomC, but is actually written (elsewhere) in C++. For example:

```
native float sin(float angle);
```

- static: Says that function has no self object. Currently only native functions can be static.

## Control statements

VavoomC supports all the standard flow-control statements of C/C++/Java.

### For Loops

“for” loops let you cycle through a loop as long as some condition is met.

For example:

```
// Example of "for" loop.
void ForExample(void)
{
    int i;
    print("Demonstrating the for loop");
    for (i = 0; i < 4; i++)
    {
        print("The value of i is %d\n", i);
    }
    print("Completed with i=%d\n", i);
}
```

The output of this loop is:

```
Demonstrating the for loop
The value of i is 0
The value of i is 1
The value of i is 2
The value of i is 3
Completed with i=4
```

In a “for” loop, you must specify three expressions separated by semicolons. The first expression is for initializing a variable to its starting value. The second expression gives a condition which is checked before each iteration of the loop executes; if this expression is true, the loop executes. If it’s false, the loop terminates. The third condition gives an expression which increments the loop counter.

Though most “for” loop expressions just update a counter, you can also use “for” loops for more advanced things like traversing linked lists, by using the appropriate initialization, termination, and increment expressions.

In all of the flow control statements, you can either execute a single statement, without brackets, as follows:

```
for (i = 0; i < 4; i++)
    print("The value of i is %d", i);
```

Or you can execute multiple statements, surrounded by brackets, like this:

```
for (i = 0; i < 4; i++)
{
    print("The value of i is");
    print("%d\n", i);
}
```

### Do-While Loops

“do”-“while” loops let you cycle through a loop while some ending expression is true.

```
// Example of "do" loop.
void DoExample(void)
{
    int i;
    print("Demonstrating the do loop");
    i = 0;
    do
    {
        print("The value of i is %d\n", i);
        i = i + 1;
    } while (i < 4);
    print("Completed with i=%d\n", i);}
```

The output of this loop is:

```
Demonstrating the do loop
The value of i is 0
The value of i is 1
The value of i is 2
The value of i is 3
Completed with i=4
```

### While Loops

“While” loops let you cycle through a loop while some starting expression is true.

```
// Example of "while" loop.
void WhileExample(void)
{
    int i = 0;

    print("Demonstrating the while loop");
    while (i < 4)
    {
        print( "The value of i is %d\n", i);
        i = i + 1;
    }
    print("Completed with i=%d\n", i);}
```

The output of this loop is:

```
Demonstrating the do loop
The value of i is 0
The value of i is 1
The value of i is 2
The value of i is 3
Completed with i=4
```

### Break

The “break” command exits out of the nearest loop (“for”, “do”, or “while”).

```
// Example of "while" loop.
void WhileExample(void)
{
    int i;

    print("Demonstrating break");
    for (i = 0; i < 10; i++)
    {
        if (i == 3)
            break;
        print("The value of i is %d\n", i);
    }
    print("Completed with i=%d\n", i);
}
```

The output of this loop is:

```
Demonstrating break
The value of i is 0
The value of i is 1
The value of i is 2
Completed with i=3
```

### Conditional Statements

“if” and “else” let you execute code if certain conditions are met.

```
// Example of simple "if".
if (LightBrightness < 20)
    print("My light is dim\n");

// Example of "if-else".
if (LightBrightness < 20)
    print("My light is dim\n");
else
    print("My light is bright\n");
```

### Case Statements

“switch”, “case”, “default”, and “break” let you handle lists of conditions easily.

```
// Example of switch-case.
void TestSwitch(void)
{
    // Executed one of the case statements below, based on
    // the value in LightType.
    switch (LightType)
    {
    case LT_None:
        print("There is no lighting\n");
        break;
    case LT_Steady:
        print("There is steady lighting\n");
        break;
    case LT_Backdrop:
```

```
        print("There is backdrop lighting\n");
        break;
    default:
        print("There is dynamic\n");
        break;
    }
}
```

A "switch" statement consists of one or more "case" statements, and an optional "default" statement. After a switch statement, execution goes to the matching "case" statement if there is one; otherwise execution goes to the "default" statement; otherwise execution continues past the end of the "select" statement.

After you write code following a "case" label, you must use a "break" statement to cause execution to go past the end of the "switch" statement. If you don't use a "break", execution "falls through" to the next "case" handler.

## Class Reference

Here goes the default progs class list, as is in Vavoom 1.28. You most probably will use and/or edit at least several of these classes when creating a new mod for Vavoom.

Classes are divided by a number of sections. The **common** section enlists classes at the top of class hierarchy that every Vavoom mod must use, this section is divided in sub-sections the **engine** sub-section defines classes that are *native* to the core of the engine. The **linespec** sub-section shows classes that are shared by all the games and related to the line, sector and thing specials available for ACS and map editing, they also have the most basic algorithms used by actor AI. There's one last sub-section called **uibase** that contains the menu API methods for drawing the status bar and menus for every game. Finally, game sections (**Doom**, **Doom2**, **Heretic** etc) have a list of classes which only belong to named games' progs correspondingly.

### Common classes

#### Engine (engine)

- ActorDisplayWindow
- BasePlayer
- ClientGameBase
- Entity
- GameInfo
- Level
- LevelInfo
- Object
- PlayerReplicationInfo
- RootWidget
- ScriptsParser
- Thinker
- Widget
- WorldInfo

#### Line Specials (linespec)

- Actor.DoomAttack
- Actor.DoomSpecific
- Actor.FlagsAndAppearance
- Actor.FreezeDeath
- Actor.GenericAttacks
- Actor.Inventory
- Actor.Misc
- Actor.MissileMovement
- Actor.MonsterAi

- Actor.Sound
- Actor.Spawn
- Actor.Special
- Actor.StateJump
- Actor.States
- ActorMover
- AimingCamera
- AmbientSound
- Ammo
- BackpackItem
- Bang4Cloud
- BasicArmor
- BasicArmorBonus
- BasicArmorPickup
- BossBrain
- BossEye
- BossTarget
- BrainState
- CeilingMover
- CeilingWaggle
- Coin
- CustomInventory
- DoomBuilderCamera
- DynamicLight
- Elevator
- EntityEx.AiUtils
- EntityEx.Damage
- EntityEx.Defaults
- EntityEx.Head
- EntityEx.Inventory
- EntityEx.LineAttack
- EntityEx.Misc
- EntityEx.Physics
- EntityEx.SpawnMissile
- FakeInventory
- Feather
- FireDroplet
- FireFlicker
- FlashFader
- FloorMover
- FloorWaggle
- GlassShard
- GlowingLight
- Gold10
- Gold25
- Gold50
- HateTarget
- Health
- HealthPickup
- HexenArmor
- IceChunk
- IceChunkHead
- InterpolationPoint
- InterpolationSpecial
- Inventory
- InvisibleBridge
- InvisibleBridge8

- InvisibleBridge16
- InvisibleBridge32
- Junk
- Key
- LightEffect
- LightFlash
- Lighting
- LineSpecialClientGame
- LineSpecialGameInfo
- LineSpecialLevelInfo
- LookAtCamera
- LowerStackLookOnly
- MapRevealer
- Meat
- MiniMissilePuff
- MorphProjectile
- ParticleFountain
- PathFollower
- PatrolSpecial
- PhasedLight
- PickupFlash
- Pillar
- PlaneWatcher
- Platform
- PlayerChunk
- PlayerEx
- PlayerPawn
- PointLight
- PointLightFlicker
- PointLightFlickerRandom
- PointLightPulse
- PointPuller
- PointPusher
- PolyobjDoor
- PolyobjMover
- PolyobjRotator
- PolyobjThinker
- PowerFlight
- PowerGhost
- PowerInvisibility
- PowerInvulnerable
- PowerIronFeet
- PowerLightAmp
- PowerMask
- PowerMinotaur
- PowerScanner
- PowerShadow
- PowerSpeed
- PowerStrength
- PowerTargeter
- PowerTorch
- Powerup
- PowerupGiver
- PowerWeaponLevel2
- Pusher
- PuzzleItem
- QuakeFocus

- RocketTrail
- Scroller
- SecActEnter
- SecActExit
- SecActEyesAboveC
- SecActEyesBelowC
- SecActEyesDive
- SecActEyesSurface
- SecActHitCeiling
- SecActHitFakeFloor
- SecActHitFloor
- SecActUse
- SecActUseWall
- SecretTrigger
- SectorAction
- SectorMover
- SectorPointLight
- SectorSilencer
- SectorThinker
- SecurityCamera
- SkyPicker
- SkyViewpoint
- SoundEnvironment
- SoundSequence
- SoundSequenceSlot
- Spark
- SpawnFire
- SpawnShot
- StackPoint
- StairStepMover
- StaticLightSource
- StaticRGBLightSource
- StrifeHumanoid
- StrifePuff
- Strobe
- SwitchableDecoration
- SwitchingDecoration
- TeleportDest
- TeleportDest2
- TeleportDest3
- TextureChangeDoor
- UpperStackLookOnly
- VerticalDoor
- WallLightTransfer
- WaterZone
- Weapon
- WeaponHolder
- WeaponPiece
- WorldInfoEx

**UI Base (uibase)**

- ClientGameShared
- ConDialog
- ConDlgChoice
- FinaleBackground
- FinaleScreen

- HUDMessage
- HUDMessageFadeInOut
- HUDMessageFadeOut
- HUDMessageTypeOnFadeOut
- IntermissionBackground
- MenuBigTextButton
- MenuChoice
- MenuChoiceEnum
- MenuChoiceEpisode
- MenuChoicePClass
- MenuChoiceSkill
- MenuChoiceSlider
- MenuChoiceSlot
- MenuChoice_LoadSlot
- MenuChoice_OnOff
- MenuChoice_SaveSlot
- MenuControlKey
- MenuInputLine
- MenuModel
- MenuSaveSlot
- MenuScreen
- MenuScreenAdvancedVideoOptions
- MenuScreenClass
- MenuScreenControls
- MenuScreenEpisode
- MenuScreenJoinGame
- MenuScreenLoadGame
- MenuScreenMasterList
- MenuScreenMouseOptions
- MenuScreenSaveGame
- MenuScreenScreenResolution
- MenuScreenSinglePlayer
- MenuScreenSkill
- MenuScreenSList
- MenuScreenSoundOptions
- MenuScreenVideoOptions
- MenuSelector_Big
- MenuSelector_SmallLeft
- MenuSelector_SmallRight
- MenuSList
- MenuSmallTextButton
- MenuSpriteAnim
- MenuStaticAnim
- MenuStaticBitmap
- MenuTextButton
- MenuTitleText
- StatusBarShared

### Game classes

Since Vavoom engine added support for DECORATE, most of the game classes were ported, but their actions are still defined in VavoomC language inside of the Actor.Game classes, they are listed here.

**Doom**

- Cyberdemon
- DeadMarine

- GibbedMarine
- GibbedMarineExtra
- DeadZombieMan
- DeadShotgunGuy
- DeadDoomImp
- DeadDemon
- DeadCacodemon
- DeadLostSoul
- Demon
- Spectre
- Clip
- ClipBox
- RocketAmmo
- RocketBox
- Cell
- CellPack
- Shell
- ShellBox
- Backpack
- GreenArmor
- BlueArmor
- ArmorBonus
- Soulsphere
- InvulnerabilitySphere
- BlurSphere
- RadSuit
- Infrared
- Allmap
- MegasphereHealth
- Megasphere
- Berserk
- TechLamp
- TechLamp2
- Column
- TallGreenColumn
- ShortGreenColumn
- TallRedColumn
- ShortRedColumn
- SkullColumn
- HeartColumn
- EvilEye
- FloatingSkull
- TorchTree
- BlueTorch
- GreenTorch
- RedTorch
- ShortBlueTorch
- ShortGreenTorch
- ShortRedTorch
- Stalagtite
- TechPillar
- Candlestick
- Candelabra
- BloodyTwitch
- Meat2
- Meat3
- Meat4

- Meat5
- NonsolidMeat2
- NonsolidMeat4
- NonsolidMeat3
- NonsolidMeat5
- NonsolidTwitch
- HeadsOnAStick
- HeadOnAStick
- HeadCandles
- DeadStick
- LiveStick
- BigTree
- BurningBarrel
- HangNoGuts
- HangBNoBrain
- HangTLookingDown
- HangTSkull
- HangTLookingUp
- HangTNoBrain
- ColonGibs
- SmallBloodPool
- BrainStem
- HealthBonus
- Stimpack
- Medikit
- DoomImp
- DoomImpBall
- DoomKey
- BlueCard
- YellowCard
- RedCard
- BlueSkull
- YellowSkull
- RedSkull
- ExplosiveBarrel
- BulletPuff
- DoomUnusedStates
- DoomPlayer
- DoomWeapon
- Fist
- Chainsaw
- Pistol
- Shotgun
- Chaingun
- RocketLauncher
- Rocket
- PlasmaRifle
- PlasmaBall
- BFG9000
- BFGBall
- BFGExtra
- LostSoul
- ZombieMan
- ShotgunGuy
- MarineFist
- MarineBerserk
- MarineChainsaw

- MarinePistol
- MarineShotgun
- MarineSSG
- MarineChaingun
- MarineRocket
- MarinePlasma
- MarineRailgun
- MarineBFG
- SpiderMastermind
- StealthBaron
- StealthCacodemon
- StealthDemon
- StealthDoomImp
- StealthShotgunGuy
- StealthZombieMan

**Doom2**

Doom2 shares all classes from Doom, here are the Doom2 specific classes.

- Arachnotron
- ArachnotronPlasma
- Archvile
- ArchvileFire
- BossBrain
- BossEye
- BossTarget
- SpawnShot
- SpawnFire
- HellKnight
- SuperShotgun
- Fatso
- FatShot
- CommanderKeen
- PainElemental
- ChaingunGuy
- WolfensteinSS
- Revenant
- StealthArachnotron
- StealthArchvile
- StealthChaingunGuy
- StealthHellKnight
- StealthFatso
- StealthRevenant

**Heretic**

- Beast
- BeastBall
- Puffy
- Beak
- BeakPowered
- BeakPuff
- Chicken
- Feather
- Clink
- Sorcerer1

- SorcererFX1
- Sorcerer2FX1
- Sorcerer2FX2
- Sorcerer2FXSpark
- Sorcerer2Telefade
- GoldWandAmmo
- GoldWandHefty
- MaceAmmo
- MaceHefty
- CrossbowAmmo
- CrossbowHefty
- SkullRodAmmo
- SkullRodHefty
- PhoenixRodAmmo
- PhoenixRodHefty
- BlasterAmmo
- BlasterHefty
- BagOfHolding
- SilverShield
- EnchantedShield
- SuperMap
- ArtiInvisibility
- ActivatedTimeBomb
- SkullHang70
- SkullHang60
- SkullHang45
- SkullHang35
- Chandelier
- SerpentTorch
- SmallPillar
- StalagmiteSmall
- StalagmiteLarge
- StalactiteSmall
- StalactiteLarge
- FireBrazier
- Barrel
- BrownPillar
- Moss1
- Moss2
- WallTorch
- HangingCorpse
- HereticImp
- HereticImpLeader
- HereticImpChunk1
- HereticImpChunk2
- HereticImpBall
- HereticKey
- KeyGreen
- KeyBlue
- KeyYellow
- KeyGizmoFloatBlue
- KeyGizmoBlue
- KeyGizmoFloatGreen
- KeyGizmoGreen
- KeyGizmoFloatYellow
- KeyGizmoYellow
- Pod

- PodGoo
- PodGenerator
- TeleGlitterGenerator1
- TeleGlitterGenerator2
- TeleGlitter1
- TeleGlitter2
- Volcano
- VolcanoBlast
- VolcanoTBlast
- HereticPlayer
- BloodySkull
- Staff
- StaffPowered
- StaffPuff
- StaffPuff2
- Gauntlets
- GauntletsPowered
- GauntletPuff1
- GauntletPuff2
- GoldWand
- GoldWandPowered
- GoldWandFX1
- GoldWandFX2
- GoldWandPuff1
- GoldWandPuff2
- Crossbow
- CrossbowPowered
- CrossbowFX1
- CrossbowFX2
- CrossbowFX3
- CrossbowFX4
- Blaster
- BlasterPowered
- BlasterPuff
- BlasterSmoke
- SkullRod
- SkullRodPowered
- HornRodFX1
- PhoenixPuff
- MacePowered
- MaceSpawner
- MaceFX1
- MaceFX2
- MaceFX3
- Ironlich
- HeadFX1
- HeadFX2
- HeadFX3
- Knight
- KnightGhost
- KnightAxe
- RedAxe
- Mummy
- MummySoul
- MummyLeader
- MummyFX1
- MummyGhost

- MummyLeaderGhost
- Snake
- SnakeProjA
- SnakeProjB
- Wizard
- WizardFX1

**Hexen**

**Strife**

## Entity

Object -> Thinker -> Entity
defined in *progs|common|shared|Entity.vc*

**Declaration**

```
class Entity
  native
  abstract;
```

**Abstract**

Class Entity is the parent class for all the positioned map objects; thus it may be said that its first extension to parent Thinker class are coordinates in world. According to them Entity determines map objects' collisions, centers of objects' images and sound sources.

Since Entity class is *abstract*, you cannot create objects of this type, only of its non-abstract descendants.

**Member variables**

- Angles
- Args
- bAvoidingDropoff
- bBlasted
- bCantLeaveFloorpic
- bCheckLineBlocking
- bCheckLineBlockMonsters
- bColideWithThings
- bColideWithWorld
- bCorpse
- bDropOff
- bFixedModel
- bFloat
- bFloorClip
- bFly
- bHidden
- bIgnoreCeilingStep
- bIsPlayer
- bNoBlockmap
- bNoGravity
- bNoPassMobj
- bOnMobj
- bSolid
- BlockMapNext
- BlockMapPrev

- Ceiling
- CeilingZ
- DropOffZ
- Effects
- Floor
- FloorClip
- FloorZ
- Health
- Height
- InUse
- Mass
- MaxDropoffHeight
- MaxStepHeight
- ModelFrame
- ModelIndex
- ModelSkinIndex
- ModelSkinNum
- NetID
- NextState
- Origin
- Player
- Radius
- SoundClass
- SoundGender
- Special
- SpriteFrame
- SpriteIndex
- SpriteName
- SpriteType
- State
- StateTime
- SubSector
- Sector
- TID
- Translation
- Translucency
- ValidCount
- Velocity
- WaterLevel
- WaterType

**Member functions**

- Activate
- ApplyFriction
- BlockedByLine
- BounceWall
- CanSee
- CheckDropOff
- CheckInventory
- CheckOnmobj
- CheckPosition
- CheckRelPosition
- CheckSides
- CheckWater
- CrossSpecialLine
- Deactivate

- Destroyed
- DistTo
- DistTo2
- FindState
- GetGravity
- GetSigilPieces
- GiveInventory
- HandleFloorclip
- InsertIntoTIDList
- LinkToWorld
- PlayFullVolumeSound
- PlaySound
- PushLine
- Remove
- RemoveFromTIDList
- RemoveThing
- RoughCheckThing
- RoughMonsterSearch
- SectorChanged
- SetInitialState
- SetOrigin
- SetState
- SlideMove
- StopSound
- TakeInventory
- TestMobjZ
- Touch
- TryMove
- UnlinkFromWorld
- UpdateVelocity

**Description**

Entities are used to tell the refresh where to draw an image, tell the world simulation when objects are contacted, and tell the sound driver how to position a sound.

The refresh uses the snext and sprev links to follow lists of things in sectors as they are being drawn. The sprite, frame, and angle elements determine which patch_t is used to draw the sprite if it is visible.
The sprite and frame values are allmost allways set from state_t structures. The xyz origin point represents a point at the bottom middle of the sprite (between the feet of a biped). This is the default origin position for patch_ts grabbed with lumpy.exe. A walking creature will have its z equal to the floor it is standing on.

The sound code uses the x,y, and z fields to do stereo positioning of any sound effited by the Entity.

The play simulation uses the BlockLinks, x,y,z, radius, height to determine when mobj_ts are touching each other, touching lines in the map, or hit by trace lines (gunshots, lines of sight, etc). The Entity->flags element has various bit flags used by the simulation.

Every Entity is linked into a single sector based on its origin coordinates. The subsector_t is found with R_PointInSubsector(x,y), and the sector_t can be found with subsector->sector. The sector links are only used by the rendering code, the play simulation does not care about them at all.

Any Entity that needs to be acted upon by something else in the play world (block movement, be shot, etc) will also need to be linked into the blockmap. If the thing has the MF_NOBLOCK flag set, it will not use the block links. It can still interact with other things, but only as the instigator (missiles will run into other things, but nothing can run into a missile). Each block in the grid is 128*128 units, and knows about every line_t that it contains a piece of, and every interactable Entity that has its origin contained.

A valid Entity is a Entity that has the proper subsector_t filled in for its xy coordinates and is linked into the sector from which the subsector was made, or has the MF_NOSECTOR flag set (the subsector_t needs to be valid even if MF_NOSECTOR is set), and is linked into a blockmap block or has the MF_NOBLOCKMAP flag set. Links should only be modified by the P_[Un]SetThingPosition() functions. Do not change the MF_NO? flags while a thing is valid.

## Level

Object -> Level
defined in *progs|common|shared|Level.vc*

**Declaration**

```
class Level:Object
native;
```

**Abstract**

Class Level represents data, which describes a single game level.

**Member variables**

- __BlockLinks
- __BlockMap
- __BlockMapHeight
- __BlockMapLump
- __BlockMapWidth
- __Nodes
- __NoVis
- __NumNodes
- __NumPolyObjs
- __NumSegs
- __NumSubsectors
- __PolyBlockMap
- __PolyObjs
- __RejectMatrix
- __Segs
- __Subsectors
- __ThinkerHead
- __ThinkerTail
- __VisData
- bForServer
- bExtended
- bGLNodesV5
- BlockMapOrgX
- BlockMapOrgY
- GenericSpeeches
- LevelSpeeches
- NumGenericSpeeches
- NumLevelSpeeches
- NumLines
- NumSectors
- NumSides
- NumThings
- NumVertexes Lines
- Sectors
- Sides

- Things
- Vertexes

#### Member functions

- PointInSector

### Object

defined in *progs|common|shared|Object.vc*

#### Declaration

```
class Object
  native
  abstract
```

#### Abstract

Class Object is the base class for all other VavoomC classes. If you create a new class you must derive it from Object class directly or from any other existing class, that have Object class as a top parent.

Since Object class is *abstract*, you cannot create objects of this type, only of its non-abstract descendants.

#### Member variables

- Class
- CxxVTable
- ObjectFlags
- ObjectIndex
- VTable

#### Member functions

- Destroy
- IsA
- IsDestroyed

### Thinker

Object -> Thinker
defined in *progs|common|shared|Thinker.vc*

#### Declaration

```
class Thinker
  native
  abstract;
```

#### Abstract

Class Thinker implements timed events processing and is the parent class for all level objects. These objects are not necessarily visible or interactable by player, their only obligatory peculiarity is that they are linked with current Level and are destroyed when Level is destroyed.

Since Thinker class is *abstract*, you cannot create objects of this type, only of its non-abstract descendants.

#### Member variables

- Level
- Next
- Prev
- XLevel

#### Member functions

- Tick

#### Description

Thinker is a single process manager. For each Thinker linked to Level Tick function is called periodically over time, thus allowing Thinker object to update its state and make any changes necessary.

Object of any class derived directly from Thinker should be created using NewSpecialThinker static function.This will ensure new Thinker is linked to Level properly. In Level object Thinkers are stored in linked list(See Level::__ThinkerHead and Level::__ThinkerTail). They are also accessible from each Thinker object: using Next and Prev references you may cycle through all the list.

Destroy Thinker object using RemoveSpecialThinker static method.

A notice should be made, that Level *class* object is referenced as XLevel from Thinker class, while Level *variable* is a reference to LevelInfo class object.This may cause confusion at first time.

## Static Function Reference

The Class Object declares a vast list of static functions. All of them are native (built in engine). Since they are static, they can be called from any function of any class without associating with a specific object; thus the ymay be thought of as global functions for simplicity.

Built-in functions still are split, however, on three main parts: shared (or common), client and server. Shared functions may be used anywhere, client and server functions have sense only when called from member function of client or server class correspondingly; improper use may cause program failure.

Following a categorized list of static functions.

### Shared

#### Error functions

- void Error (string format, ...)
  - Shut down client and server, go to title
- void FatalError (string format, ...)
  - Exit program with error

#### Console variables functions

- void CreateCvar(name Name, string default_value, int flags)
  - Create a new cvar
- int GetCvar(name Name)
  - Read value
- void SetCvar(name Name, int value)
  - Set cvar value
- float GetCvarF(name Name)
  - Read float value
- float GetCvarF(name Name)

- Read float value
- void SetCvarF(name Name, float value)
  - Set cvar float value
- string GetCvarS(name Name)
  - Read string value
- void SetCvarS(name Name, string value)
  - Set cvar string value

**Math functions**

Value processing

- int abs(int val)
  - Absolute value of an "integer".
- float fabs(float val)
  - Absolute value of a "float".
- float AngleMod180(float angle)
  - Normalises angle in range -180..180
- float AngleMod360(float angle)
  - Normalises angle in range 0..360
- int Clamp(int Val, int Min, int Max)
  - Clamped value: If "Val" <= "Min" returns "Min"; If "Val" >= "Max" returns "Max"; otherwisereturns "Val".
- float FClamp(float Val, float Min, float Max)
  - Like above, but for "floats".
- int Min(int v1, int v2)
  - Minimal value of two "ints".
- float FMin(float v1, float v2)
  - Minimal value of two "floats".
- int Max(int v1, int v2)
  - Maximal value of two "ints".
- float FMax(float v1, float v2)
  - Maximal value of two "floats".
- float sqrt(float x)
  - Square root.

Trigonometry

- float sin(float angle)
  - Sine
- float cos(float angle)
  - Cosine
- float tan(float angle)
  - Tangent
- float asin(float x)
  - Arcsine
- float acos(float x)
  - Arccosine
- float atan(float slope)
  - Arctangent
- float atan2(float y, float x)
  - The atan2 function is useful in many applications involving vectors in Euclidean space, such as "finding the direction from one point to another". — [http://en.wikipedia.org/wiki/Atan2Wikipedia]

Vectors

- void AngleVectors(TAVec * angles, TVec * forward, TVec * right, TVec * up)
  - Creates vectors for given angle vector
- void AngleVector(TAVec * angles, TVec * forward)

    - Simplified version of AngleVectors, creates only forward vector
- TVec CrossProduct(TVec v1, TVec v2)
    - Cross product (perpendicular vector)
- float DotProduct(TVec v1, TVec v2)
    - Dot product
- float GetPlanePointZ(TPlane * plane, TVec point)
    - Get z of point with given x and y coords. Don't try to use it on a vertical plane
- float Length(TVec vec)
    - Vector length
- TVec Normalise(TVec vec)
    - Normalises vector
- int PointOnPlaneSide(TVec point, TPlane * plane)
    - Returns side: 0 (front) or 1 (back).
- TVec RotateDirectionVector(TVec vec, TAVec rot)
    - Rotates a direction vector
- void VectorAngles(TVec * vec, TAVec * angles)
    - Create angle vector for a vector
- void VectorRotateAroundZ(TVec * vec, float angle)
    - Rotates vector around Z axis
- TVec RotateVectorAroundVector(TVec Vector, TVec Axis, float angle)
    - Rotates vector around another vector

**Random numbers**

- float Random()
    - Floating random number 0.0 ... 0.999999
- int P_Random()
    - Integer random number 1 .. 255

**String functions**

- string strcat(string s1, string s2)
    - Append string to string
- int strcmp(string s1, string s2)
    - Compare strings
- int stricmp(string s1, string s2)
    - Compare strings ignoring case
- int strlen(string s)
    - String length
- string strlwr(string s)
    - Convert string to lowercase
- string substr(string Str, int Start, int Len)
    - Gets a substring
- string strupr(string s)
    - Convert string to uppercase
- string va(string format, ...)
    - Does varargs print into a temporary buffer
- bool StrStartsWith(string Str, string Check)
    - Checks if string starts with given string
- bool StrEndsWith(string Str, string Check)
    - Checks if string ends with given string
- string StrReplace(string Str, string Search, string Replacement
    - Replaces substrings with another string

**Printing to console**

- void print(string format, ...)

    - Print to console
- void dprint(string format, ...)
    - Print to console only when developer == 1 (Development mode)

#### Texture / flat number retrieval

- int CheckFlatNumForName(name Name)
- int CheckTextureNumForName(name Name)
- int FlatNumForName(name Name)
- float TextureHeight(int pic)
- int TextureNumForName(name Name)
- name GetTextureName(int Handle)

#### Message IO functions

- MSG_CheckSpace
- MSG_ReadByte
- MSG_ReadChar
- MSG_ReadLong
- MSG_ReadShort
- MSG_ReadWord
- MSG_Select
- MSG_WriteByte
- MSG_WriteLong
- MSG_WriteShort

#### Type conversions

- int ftoi(float f)
    - float -> int
- float itof(int f)
    - int -> float
- int atoi(string str)
    - Converts string to integer
- float atof(string str)
    - Converts string to float
- name StrToName(string Str)
    - Converts string to name

#### Console command functions

- int Cmd_CheckParm(string str)
- void CmdBuf_AddText(string format, ...)
    - Adds text to command buffer, same as typing it in console

#### Class methods

- class FindClass(name Name)
- class FindClassLowerCase(name Name)
- bool ClassIsChildOf(class SomeClass, class BaseClass)
- name GetClassName(class C)
- class GetClassParent(class C)
- class GetClassReplacement(class C)
- class GetClassReplacee(class C)
- state FindClassState(class C, name StateName)
- int GetClassNumOwnedStates(class C)
- state GetClassFirstState(class C)

#### State methods

- bool StateIsInRange(state State, state Start, state End, int MaxDepth)
- bool StateIsInSequence(state State, state Start)
- name GetStateSpriteName(state State)
- float GetStateDuration(state State)
- state GetStatePlus(state S, int Offset, optional bool IgnoreJump)
- bool AreStateSpritesPresent(state State)

#### Iterators

- iterator AllObjects(class BaseClass, out Object Obj)
- iterator AllClasses(class BaseClass, out class Class)

#### Misc

- string Info_ValueForKey(string info, string key)
  - Reads key value from info string (userinfo or serverinfo)
- bool WadLumpPresent(name Name)
  - Checks if WAD lump is present, used to check for shareware/extended WADs
- Object SpawnObject(class cid)
- GameObject::VAnimDoorDef* FindAnimDoor(int BaseTex)
- string GetLangString(name Id)
- int RGB(byte r, byte g, byte b)
- int RGBA(byte r, byte g, byte b, byte a)
- GameObject::LockDef* GetLockDef(int Lock)
- int ParseColour(string Name)
- string TextColourString(int Colour)
- bool StartTitleMap()
- void LoadBinaryLump(name LumpName, out array<byte> Array)
- boolIsMapPresent(nameMapName:)
- void Clock(int Idx)
- void Unclock(int Idx)

### Client

#### Graphics

- void SetVirtualScreen(int Width, int Height)
- int R_RegisterPic(name Name)
  - Registers a graphic, returns handle
- int R_RegisterPicPal(name Name, name palname)
  - Registers a graphic with custom palette, returns handle
- void R_GetPicInfo(int handle, picinfo_t * info)
  - Retrieves pic info
- void R_DrawPic(int x, int y, int handle)
  - Draws a pic
- void R_InstallSprite(string Name, int index)
  - Installs a sprite model
- void R_DrawSpritePatch(int x, int y, int sprite, int frame, int rot, int TranslStart, int TranslEnd, int Colour)
  - Draws a sprite
- void* InstallModel(string Name)
  - Installs model
- void R_DrawModelFrame(TVec origin, float angle, void* model, int frame, int nextframe, string skin, intTranslStart, int TranslEnd, int Colour)
  - Draws a model
- void R_FillRect(int x, int y, int w, int h, int colour)
  - Draws a coloured rectangle

#### Client Sound

- void LocalSound(name Name)
  - Plays a sound
- bool IsLocalSoundPlaying(name Name)
  - Checks if sound is still playing.
- void StopLocalSounds()
  - Stops all local sounds.

#### Other

- string TranslateKey(int c)Handle shift+key
- string P_GetMapName(int map)
- name P_GetMapLumpName(int map)
- name P_TranslateMap(int map)
- int P_GetNumEpisodes()
- EpisodeDef* P_GetEpisodeDef(int i)
- int P_GetNumSkills()
- SkillDef* P_GetSkillDef(int i)
- bool SV_GetSaveString(int i, string* buf)
- void StartSearch(bool Master)
- slist_t * GetSlist()
- string KeyNameForNum(int KeyNr)
- void IN_GetBindingKeys(stringcmd:, int *key1, int *key2)
- void IN_SetBinding(int key, string ondown, string onup)* string LoadTextLump(name Name)

### Server

#### Map utilites

- GameObject::opening_t *LineOpenings(GameObject::line_t * linedef, TVec point)
- int P_BoxOnLineSide(float *tmbox, GameObject::line_t * ld)
  - Returns 0 – front, 1 – back, -1 – on
- GameObject::sec_region_t *FindThingGap(GameObject::sec_region_t * gaps, TVec point, float z1, floatz2)
  - Find the best gap that the thing could fit in, given a certain Z*, position (z1 is foot, z2 ishead).
- GameObject::opening_t *FindOpening(GameObject::opening_t * gaps, float z1, float z2)
  - Find the best opening
- GameObject::sec_region_t *PointInRegion(GameObject::sector_t * sector, TVec p)
  - Find best region in sector for a given point

#### Server Sound functions

- bool GetSoundPlayingInfo(Entity mobj, int sound_id)
- int GetSoundID(name Name)
- void SetSeqTrans(name Name, int Num, int SeqType)
- name GetSeqTrans(int Num, int SeqType)
- name GetSeqSlot(name Sequence)
- GameObject::VTerrainInfo
- TerrainType(int pic)
- GameObject::VSplashInfo
- GetSplashInfo(name Name)
- GameObject::VTerrainInfo
- GetTerrainInfo(name Name)
- void SB_Start()
- class FindClassFromEditorId(int Id, int GameFilter)
- class FindClassFromScriptId(int Id, int GameFilter)