

## Blades of Avernum Editor Appendices

The appendices contain lists of the info you will need to write your scenarios. There are lists of spells, skills, animation effects, and, most importantly all of the calls you can use in your scripts.

### Table of Contents:

Wall and Cliff Types  
Character Skills  
Mage Spells  
Priest Spells  
Alchemy Recipes  
Character Traits  
Status Types  
Incidental Town Noises  
Missile Animations  
Effects  
Booms  
Sparkles  
Scenario Icons  
Sound Effects  
The Calls

## Wall and Cliff Types

Towns have two sorts of walls. On the Town Details window, you can select the graphics sheet that will be used for each of the two wall types. A number of premade wall graphics are provided. To access them, enter these values for "Wall 1 Sheet" and "Wall 2 Sheet":

**Full Wall Set** (All wall terrains, including doors, gates, windows, etc.)

- 600 - Smooth marble wall with trim
- 601 - Stone block wall
- 602 - Tan dirt wall
- 603 - Rough stone wall
- 604 - Blue vahnatai wall
- 605 - Wood wall

**Partial wall set** (Only basic walls, open doors, and closed doors)

- 606 - Smooth gray stone
- 607 - Rough black and white stone

**Walls only** (Just the basic walls, no doors, windows, etc.)

- 608 - Stucco with wooden support post
- 609 - Web
- 610 - Hedge wall
- 611 - Dark dirt
- 612 - Dark dirt with wooden support post
- 613 - Light marble
- 614 - Blue-green cave wall
- 615 - Dark, evil wall (black with electric blue trim)
- 616 - Surface outdoor cliff
- 617 - Trash wall
- 618 - Machinery/electronics

Similarly, you can specify a graphics sheet for cliffs/pit sides. The preset options are:

### **Cliff Types**

- 650 - Dirt
- 651 - Smooth gray stone (match wall 606)
- 652 - Rough black and white stone (match wall 607)
- 653 - Dark dirt (match wall 611)
- 654 - Light marble (match wall 613)
- 655 - Smooth marble (match wall 600)
- 656 - Dark, evil wall (match wall 615)
- 657 - Surface outdoor cliff (match wall 616)
- 658 - Blue-green cave wall (match wall 614)
- 659 - Vahnatai wall (match wall 604)
- 660 - Wood wall (match wall 605)
- 661 - Tan clay wall (match wall 602)
- 662 - Rough stone wall (match wall 603)

## Character Skills

These are the numbers of the skills characters in Blades of Avernum can have.

- 0 - Strength
- 1 - Dexterity
- 2 - Intelligence
- 3 - Endurance
- 4 - Melee Weapons
- 5 - Pole Weapons
- 6 - Bows
- 7 - Thrown Missiles
- 8 - Hardiness
- 9 - Defense
- 10 - Assassination
- 11 - Mage Spells
- 12 - Priest Spells
- 13 - Arcane Lore
- 14 - Potion Making
- 15 - Tool Use
- 16 - Nature Lore
- 17 - First Aid
- 18 - Luck
- 19 - Quick Strike
- 20 - Parry
- 21 - Blademaster
- 22 - Anatomy
- 23 - Gymnastics
- 24 - Pathfinder
- 25 - Magery
- 26 - Resistance
- 27 - Magical Efficiency
- 28 - Lethal Blow
- 29 - Riposte
- 30 - Sharpshooter
- 31 - Dread Curse

Dread Curse is a special skill. Each level of Dread Curse reduces each other skill by 1 (so if a character has Strength of 5 and Dread Curse of 2, the character's Strength is treated as 3 by the game). All levels of Dread Curse are removed when the party leaves a scenario.

In addition, there are several secondary skills derived from the regular skills.

- 35 - Maximum Health
- 36 - Maximum Spell Energy
- 37 - Poison Resistance
- 38 - Magic Resistance
- 39 - Willpower
- 40 - Resist Elements
- 41 - Item Lore
- 42 - Rune Reading (This is the skill you should check when determining if the party can read magical writing.)

## **Mage Spells**

These are the 20 Mage spells in Blades of Avernum, with numbers.

- 0 - Bolt of Fire
- 1 - Light
- 2 - Call Beast
- 3 - Spray Acid
- 4 - Haste
- 5 - Slow
- 6 - Ice Lances
- 7 - Unlock Doors
- 8 - Create Illusions
- 9 - Far Sight
- 10 - Lightning Spray
- 11 - Capture Mind
- 12 - Simulacrum
- 13 - Dispel Barrier
- 14 - Summon Aid
- 15 - Forcecage
- 16 - Fireblast
- 17 - Arcane Summon
- 18 - Arcane Shield
- 19 - Arcane Blow

## Priest Spells

These are the 20 Priest spells in Blades of Avernium, with numbers.

- 0 - Healing
- 1 - Curing
- 2 - War Blessing
- 3 - Terror
- 4 - Repel Spirit
- 5 - Smite
- 6 - Summon Shade
- 7 - Enduring Barrier
- 8 - Unshackle Mind
- 9 - Move Mountains
- 10 - Mass Healing
- 11 - Mass Curing
- 12 - Radiant Shield
- 13 - Divine Fire
- 14 - Control Foes
- 15 - Cloud of Blades
- 16 - Return Life
- 17 - Divine Retribution
- 18 - Divine Restoration
- 19 - Divine Host

## **Alchemy Recipes**

These are the 17 potion recipes in Blades of Avernum, with numbers.

- 0 - Healing Potion
- 1 - Curing Potion
- 2 - Hasting Potion
- 3 - Energy Potion
- 4 - Strength Potion
- 5 - Graymold Salve
- 6 - Balm of Life
- 7 - Healing Elixir
- 8 - Hasting Elixir
- 9 - Energy Elixir
- 10 - Rogue's Elixir
- 11 - Strength Elixir
- 12 - Bliss Elixir
- 13 - Restoration Brew
- 14 - Protection Brew
- 15 - Heroic Brew
- 16 - Knowledge Brew

## Character Traits

There are 15 possible traits a character can have. These are their numbers, in case you want to see if a character has one (using, say, the `char_has_trait` call).

- 0 - Strong Back
- 1 - Nimble Fingers
- 2 - Beastmaster
- 3 - Strong Will
- 4 - Good Education
- 5 - Toughness
- 6 - Fast on Feet
- 7 - Natural Mage
- 8 - Elite Warrior
- 9 - Divinely Touched
- 10 - Cursed At Birth
- 11 - Sickness Prone
- 12 - Sluggish
- 13 - Brittle Bones
- 14 - Completely Inept

In addition, characters have some innate abilities. Some, like Call Spirit, come from the traits above. Some, like Ritual of Sanctification, are learned in the scenarios. These innate abilities are below. When their value is 1, the character has the ability and hasn't used it today. When the value is 2, it hasn't been used that day.

- 15 - Summon Beast
- 16 - Call Spirit
- 17 - Regenerate
- 18 - Cure Afflictions
- 19 - Battle Frenzy
- 20 - Divine Aid
- 21 - Restore Energy
- 22 - Ritual of Sanctification

## **Status Types**

A status is any temporary condition a character can have. Examples are blessing, invulnerability, poison, and forcecage. Each status is represented by a number. Most of the time, each status will be at 0, which means the character doesn't have it (a non-poisoned character has a poison status of 0). Statuses are changed by spells, enemy attacks, special encounters, and similar circumstances.

Below is a list of all the possible statuses. Unless said otherwise, they can not be negative, and they go down by one per turn. The different statuses a character can have (and their numbers) are

The complete status list:

- 0 - Poison. Each turn, game divides the amount of poison the character has by four (rounded up), does 1-2 points of damage that many times, and reduces the poison level by that amount.
- 1 - Bless/Curse. If positive, character is blessed. If negative, cursed.
- 2 - Shield/Weaken. If positive, character is shielded. If negative, weakened.
- 3 - Haste/Slow. If positive, character is hasted. If negative, slowed.
- 4 - Invulnerable.
- 5 - Magic Resistant.
- 6 - Webbed.
- 7 - Diseased. Goes down slower than 1 per turn.
- 8 - Charmed.
- 9 - Berserk.
- 10 - Asleep.
- 11 - Paralysis.
- 12 - Acid. Each turn, game divides the amount of acid the character has by two (rounded up), does 1-2 points of damage that many times, and reduces the acid level by that amount.
- 13 - Dumbfounded.
- 14 - Sanctuary.
- 15 - Martyr's Shield.
- 16 - Divinely Touched
- 17 - Resistant
- 18 - Confused
- 19 - Enlightened. This unusual status is only used for special encounters. You can have one special encounter that makes the party enlightened, and another encounter that checks to see if they still have this status to, say, open a door or use some other effect.
- 20 - Terrified
- 21 - Enfeebled
- 22 - Regenerating
- 23 - Nimble Fingers
- 24 - Featherfall
- 25 - Flying
- 26 - Safe Travel
- 27 - Hovering Feet
- 28 - Drunk.
- 29 - Force Cage. Each turn, goes down by the character's strength, plus 1.
- 30 - Fleeing. Each turn, goes down by the character's intelligence, plus 1.

## **Status Ranges**

The legal range for most statuses is 0 to 250 (so it can have any value from 0, for no status, to 250, the maximum). There are some exceptions. Bless, Shield, and Haste can have a negative value, and can range from -250 to 250. Sleep, Charm, and Paralyze can be at most 10 (so a character can't magically sleep for over 10 turns).

## **Incidental Town Noises List**

For each town, you can select an incidental sound to play in the background. This is set in the Town Details window. The options are:

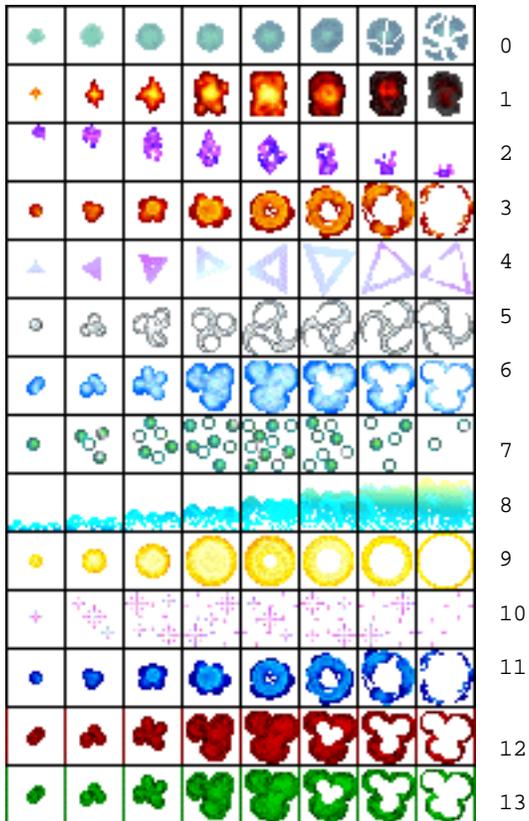
- 1 - no incidental sounds
- 0 - plays the best outdoors noise, depending on whether the town/dungeon is underground and whether it is day or night
- 144 - city
- 145 - village
- 146 - dungeon (bats, hisses, fewer drops, clangs)
- 147 - machinery
- 148 - outdoor, surface, day, first half
- 149 - outdoor, surface, night, first half
- 150 - spooky chanting/clanking
- 151 - outdoor, underground (water drops)
- 153 - outdoor, surface, day, second half
- 154 - outdoor, surface, night, second half
- 155 - winds (shorter)
- 156 - creepy woods (shorter)

## Missile Animations

|  |  |  |  |  |  |  |  |            |
|--|--|--|--|--|--|--|--|------------|
|  |  |  |  |  |  |  |  | Missile 0  |
|  |  |  |  |  |  |  |  | Missile 1  |
|  |  |  |  |  |  |  |  | Missile 2  |
|  |  |  |  |  |  |  |  | Missile 3  |
|  |  |  |  |  |  |  |  | Missile 4  |
|  |  |  |  |  |  |  |  | Missile 5  |
|  |  |  |  |  |  |  |  | Missile 6  |
|  |  |  |  |  |  |  |  | Missile 7  |
|  |  |  |  |  |  |  |  | Missile 8  |
|  |  |  |  |  |  |  |  | Missile 9  |
|  |  |  |  |  |  |  |  | Missile 10 |
|  |  |  |  |  |  |  |  | Missile 11 |

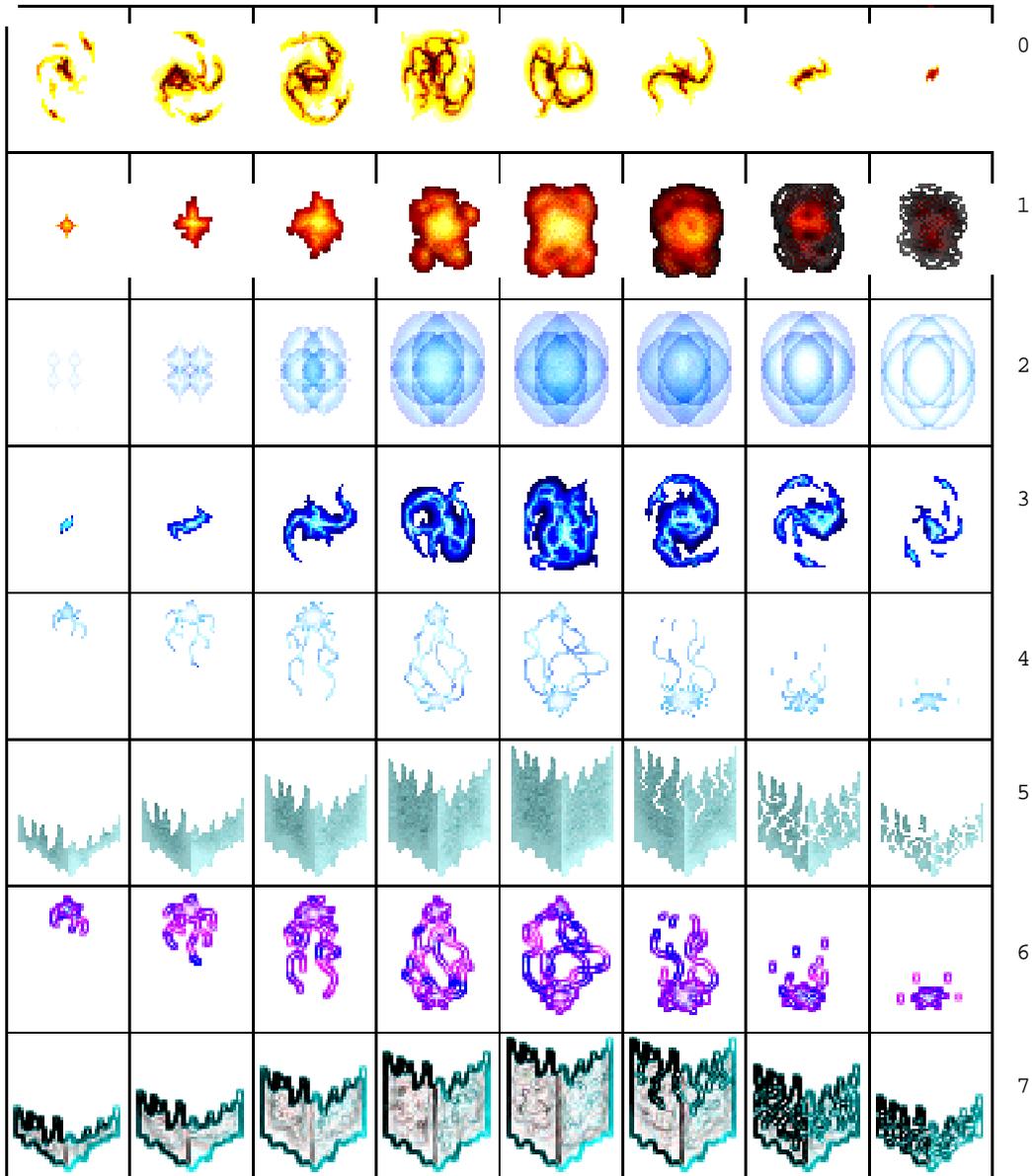
When defining a custom item that is a missile, you can use the `it_missile_anim_type` to define the appearance of the missile. The missile options are visible above.

## Effects



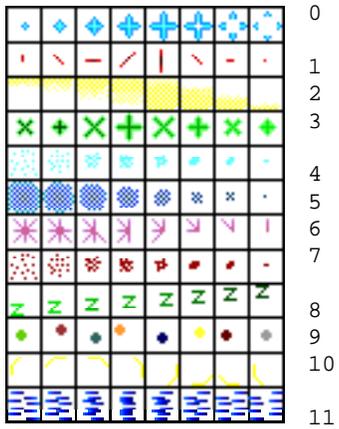
There are a variety of calls that create graphical effects on the screen. The calls `put_effect_on_char` and `put_effect_on_space` require you to select an effect. The different effects are listed above.

## Explosions



There are a variety of calls that create graphical effects on the screen. The calls `put_boom_on_char` and `put_boom_on_space` require you to select an explosion. The different explosions are listed above.

## Sparkles

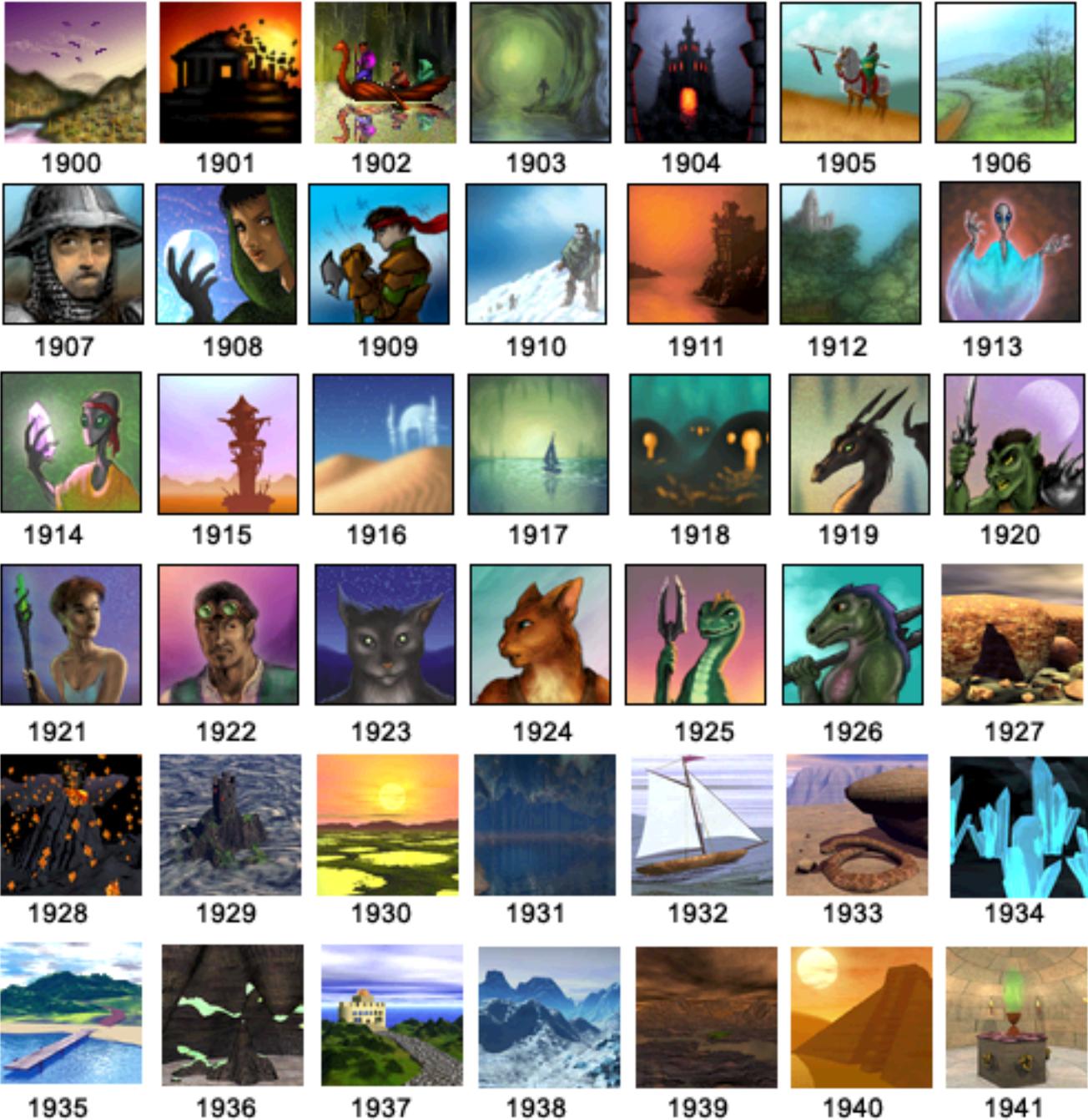


There are a variety of calls that create graphical effects on the screen. The calls `put_sparkles_on_char` and `put_sparkle_on_space` require you to select a small sparkly effect. The different sparkles are listed above.

## Scenario Icons

These are all of the 64 x 64 icons available for you to select as the symbol for your scenario and for the pictures that appear when talking to certain characters (set using the `set_char_dialogue_pic` call).

### Scenario and Dialogue Icons





1942



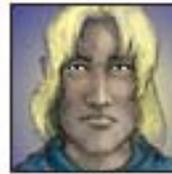
1943



1944



1945



1946



1947



1948



1949



1950



1951



1952



1953



1954



1955



1956



1957



1958



1959



1960



1961



1962



1963



1964



1965



1966



1967

## Sound Effects

Blades of Avernum has a long list of available sound effects, which are referred to by number. They are:

- 0 - high beep
- 1 - low beep
- 2 - weapon swish 1
- 3 - cough
- 4 - magic sound (blessing)
- 5 - small explosion
- 6 - chomping noise
- 7 - magic sound (slowing)
- 8 - making a potion
- 9 - unlock click
- 10 - teleportation sound
- 11 - flying fireball
- 12 - magic sound (haste)
- 13 - party dead
- 14 - thrown missile
- 15 - cash register ding
- 16 - enter town
- 17 - poison squish
- 18 - drawing sword
- 19 - weapon swish 2
- 20 - yawn
- 21 - female human dying 1
- 22 - starting music
- 23 - outdoor combat starts
- 24 - magic sound (generic priest)
- 25 - magic sound (generic mage)
- 26 - gain level chime
- 27 - humanoid dying
- 28 - magic sound (generic beneficial)
- 29 - male human dying
- 30 - female human dying 2
- 31 - generic death
- 32 - lizard dying
- 33 - beast dying
- 34 - click
- 35 - high-pitched "Hi"
- 36 - high-pitched "Hello"
- 37 - button click
- 38 - coin hitting counter
- 39 - jingling coins
- 40 - undead dying
- 41 - failure "Ungh"
- 42 - acid hiss
- 43 - magic sound (stoning)
- 44 - breath weapon
- 45 - stone sliding on stone
- 46 - magic sound (curse)
- 47 - footstep on gravel
- 48 - magic sound (going berzerk)
- 49 - footstep 1
- 50 - footstep 2
- 51 - magic sound (shooting flames)
- 52 - magic sound (ice)
- 53 - magic sound (magic bolt)
- 54 - magic sound (lightning)
- 55 - footstep (squish)
- 56 - swallowing sound
- 57 - special encounter chime
- 58 - door opening sound 1
- 59 - door closing
- 60 - magic sound (angelic)

61 - magic sound (summoning boing)  
62 - magic sound (long zap)  
63 - "Owwww!"  
64 - spit  
65 - magic sound (draining)  
66 - disease gagging  
67 - light torch flames  
68 - magic sound (identification peeyoop)  
69 - sword blow 1  
70 - sword blow 2  
71 - pole weapon 1  
72 - club/punch blow  
73 - fire blow  
74 - magic sound (fireball)  
75 - magic sound (cold damage)  
76 - clunk  
77 - equip/get armor  
78 - water drip  
79 - slide and clunk  
80 - equip/get weapon  
81 - tiny poink sound  
82 - open backpack  
83 - close backpack  
84 - male human dying 2  
85 - rowing  
86 - clawing sound  
87 - bite sound  
88 - slimy blow  
89 - magic sound (magic zap)  
90 - magic sound (laser zap)  
91 - arrow sound  
92 - humanoid dying  
93 - sword sheathed  
94 - level clunk  
95 - long enter dungeon sound  
96 - long yawn  
97 - generic monster death  
98 - missile blow  
99 - rattling chain  
100 - waterfall  
101 - dropped item thud  
102 - magic sound (big zap)  
103 - slith dying  
104 - magic sound (fast magic missile)  
105 - nephilim death  
106 - latch sound  
107 - box opened  
108 - crumbling rock  
109 - magic sound (small fire boom)  
110 - vahnatai door opening  
111 - vahnatai killed  
112 - magic sound (smite spell)  
113 - talking skull 1  
114 - talking skull 2  
115 - talking skull 3  
116 - horse moving  
117 - mounting a horse  
118 - large blow  
119 - demon dying  
120 - digging  
121 - bug dying  
122 - slime attack  
123 - slime dying  
124 - equip/get pole weapon  
125 - equip/get thrown weapon  
126 - equip/get bow  
127 - equip/get potion

128 - equip scroll  
129 - equip/get /get wand  
130 - equip/get tool  
131 - equip/get pants  
132 - equip/get shield  
133 - equip/get leather  
134 - equip/get chain  
135 - equip/get plate  
136 - equip/get helm  
137 - equip/get gloves  
138 - equip/get boots  
139 - equip/get cloak  
140 - equip/get ring  
141 - equip/get necklace  
142 - equip/get bracelet  
143 - equip/get arrow/bolt  
144 - ambience - city background noise  
145 - ambience - village background noise  
146 - ambience - dungeon background noise (bats, hisses, fewer drops, clangs)  
147 - ambience - machinery background noise  
148 - ambience - outdoor, day background noise, first half  
149 - ambience - outdoor night background noise, first half  
150 - ambience - spooky chanting/clanking background noise  
151 - ambience - outdoor, underground background noise (water drops)  
152 - big explosion  
153 - ambience - outdoor, day background noise, second half  
154 - ambience - outdoor night background noise, second half  
155 - ambience - winds  
156 - ambience - creepy woods  
157 - kick attack  
158 - pole weapon 2  
159 - door opening sound 2  
160 - door opening sound 3  
161 - meow  
162 - odd, magical chime  
163 - magical whoosh  
164 - lightning strike  
165 - magical conjuration  
166 - blessing sound  
167 - fiery breath weapon  
168 - bubbly magic sound  
169 - gassy, fiery boom  
170 - fast, magic zap  
171 - quick lightning zap  
172 - moo  
173 - dog bark

## **The Calls**

All of the multitude of calls available in Blades of Avernum are listed in this section. They are organized into subsections:

### **Basic Topics**

Basic Script and IO Calls  
Campaign and Scenario Calls  
Basic Character Calls  
Item, Special Item and Gold Calls  
Location and Distance Calls  
Statistic and Damage Functions  
Animation and SFX Functions  
Scenario Initialization Calls  
Boats and Horses  
Event Calls  
Town Calls  
Outdoor Calls  
The Passage of Time  
Miscellaneous Special Calls  
Terrain Checking and Modification  
Dialogue Calls  
Basic Dialog Box Calls

### **Calls For Terrain and Creature Scripts**

Terrain Script and Creature Script Calls  
Terrain Script Calls  
Movement, AI Functions  
Combat and Targeting Calls

### **Advanced Topics**

Advanced Dialog Box Calls  
Grouping Calls  
Messaging Calls  
Advanced Item Management Calls  
NPCs Joining Party Calls  
Splitting Up the Party Calls  
Custom Item Ability Calls  
Custom Character Ability Calls  
Cutscene Calls  
String Manipulation Calls  
Debugging Procedures

### **General notes about scripting:**

**which\_char** - Many calls expect to be given a value for which\_char. This expects the number of a character, which will be from 0 to 119.

(Note that you are not entering the character's Character ID or the personality. To find the number of a character, in the editor, select the character and look at the number in the first description line.)

If you use such a call in a creature script, if you enter a value of -1 for which\_char, the function will act on the creature whose script is running. In such a case, you can also enter ME for which\_char. This does the same thing.

**which\_char\_or\_group** - When a call expects a value for which\_char\_or\_group, it wants the number of a character or group. This is exactly like which\_char, except for one difference: the call can also act on every member of a group. If you want to have the call act on group x, give which\_char\_or\_group a value of 1000 + x. (And remember that group 0 is always the party.)

## The Most Important Calls:

You should pay special attention to these calls. They are the calls you will use the most frequently to write your special encounters.

This call is the way to begin a conversation.

```
void begin_talk_mode(short start_node)
```

This call lets you block containers and spaces from the party.

```
void block_entry(short do_blockage)
```

These two calls let you easily reward the party.

```
void change_coins(short coin_amount)
short reward_give(short which_item)
```

These two calls let you see if a character exists and instantly make a new one.

```
short char_ok(short which_char)
short spawn_creature(short which_char)
```

Every scenario must use this call. Otherwise, the party can never escape.

```
void end_scenario(short party_won)
```

The best call to display some simple text to the player is:

```
void message_dialog(char text1,char text2)
```

This call enables you to customize your outdoor encounters.

```
void outdoor_enc_result(short which_result)
```

These are the two primary calls for setting and checking Stuff Done flags.

```
void set_flag(short a,short b,short new_value)
short get_flag(short a,short b)
```

These two calls enable you to instantly customize creatures in your towns:

```
void set_name(short which_char_or_group,char new_name)
void set_level(short which_char_or_group,short what_level)
```

This call lets you jump around special scripts easily.

```
void set_state_continue(short new_state)
```

## Basic Topics

### Basic Script and IO Calls

**print\_big\_str(string str,short num\_to\_print,string str2)** - Like print\_str, but prints a more customized piece of text. Prints, on one line, the text str, followed by the number num\_to\_print, followed by the text str2.

Example: print\_big\_str("You are hit for ", 10000, " damage."); prints "You are hit for 10000 damage." to the text window.

**print\_big\_str\_color(string str,short num,string str2,short color)** - Like print\_big\_str, but gives the text a different color. Colors are listed in the description of print\_str\_color.

**void block\_entry(short do\_blockage)** - This very important call prevents the character from accessing the terrain spot/area that called the script. If called from a special encounter triggered by walking, this call prevents the party/character from entering the space. If called when searching a container, blocks the party from being able to get the item inside. If do\_blockage is 1, the call blocks entry. If it is 0, allows entry, and undoes any earlier uses of this call with do\_blockage equal to 1.

**void end()** - If called in a script, immediately stops running the script. If called in the code of a dialogue node, ends the conversation.

**short get\_ran(short num\_dice,short min,short max)** - Returns a random number. The game generates num\_dice random numbers, each is from min to max. It then adds them up and returns the sum.

Example: get\_ran(3,1,4) returns a random number from 3-12 (or 1-4 + 1-4 + 1-4).

**short get\_selected\_char()** - Returns the number of the character currently selected in the party roster area (which will be from 0 to 3).

**short is\_combat()** - Returns 1 if party is in combat mode, 0 otherwise.

**short is\_outdoor()** - Returns 1 if party is in outdoor mode, 0 otherwise. (Being in a combat outdoors is NOT outdoor mode.)

**short is\_town()** - Returns 1 if party is in town mode, 0 otherwise.

**void play\_sound(short which\_sound)** - Plays the sound which\_sound. Sounds are listed in the appendices. If the number of the sound is positive, it stops the action while the sound plays. If negative, the sound plays while the game continues.

**void print\_str(string to\_print)** - Displays the given text in the game's text area. The maximum allowable length of text is 70 characters.

**void print\_str\_color(string to\_print,short color)** - Like print\_str, but also gives the text a different color. Values for color are:

- 0 - normal black
- 1 - red (usually used for errors)
- 2 - blue
- 3 - magenta
- 4 - green

**void run\_scenario\_script(short which\_node)** - Immediately runs the scenario script at state which\_node. Once the scenario script stops running, the script that made this call continues running normally.

**void run\_town\_script(short which\_node)** - Immediately runs the current town script at state which\_node. Once the scenario script stops running, the script that made this call continues running normally. You can only make this call from dialogue, creature, or terrain scripts.

**void set\_incidental\_sound(short on\_or\_off)** - Sets whether incidental sounds (birds chirping, etc.) are played in the background. If on\_or\_off is 1, turns them off. If 0, turns them back on. If turned off, they stay off until this call turns them on again.

**void set\_state(short new\_state)** - Changes the state of the script to new\_state and immediately stops running the script. Has no effect in a dialogue script or in the states INIT\_STATE, DEAD\_STATE, or other predefined state types (except START\_STATE).

**void set\_state\_continue(short new\_state)** - Exactly like set\_state, except that it doesn't stop running the script.

### Campaign and Scenario Calls

These are the functions for manipulating the scenario variables and the Stuff Done flags. For information on Stuff Done flags and the meaning of SDF(a,b), read the chapter on scripting basics.

**void clear\_quest(short which\_quest)** - Resets the quest which\_quest (which can be from 0 to 99). It is as if it was never given. No text message is displayed.

**void end\_scenario(short party\_won)** - Immediately terminates the scenario and returns the party to the title screen. If party\_won is 1, the party's won scenarios value is incremented. If 0, not.

If party\_won is 2, the party is taken out of the scenario and the number of scenarios the party is recorded as having played is not incremented. (You probably usually won't want to use this.)

**short get\_flag(short a,short b)** - Identical to get\_sdf.

**short get\_sdf(short a,short b)** - Returns SDF(a,b).

**void inc\_flag(short a,short b,short how\_much)** - Changes SDF(a,b) by how\_much (which can be negative). Note the legal range of values for a flag is 0..255.

**void set\_flag(short a,short b,short new\_value)** - Sets SDF(a,b) to be new\_value.

**void toggle\_quest(short which\_quest,short start\_or\_finish)** - Toggles whether the party is doing quest number which\_quest (which can be from 0 to 99) or not. If start\_or\_finish is 1, and the party has not completed this quest, the quest is started and appears on the party's quest list. If start\_or\_finish is 0 and the party is currently on the quest, the quest is marked as finished and removed from the quest list.

### Basic Character Calls

**void alert\_char(short which\_char\_or\_group)** - Alerts the Creature/Group. This means that, if it is hostile, it will actively hunt the party down when they are nearby.

**void award\_char\_xp(short which\_char,short amount,short source\_level)** - Gives character which\_char amount experience, adjusted as if it was gained by killing a creature of level source\_level. You can't give more than 1000 experience at any one time.

**void award\_party\_xp(short amount,short source\_level)** - Gives each party member amount skill points, adjusted as if they were gained by killing a creature of level source\_level. Source\_level is relative level of party/monster giving experience. (1-10 for Beginner, 10-20 for Medium, 20-30 for High)

Note that a character can't get more than 1000 experience in one lump.

**short can\_see\_char(short which\_char\_or\_group)** - This can only be called in a creature or terrain script. Returns 1 if the creature/terrain spot whose script it is can see the given character or member of the given group, or 0 otherwise.

**void change\_char\_xp(short which\_char,short amt\_to\_change)** - Changes experience character which\_char by amount. Amt\_to\_change can be negative.

**void change\_pc\_skill\_pts(short which\_char,short amt\_to\_change)** - Changes the character which\_char skill points by amt\_to\_change. Amt\_to\_change can be negative.

**void change\_spell\_level(short which\_char,short mage\_or\_priest,short which\_spell,short amt\_to\_change)** - Changes the skill character which\_char knows spell which\_spell by

amt\_to\_change (which can be negative). If mage\_or\_priest is 0, changes mage spell. If 1, changes priest spell.

If the character is a party member, he or she can only learn the spell if his or her mage or priest spells skill is high enough. If the skill isn't high enough, this call has no effect.

**short char\_attitude\_to\_char(short which\_char,short char\_to\_evaluate)** - Evaluates how character which\_char feels about character char\_to\_evaluate. Returns 0 if the character likes the other character, 1 if the character is neutral about the other character, and 2 if the character hates the other character.

**short char\_has\_trait(short which\_char,short which\_trait)** - Returns 1 if character which\_char has trait which\_trait, 0 if it doesn't.

Traits are listed in the appendices.

**short char\_ok(short which\_char)** - Returns 1 if this character is existing and alive, 0 otherwise.

**short char\_on\_spot(short loc\_x,short loc\_y)** - Checks if there is an NPC (a character not controlled by the player, numbers 6 or higher) on space {loc\_x,loc\_y}. Returns -1 if no character there. Otherwise, returns the number of the character there.

If you also want to check for the presence of the player's characters, use the call char\_on\_loc.

**short char\_status(short which\_char)** - Returns the current status of this character:

- 0 - doesn't exist
- 1 - alive and ok
- 2 - dead
- 3 - dust
- 4 - stoned
- 5 - fled
- 11 - party split up, character alive but not present
- 12 - party split up, character dead but not present
- 13 - party split up, character dust but not present
- 14 - party split up, character stoned but not present

**short creature\_type(short which\_char)** - Returns the creature type of which\_char.

**void deduct\_ap(short num\_ap)** - This can only be called in a creature script and only has effect in combat. Removes num\_ap action points from the character whose script is running.

**void end\_combat\_turn()** - This can only be called in a creature script and only has effect in combat. Removes all action points from the character.

**short enemies\_nearby(short range)** - This can only be called in a creature's script. Returns the number of characters within range spaces that the character can see and which are hostile.

**void erase\_char(short which\_char\_or\_group)** - Makes the character/group stop existing. No treasure is dropped. The character's killed flag is not set (so it can reappear when the party returns).

**short friends\_nearby(short range)** - This can only be called in a creature's script. Returns the number of characters within range spaces that the character can see and which are friendly.

**short get\_attitude(short which\_char)** - Returns the attitude of the character.

- 3 - friendly to party
- 4 - neutral
- 10 - hostile A
- 11 - hostile B

**short get\_char\_skill\_pts(short which\_char)** - Returns the character's skill points.

**short get\_energy (short which\_char)** - Returns the character's current spell energy. *This call was added for Scenario Format Version 2, only use it with Mac version 1.1 or later or Windows version 1.0 or later.*

**short get\_health(short which\_char)** - Returns the character's current health.

**short get\_level(short which\_char)** - Returns the character's level.

**short get\_max\_health(short which\_char)** - Returns the character's maximum health.

**short get\_species(short which\_char)** - Returns the character's species.

**short get\_spell\_level(short which\_char,short mage\_or\_priest,short which\_spell)** - Returns the skill level that character which\_char knows spell which\_spell. If mage\_or\_priest is 0, checks mage spell. If 1, checks priest spell.

**short get\_summon\_level(short which\_char)** - Returns the summoning level of the current character (which can reveal whether it is summoned or not). The results are:

0 - Not summoned. Real creature.

1 - Summoned, but will not disappear over time.

100-199 - Summoned. Subtract 100 from amount to get number of turns until it disappears.

200-299 - Illusion. Subtract 200 from amount to get number of turns until it disappears.

**void give\_recipe(short which\_recipe)** - Teaches the party recipe which\_recipe. Recipes are listed in the appendices.

**short has\_recipe(short which\_recipe)** - Returns 1 if party knows alchemy recipe which\_recipe, 0 if it doesn't. Recipes are listed in the appendices.

**void kill\_char(short which\_char\_or\_group, short death\_type, short luck\_helps)** - Kills the character or every member of the group. Death\_type determines the sort of death:

2 - dead

3 - dust

4 - stoned

If luck\_helps is 1, the character's luck can save it. If 0, there is no escape.

**short my\_ap()** - This can only be called in a creature script and only has effect in combat. Returns the character's current number of action points.

**short my\_number()** - Can only be called in a creature script or terrain script. Returns this creature's number (0 - 119) if in a creature script or the terrain script's number (0 - 99) in a terrain script.

**short party\_can\_see\_loc(short loc\_x,short loc\_y)** - This call only works correctly inside a town. Returns 1 if a party member can see location {loc\_x,loc\_y}, 0 otherwise.

**short party\_has\_trait(short which\_trait)** - Returns 1 if some party member has trait which\_trait, 0 if it doesn't. Traits are listed in the appendices.

**short party\_size()** - Returns the number of living, present characters in the party.

**void print\_named\_str(short which\_char,char print\_text)** - Prints in the text area the name of character which\_char, immediately followed by print\_text.

Example: If character 17 is a Cute Puppy, the command  
print\_named\_str(17," fires ray of doom.");  
puts the text "Cute Puppy fires ray of doom." in the text area.

**short random\_party\_member()** - Returns the number of a random character in the party (will be a number from 0 to 5).

**void revive\_party()** - Completely heals the party's damage and restores all spell energy.

**void set\_attitude(short which\_char\_or\_group,short which\_attitude)** - Sets the attitude of the character/group:

3 - friendly to party

4 - neutral

10 - hostile A  
11 - hostile B

**void set\_char\_alert(short which\_char\_or\_group,short alert\_level)** - Sets the alert level of the Creature/Group which\_char\_or\_group to value alert\_level. If alert\_level is 1, the characters are alerted and will, if hostile to the party, hunt the party down. If 0, they will no longer be alerted.

**short set\_char\_script\_mode(short which\_char\_or\_group,short what\_setting)** - Adjusts when the scripts for the Creature/Group will be run by the game when the party is not nearby. (By default, the game does not call all the scripts all the time, to keep the game from slowing down.) Values for what\_setting are:

0 - call every 8 moves, but not if party is far away  
1 - call every 8 moves, even if party is far away  
2 - call every move, but not if party is far away  
3 - call every move, even if party is far away

The default is always 0.

**void set\_char\_trait(short which\_char,short which\_trait,short set\_to)** - For character which\_char, changes whether the character has trait which\_trait. If set\_to is 0, takes the trait away. If set\_to is 1, gives the trait. Traits are listed in the appendices.

Note that you should probably only use this ability to teach the character Ritual of Sanctification (trait 22). You really shouldn't mess with a character's other traits. This call only has a real effect on party members.

**void set\_creature\_memory\_cell(short which\_char\_or\_group,short which\_cell,short new\_value)** - For the character or each member of the group, sets the value of memory cell which\_cell to new\_value.

**void set\_level(short which\_char\_or\_group,short what\_level)** - Changes the level of the Creature/Group to what\_level. Automatically adjusts the character's statistics and health to reflect the new level. This call also completely heals and restores the character, so it should probably only be called when the party enters the area (i.e. in the town script's state INIT\_STATE).

You shouldn't use this call on party members.

**void set\_mobility(short which\_char\_or\_group,short mobile)** - Sets whether the Creature/Group can move or not. 1 is yes, 0 is no.

**void set\_name(short which\_char\_or\_group,char new\_name)** - Sets the name of the character/group to new\_name. Maximum name length is 19 characters.

**void set\_special\_ability(short which\_char\_or\_group,short what\_ability)** - Changes the special ability of the Creature/Group to what\_ability. The different special abilities are listed in the section on scripting creature types. If what\_ability is 0, the creature gets no special ability.

**void set\_summon\_level(short which\_char\_or\_group,short new\_level)** - Sets the summoning level of Creature/Group to new\_level. Summoned creatures and illusions do not give experience or treasure when killed. The legal values are:

0 - Not summoned. Real creature.  
1 - Summoned, but will not disappear over time.  
100-199 - Summoned. Subtract 100 from amount to get number of turns until it disappears.  
200-299 - Illusion. Subtract 200 from amount to get number of turns until it disappears.

**short species\_in\_party(short which\_species)** - which\_species is the number of a species (most often, 2 for nephil or 3 for slith). This call returns 1 if one of the characters in the party is alive and that species, 0 otherwise.

This call does check any extra characters that are traveling with the party.

### Item, Special Item and Gold Calls

**void change\_coins(short coin\_amount)** - Changes the number of coins the party has by coin\_amount (which can be negative).

If the number is negative, then coins are taken away. If the party doesn't have enough coins, no coins are taken.

**void change\_spec\_item(short which\_item,short how\_many)** - Changes the amount of special item which\_item the party has by how\_many (which can be negative if you want to take some of an item away). If you try to take away more of the item than the party has, the party is left with 0 of the item.

**void char\_give\_item(short which\_char\_or\_group,short which\_item)** - Tries to give to the Creature/Group an item of type which\_item. If you call char\_give\_item for a group, gives one of the item to each group member.

The item will not automatically be identified. If the character doesn't have enough space to take the item, nothing happens and the game gives a "can't take the item" message. *This call was added for Scenario Format Version 2, only use it with Mac version 1.1 or later or Windows version 1.0 or later.*

**short char\_has\_item(short which\_char,short which\_item)** - Returns 1 if character which\_char has an item of type which\_item, 0 otherwise.

**short char\_has\_item\_of\_class Equip(char char\_num,short which\_class,short take\_item)** - Returns 1 if a party member has an item with special class which\_class and the item is equipped, 0 otherwise. If take\_item is 1, one of the items is taken. If 0, it isn't. *This call was added for Scenario Format Version 2, only use it with Mac version 1.1 or later or Windows version 1.0 or later.*

**void char\_take\_item(short which\_char\_or\_group,short which\_item)** - Takes from the Creature/Group one item of type which\_item (or, if it has charges, 1 charge). If a group is given, takes one of the item from each member of the group.

**short coins\_amount()** - Returns the number of coins the party has.

**short has\_item(short which\_item)** - Returns 1 if a party member has an item of type which\_item, 0 otherwise.

**short has\_item\_of\_class(short which\_class,short take\_item)** - Returns 1 if a party member has an item with special class which\_class (this is set in the scenario item script), 0 otherwise. If take\_item is 1, one of the items is taken. If 0, it isn't.

**short has\_num\_of\_item(short which\_item)** - Returns how many items which\_item the party has, 0 otherwise.

If the items in question have charges, this call returns the total number of charges.

**short has\_special\_item(short which\_item)** - Returns the number of the special item which\_item the party has, 0 if they don't have any.

**short item\_of\_class\_on\_spot(short loc\_x,short loc\_y,short what\_class)** - Returns 1 if there is an item of special class what\_class on town space {loc\_x,loc\_y}, 0 otherwise. If what\_class is -1, returns 1 if any item is on the space, 0 otherwise.

**void move\_item\_on\_spot(short loc\_x,short loc\_y,short dest\_x,short dest\_y)** - Takes all items sitting on town spot {loc\_x,loc\_y} and moves them to {dest\_x,dest\_y}. If dest\_x is a negative value, the items on the space are simply destroyed.

**short pay\_coins(short coin\_amount)** - Tries to take coin\_amount coins from the party. Returns 0 if party doesn't have enough, 1 if they do.

**void put\_item\_on\_spot(short loc\_x,short loc\_y,short what\_item)** - Places an item of type what\_item on town space {loc\_x,loc\_y}. If that spot is a container, the item won't be contained inside it.

**short reward\_give(short which\_item)** - Tries to give item of number which\_item to party. If the party can't hold it, tries to put it at their feet (which only works indoors). If this fails, returns 0. Otherwise, returns 1.

**short take\_all\_of\_item(short which\_item)** - Takes all of item which\_item from party, and returns the number of items taken. If the item given has charges, returns the total number of charges taken.

**short take\_all\_of\_item\_class(short which\_class,short gold\_to\_pay)** - Takes from the party all items of special class `which_class` and, for each item taken, gives the party `gold_to_pay` gold. Returns 1 if the party had some items of that type, 0 otherwise. If the item taken has charges, gives the amount of gold for each charge taken.

**void take\_item(short which\_item)** - Searches through the party's inventory until it finds an item of type `which_item` and takes it (or, if it has charges, takes one of them).

**short take\_item\_of\_class\_on\_spot(short loc\_x,short loc\_y,short what\_class)** - Returns 1 if there is an item of special class `what_class` on town space `{loc_x,loc_y}`, 0 otherwise. If there was an item of that type on the spot, destroys one of it (or, if item has charges, takes 1 charge). If `what_class` is -1, destroys all items on the spot and returns TRUE if an item destroyed.

**void take\_num\_of\_item(short which\_item,short num\_to\_take)** - Takes `num_to_take` items of type `which_type` from the party. If the item has charges, takes that many charges. If the party has less than `num_to_take` items of that type, all are taken.

**void take\_special\_item(short which\_item)** - Takes 1 of special item `which_item` from party. If party has none of that special item, does nothing.

### Location and Distance Calls

Note that none of these functions is meant to be used in outdoors mode. They will not return proper values.

Also, if you are writing a creature or terrain script, be sure to look up the calls `my_loc_x` and `my_loc_y`.

**short char\_loc\_x(short which\_char)** - Returns the x coordinate of the character `which_char`.

**short char\_loc\_y(short which\_char)** - Returns the y coordinate of the character `which_char`.

**short char\_dist\_to\_loc(short which\_char,short x, short y)** - Returns the distance of character `which_char` from location `{x,y}`. Note that `which_char` can't be a group.

**short char\_on\_loc(short x, short y)** - Returns the number of the character on space `{x,y}`. If there is no character there, returns -1. Unlike `char_on_spot`, this call also checks if one of the player's characters is there.

**short dist\_to\_waypoint(short which\_char,short which\_point)** - Returns the distance of character `which_char` from waypoint `which_point`. Note that `which_char` can't be a group.

**short group\_dist\_to\_loc(short which\_group,short x, short y)** - Returns the distance of closest character in group `which_group` to the location `{x,y}`. Remember that group 0 is the party (in case you want to see who in the party is closest to a point). If the group is empty, the distance returned is 10000.

**short my\_dist\_from\_start()** - This call can only be made from a creature script. Returns the character's distance, in spaces, from where it started.

### Statistic and Damage Functions

**void alter\_stat(short which\_char\_or\_group,short which\_stat,short how\_much)** - For character `which_char`, changes statistic `which_stat` by `how_much`. You can't increase a statistic above 100.

`How_much` can be negative. Skills 0-3 can't be taken below 1 by this call, and no skill can be taken below 0.

**void change\_char\_energy(short which\_char\_or\_group,short energy\_amount)** - Changes spell energy for the Creature/Group by `energy_amount`. This can't raise a character over its normal energy maximum.

**void change\_char\_health(short which\_char\_or\_group,short heal\_amount)** - Changes the health of the Creature/Group by heal\_amount. This can't raise a character over its normal health maximum.

**short char\_with\_highest\_skill(short which\_skill)** - Returns the number of the party member with the highest level of the skill which\_skill. Returns -1 if no character has any of the skill.

**void damage\_char(short which\_char\_or\_group,short damage\_amount,short damage\_type)** - Does damage\_amount damage of type damage\_type to the Creature/Group.  
Damage types - 0 - weapon, 1 - fire, 2 - poison, 3 - general magic, 4 - unblockable, 5 - cold, 6 - acid

**void damage\_near\_loc(short loc\_x,short loc\_y,short damage\_amount,short radius,short damage\_type)** - Does damage\_amount damage of type damage\_type to all characters visible to and within radius spaces of location {loc\_x,loc\_y}. This does not work outdoors.  
Damage types - 0 - weapon, 1 - fire, 2 - poison, 3 - general magic, 4 - unblockable, 5 - cold, 6 - acid

**short get\_char\_status(short which\_char,short which\_status)** - Returns the current value of status which\_status for character which\_char.

**short get\_highest\_skill(short which\_skill)** - Returns value of the skill which\_skill for the party member who has the most for that skill.

**short get\_skill\_total(short which\_skill)** - Returns the sum of all the party members' values of skill which\_skill.

**short get\_stat(short which\_char,short which\_stat)** - Returns the current value of statistic which\_stat for character which\_char. This takes into account all bonuses for items, etc.

**short get\_stat\_levels\_bought(short char\_num,short which\_stat)** - Returns the number of levels of statistic which\_stat for character which\_char that the character has actually bought at a trainer. This will be less than the current value of the statistic. For non-player characters, this will always end up being 0.

**void heal\_char(short which\_char\_or\_group,short heal\_amount)** - Heals heal\_amount damage for the Creature/Group.

**short party\_has\_status(short which\_status)** - Returns 1 if one party member has a positive value for status which\_status, 0 otherwise.  
This is most useful when checking for Featherfall (status 24) or Flying (status 25);

**short party\_member\_has\_skill(short which\_skill)** - Returns 1 if a party member has at least 1 level of skill which\_skill, 0 otherwise.

**void restore\_energy\_char(short which\_char\_or\_group,short energy\_amount)** - Restores energy\_amount spell energy for the Creature/Group.

**void restore\_pc(short which\_char)** - Completely heals the character of all bad effects (including death or dust). If called in town, places the character next to the party. Removes the curse from all equipped cursed items.

**void set\_char\_status(short which\_char\_or\_group,short which\_status,short how\_much,short is\_forced,short give\_update)** - Changes status which\_status for character/group which\_char\_or\_group by amount how\_much. If is\_forced is 1, the status is always changed by the full amount. If is\_forced is 0 then, for negative statuses, the character's resistances apply.

If is\_forced is 0 and give\_update is 0, no text message about status change is given. Otherwise, it is. If the status change is forced, however, no text update is given and give\_update is ignored.

**void set\_party\_status(short which\_status,short how\_much,short give\_update)** - Changes status which\_status for every character in the party by amount how\_much. For negative statuses, character resistances apply (to force a change, use the set\_char\_status call). If give\_update is 0, no text message about status change is given. Otherwise, it is.

**void status\_near\_loc(short loc\_x,short loc\_y,short status\_amount,short radius,short status\_type)** - Change status status\_type by amount status\_amount to all characters visible to and within radius spaces of location {loc\_x,loc\_y}. This does not work outdoors. All appropriate resistances apply ... if you try to change the amount of poison for the nearby characters, their poison resistance will help them.

**void waypoint\_damage\_party(short which\_waypoint,short in\_range,short out\_range,short damage\_amount,short damage\_type)** - Does damage\_amount damage of damage\_type type to all characters near a visible waypoint which\_waypoint. Only affects creatures which are at least in\_range spaces and at most out\_range spaces from the waypoint.  
Damage types - 0 - weapon, 1 - fire, 2 - poison, 3 - general magic, 4 - unblockable, 5 - cold, 6 - acid

### Animation and SFX Functions

There are a variety of special effects that can be put on characters and terrain spots. Once the effects are placed, to actually display the animation, call run\_animation or run\_animation\_sound. All of the available effects, explosions, and sparkles are listed in the appendices.

None of these calls have any effect outdoors.

**void put\_boom\_on\_char(short which\_char,short which\_sfx,short random\_shift)** - Places explosion which\_sfx on character which\_char. If random\_shift is 0, effect appears directly on character. If 1, the explosion is shifted slightly in a random direction.

**void put\_boom\_on\_space(short loc\_x,short loc\_y,short which\_sfx,short random\_shift)** - Exactly like put\_boom\_on\_char, but makes the effect appear in terrain spot {loc\_x,loc\_y}.

**void put\_effect\_on\_char(short which\_char,short which\_sfx,short num\_effects,short float\_mode)**  
- Places effect which\_sfx on character which\_char. num\_effects is the number of effect graphics that appear, and float\_mode determines how they behave:  
0 - Appear above character and fall.  
1 - Appear below character and rise.  
2 - Hang in midair.

**void put\_effect\_on\_space(short loc\_x,short loc\_y,short which\_sfx,short num\_effects,short float\_mode)** - Exactly like put\_effect\_on\_char, but makes the effect appear in terrain spot {loc\_x,loc\_y}.

**void put\_jagged\_zap(short source\_x,short source\_y,short dest\_x,short dest\_y,short color\_scheme)** - Exactly like put\_straight\_zap, but the bolt is jagged. Values for color\_scheme are in listed in put\_straight\_zap.

**void put\_sparkles\_on\_char(short which\_char,short which\_sfx,short num\_sparkles)** - Places sparkle which\_sfx on character which\_char. num\_sparkles is the number of effect graphics that appear.

**void put\_sparkles\_on\_space(short loc\_x,short loc\_y,short which\_sfx,short num\_sparkles)** - Exactly like put\_effect\_on\_char, but makes the effect appear in terrain spot {loc\_x,loc\_y}.

**void put\_straight\_zap(short source\_x,short source\_y,short dest\_x,short dest\_y,short color\_scheme)** - Places a straight bolt of lightning running from {source\_x,source\_y} to {dest\_x,dest\_y}. Values for color\_scheme are  
0 - fire ray  
1 - ice ray  
2 - death ray  
3 - green ray  
4 - white ray  
5 - black ray  
6 - blue ray  
the bolt does not appear until one of the run\_animation calls is used.

**void run\_animation()** - Pauses the game and plays all animations you have created. Note that none of the above calls will draw anything on its own. You need to use this call to run any animations.

**void run\_animation\_sound(short which\_sound)** - Like run animation, but also plays sound which\_sound. Sounds are listed in the appendices. The sound plays while the animation runs.

### Scenario Initialization Calls

These calls are all meant to be made in the scenario script, either in the state LOAD\_SCEN\_STATE or START\_SCEN\_STATE. They can, however, be made anywhere.

**void add\_item\_to\_shop(short which\_shop,short which\_item,short how\_many)** - This call should be made in the state START\_SCEN\_STATE. This places 1 or more of an item in store which\_shop (which can be from 0 to 129). Normally, the item placed is which\_item (which is numbered 0 - 499, and how\_many copies of the item are put in the store. If there are more than 500 of an item put in the store, the store has an infinite number.

Any given shop can only have 25 items in stock.

However, if you pick certain high values of which\_item, you can put into the store things besides items (like spells).

Not only objects can be in stores. You can also have spells, alchemy recipes, and skills. The meanings of the values of which\_item are:

0 - 499 - item

2000 + x - mage spell x (x is from 0 to 19). how\_many is the highest level of the spell that can be bought in the store.

3000 + x - priest spell x (x is from 0 to 19). how\_many is the highest level of the spell that can be bought in the store.

4000 + x - alchemy recipe x (x is from 0 to 19). how\_many should be 1.

5000 + x - skill x. Each purchase gives the character 1 level of the skill. how\_many is the maximum number of levels of the skill that can be bought.

Note that, in all cases, you will want to put a number above 0 for how\_many. If you don't, the item will disappear from the store.

**void create\_boat(short which\_boat,short which\_town,short loc\_x,short loc\_y,short others\_property)** - This call should be made in the state START\_SCEN\_STATE. Initializes boat which\_boat (which can be from 0 to 29). Places the boat in town which\_town in space {loc\_x,loc\_y}. If others\_property is 0, the party can enter the boat. If 1, the boat must be set as party property by a script before the party can use it.

**void create\_horse(short which\_horse,short which\_town,short loc\_x,short loc\_y,short others\_property)** - This call should be made in the state START\_SCEN\_STATE. Initializes horse which\_horse (which can be from 0 to 29). Places the horse in town which\_town in space {loc\_x,loc\_y}. If others\_property is 0, the party can mount the horse. If 1, the horse must be set as party property by a script before the party can use it.

**void init\_quest(short which\_quest,string quest\_name,string quest\_desc)** - This call should be made in the state LOAD\_SCEN\_STATE. It sets the name of quest which\_quest to quest\_name and the description text to quest\_desc. The name can be at most 29 characters long and the description can be at most 199 characters long. The quests are numbered from 0 to 99.

**void init\_special\_item(short which\_item,string item\_name,string item\_desc)** - This call should be made in the state LOAD\_SCEN\_STATE. It sets the name of special item which\_item to item\_name and the description text to item\_desc. The name can be at most 29 characters long and the description can be at most 199 characters long. The special items are numbered from 0 to 59.

**void set\_creature\_type\_level(short which\_type,short what\_level)** - Sets the level of creature type which\_type to level what\_level. This is the only call that changes creature types (normally these changes are made in a different script). It is provided because, to balance a scenario, you will probably want to adjust the levels of some creature types.

### Boats and Horses

The values which\_horse and which\_boat are always from 0 to 29.

**short in\_boat()** - Returns -1 if party is not in a boat. Otherwise, returns the number of the boat they're in (from 0 to 29).

**short in\_horse()** - Returns -1 if party is not on a horse. Otherwise, returns the number of the horse they're on (from 0 to 29).

**void set\_boat\_property(short which\_boat,short party\_property)** - Changes property status of boat which\_boat

**short set\_boat\_range\_property(short first\_boat,short last\_boat)**- Finds one boat in the range from first\_boat to last\_boat that isn't the party's property and make's it the party's property. Returns the number of the boat that was made the party's property, or -1 if there was no existing boat in the range that the party didn't own.

**void set\_horse\_property(short which\_horse,short party\_property)** - Changes property status of horse which\_horse. If party\_property is 0, party can't use the horses. If party\_property is 1, they can.

**short set\_horse\_range\_property(short first\_horse,short last\_horse)** - Sets one horse in range from first\_horse to last\_horse that isn't the party's property to be the party's property. Returns the number of the horse that was made the party's property, or -1 if there was no existing horse in the range that the party didn't own.

### Event Calls

**short day\_event\_happened(short which\_event)** - Returns -1 if the event which\_event (which is from 0 to 9) has not happened, or the day it happened if it has happened.

**void set\_event\_happened(short which\_event,short what\_day)** - Records that event which\_event (a number from 0 to 9) happened on day what\_day (you'll probably pass what\_day\_of\_scenario()) to this.

### Town Calls

These calls have to do with towns and their status (whether they have been visited, are visible, etc.). Any calls that refer to the current town will do nothing if the party is not in town mode. To learn more about visibility, crime levels, etc., read the chapters on editing and scripting towns.

For calls that expect a value for which\_town, if you pass the value ME (or -1), the town the party is currently in is used.

**void activate\_hidden\_group(short which\_group)** - When town is entered, characters with a hidden group value do not appear. This call makes all characters with the hidden group value of which\_group appear. If the spot where the creature would appear is blocked, the game finds a clear nearby spot.

**void change\_crime\_level(short how\_much)** - Changes the level of crimes committed in the current town by how\_much (which can be negative).

**void change\_outdoor\_location(short what\_section\_x,short what\_section\_y,short loc\_in\_sector\_x, short loc\_in\_sector\_y)** - Changes the party's location in the outdoors. It moves them to location {loc\_in\_sector\_x,loc\_in\_sector\_y} in section X = what\_section\_x, Y = what\_section\_x. It can only be used in a town script. It is used, for example, when the party enters a portal in a town that is meant to carry them to a town a long distance away.

**void clear\_town()** - Sets the current town to be destroyed. When next entered, the only creatures that will be present are ones set to only be there when town destroyed.

**short current\_town()** - Returns the town the party is currently in. This call should not be used in outdoor mode.

**short current\_town\_size()** - Returns the size of the town the party is in (0 - 64x64, 1 - 48x48, 2 - 32x32).

**void enable\_add\_chars(short can\_add)** - Sets whether the player can add new characters to his or her party in this town. Defaults behavior is that the player is not able to add characters when a new town is entered. This call needs to be used to enable adding characters every time the town is entered (probably in the state INIT\_STATE).

If can\_add is 1, the player can add new characters. If 0, player can no longer add new characters.

You should probably enable the player to add characters in at least the starting town.

**short get\_crime\_level()** - Returns the level of crimes that have been committed by the party in the current town.

**void make\_town\_hostile()** - Sets the town permanently to hostile (this can be undone with the call set\_town\_status). Sets all friendly characters to hostile.

**void move\_to\_new\_town (short new\_town,short loc\_x,short loc\_y)** - Immediately takes the party out of combat mode and moves them to town new\_town, starting at {loc\_x,loc\_y}.

This call only works when called indoors. It will only function if called in a special encounter triggered by walking on a space. It won't work in creature, terrain, or dialogue scripts.

**short num\_killed\_in\_town(short which\_town)** - Returns the number of hostile creatures that have been killed in town which\_town.

**void place\_monster(short loc\_x,short loc\_y,short monst\_type,short not\_worth\_xp)** - Makes a monster of type monst\_type appear at {loc\_x,loc\_y}. It is hostile to the party. The game has 34 slots reserved for wandering monsters and creatures made using the place\_monster function.

If not\_worth\_xp is 0, than the creature is a normal creature. If not\_worth\_xp is 1, the creature drops no items and awards no experience when killed. If not\_worth\_xp is 2, the creature is considered to be summoned. It drops no items, isn't worth xp, and disappears a few turns later.

The summoned creature uses its default script.

**void set\_crime\_tolerance(short tolerance)** - Sets the crime level the town will endure before it goes hostile. This call needs to be made each time the party enters the town.

**void set\_items\_not\_property()** - Sets all items in the current town to not be someone else's property (so the party can now get them without penalty).

**void set\_town\_status(short which\_town,short new\_status)** - Manually sets town which\_town to status new\_status. For values of new\_status, see description of function town\_status.

**void set\_town\_visibility(short which\_town,short town\_visible)** - Sets whether town which\_town is visible and can be entered when party is outdoors. If town\_visible is 1, party can see. Otherwise, is not visible. When the scenario is started, all towns default to being visible.

**short spawn\_creature(short which\_char)** - Makes one of the town's creatures appear. which\_char is the number the creature would have in the game that is not a party member (so it is between 6 and 119). If that creature is not currently alive, it becomes alive, at that creature's spawning point. Returns 1 if a creature appears, 0 otherwise.

Example: The first creature you place in a town would be creature 6 in the game. If you use the call spawn\_creature(6), this creature would appear in the game, unless it is already alive.

**short teleport\_party(short new\_loc\_x,short new\_loc\_y,short fade\_type)**

mode -

- 0 full teleport flash
- 1 no teleport flash
- 2 only fade flash

**short town\_status(short which\_town)** Returns the status of which\_town. Status values are

- 0 - never been entered
- 1 - party has entered
- 2 - party has entered, town hates
- 3 - town has been destroyed

**void turn\_off\_training(short turn\_off)** - Every town defaults to let the party train in it. If you don't want the party to be able to train in the current town, use this call. If turn\_off is 1, the party can't train in this town. If it is 0, they can.

## Outdoor Calls

These calls are all only useful when outdoors. For more information (and to learn how outdoor sections are numbered) )read the chapter on the outdoors.

**void create\_out\_spec\_enc(short which\_enc)** - Creates a copy of the special encounter which\_enc (which is from 0 to 3) next to the party.

**short current\_out\_section()** - Returns the number of the outdoor section the party is currently in.

**void eliminate\_outdoor\_enc(short stuff\_done\_1, short stuff\_done\_2)** - Used to destroy groups of creatures outdoors. Each outdoor encounter can have a Stuff Done flag set which eliminates the encounter (makes it not appear). This call is given {stuff\_done\_1, stuff\_done\_2}, which are the two coordinates of a Stuff Done flag. This call eliminates all outdoor encounters whose elimination Stuff Done flag is {stuff\_done\_1, stuff\_done\_2}.

**void out\_move\_party(short loc\_x,short loc\_y)** - Relocates the party to space {loc\_x,loc\_y} in the current outdoor section.

**short outdoor\_enc\_exists(short stuff\_done\_1, short stuff\_done\_2)** - Used to check if a particular group of creatures wandering outdoors exists. Each outdoor encounter can have a Stuff Done flag set which eliminates the encounter (makes it not appear). This call is given stuff\_done\_1, stuff\_done\_2, which are the two coordinates of a Stuff Done flag. This call returns 1 if there is currently an existing group of outdoor creatures with a matching Stuff Done flag which eliminates it.

**void outdoor\_enc\_result(short which\_result)** - This call only has an effect when it is called as a result of the party encountering a group of creatures outdoors. It determines whether the party will have to fight the encounter or not. The meanings of which\_result are as follows:

- 0 - fight as normal
- 1 - don't have to fight
- 2 - don't have to fight, encounter disappears, but can reappear later
- 3 - don't have to fight, encounter disappears, sets the encounter's defeat Stuff Done flag (if it has one) to 1 (so it disappears forever).

**void place\_out\_spec\_enc(short which\_enc,short where\_x,short where\_y)** - Creates a copy of the special encounter which\_enc (which is from 0 to 3) at the location {where\_x,where\_y} in the current outdoor section.

**void set\_out\_fight\_town\_loaded(short which\_town)** - This call only has an effect when it is called as a result of the party encountering a group of creatures outdoors. It sets the town terrain that is loaded for the fight to take place in. which\_town is the town that is loaded when party fights an outdoor combat on this terrain type. If which\_town 0-999, loads the town in the current scenario. If 1000 + x, loads town x in the file Blades of Avernum Out Fight

### The Passage of Time

Time in Blades of Avernum is measured in "ticks". At the beginning of a scenario, the number of ticks that have elapsed is 0. Every turn that passes in town or every round of combat that ends increases the number of ticks by 1. Every turn that passes outdoors increases the number of ticks by 10 (5 if on a horse).

There are 5000 ticks in a day. The first eighth and last eighth of every 5000 tick period are nighttime.

There are 280 days in each year (numbered 0 to 279). There are eight months in the year, each of which have 35 days.

**void force\_start\_day(short which\_day)** - lets you manually set the day of the year the scenario starts. Which\_day is normally 0 to 279. If which\_day is -1, the scenario has no dates.

**short get\_current\_tick()** - Gets the number of ticks (i.e. turns/combat rounds) that have elapsed since the scenario began. This will be a number from 0 to 29999, and wraps around to 0 when past 29999.

**void set\_ticks\_forward(short num\_ticks)** - Manually changes the number of ticks that have happened since the scenario started (because the party sleeps, for example). num\_ticks can be negative.

**short tick\_difference(short time1,short time2)** - Returns the number of turns difference between time1 (the earlier time) and time2 (the later time).

**short what\_day\_of\_scenario()** - Returns how many days have passed since the current scenario began.

**short what\_day\_of\_year()** - Returns what day of the year it is (a number from 0 to 279).

### Miscellaneous Special Calls

**void create\_text\_bubble(string bubble\_text)** - This call only has an effect when called from a creature or terrain script. Creates a text bubble with text bubble\_text above the creature/terrain spot for a short time. The text can be at most 38 characters long. This call has no effect if called in combat.

If you pass empty text (i.e. create\_text\_bubble("")), the current text bubble is removed. Also, if there is already a text bubble on the creature/terrain spot, this call is ignored.

**void drop\_item(short which\_item)** - This call only has an effect when called from a creature or terrain script. Creates an item of type which\_item and puts it on the ground on the creature location or terrain spot. Even if the space has a container, the item will not be marked as contained.

**short get\_char\_who\_stepped\_on()** - This call is only used for custom terrain and floor types which call a state in the scenario script when stepped on (when the floor/terrain type's special property is set to 8).

If in town mode, returns -1 (the whole party is considered to have stepped on the space). If in combat mode, returns the number of the character who stepped on the space.

**void make\_wandering\_monst()** - Immediately spawns a group of wandering monsters. You can use this function in town or outside.

**void put\_object\_on\_waypoint(short which\_object,short which\_point)** - Creates an item of type which\_item and puts it on the ground on waypoint which\_point. Even if the space has a container, the item will not be marked as contained.

**short run\_bash\_door(short difficulty)** - This call only has an effect when called from a terrain script. Has the party member with the highest strength bash the object, and requires a strength of at least difficulty to succeed. Returns 1 if success, 0 otherwise.

**short run\_pick\_lock(short difficulty)** - This call only has an effect when called from a terrain script. Has the party member with a lockpick and the highest Tool Use attempt to manipulate the object, and requires a Tool Use (plus bonuses from items) of at least difficulty to succeed. Returns 1 if success, 0 otherwise.

**short set\_ter\_script\_mode(short which\_script,short what\_setting)** - For terrain script which\_script, sets how often it is run. Values for what\_setting are

- 0 - call every 8 moves, but not if party is far away
- 1 - call every 8 moves, even if party is far away
- 2 - call every move, but not if party is far away
- 3 - call every move, even if party is far away

**void set\_terrain\_memory\_cell(short which\_ter\_script,short which\_cell,short new\_value)** - For terrain script which\_ter\_script, sets cell memory which\_cell to new\_value.

**void text\_bubble\_on\_char(short which\_char, string bubble\_text)** - Creates a text bubble with text bubble\_text above creature which\_char. The text can be at most 38 characters long. This call will create a text bubble if called in combat. However, you shouldn't make a text bubble in combat unless a cutscene is happening.

If you pass empty text, the current text bubble is removed. Also, if there is already a text bubble on the creature/terrain spot, this call is ignored.

## Terrain Checking and Modification

Only a few of these calls can be used outdoors. In this case, {x,y} is a location in the current outdoor section. The rest of the calls only work in town mode. It is specifically noted if a call can be used outdoors.

Note that outdoor terrain is reloaded every time a save file outdoors is loaded. Changes made by set\_terrain will then be erased. Bear this in mind when altering outdoor terrain.

**void change\_blocked(short x, short y,short blocked)** - Change whether a space in town is blocked. If blocked, NPCs will not step on it and the party will not be placed onto it when the party enters a level or ends combat. If blocked is 0, the space becomes not blocked. If 1, the space becomes blocked.

Note that this does not affect if the floor or terrain in that space is set to be blocked.

This call does not work outdoors.

**short get\_floor(short x,short y)** - Returns the floor type at {x,y}. This call can be used outdoors.

**short get\_height(short x,short y)** - Returns the height at {x,y}. This call can be used outdoors.

**short get\_terrain(short x,short y)** - Returns the terrain type at {x,y}. This call can be used outdoors.

**void flip\_terrain(short x,short y)** - Change the terrain at spot {x,y} to its te\_swap\_terrain value (set in the object definition script). For example, sets a closed gate at {x,y} to be an open gate. If no te\_swap\_terrain value has been set for the terrain type at {x,y}, does nothing. This function will not work properly outdoors.

**short is\_blocked(short x,short y)** - Returns 1 if the given space is designated as blocked, and 0 otherwise. Note this doesn't check if the floor or terrain type in that space is blocked, just if the space was made blocked in the editor. This call doesn't work outdoors.

**short is\_field\_on\_space(short x,short y,short field\_type)** - Returns 1 if a field of type field\_type is on space {x,y}, 0 otherwise. The field types are:

- 0 - sleep cloud
- 1 - fire wall
- 2 - antimagic
- 3 - stinking cloud
- 4 - ice wall
- 5 - blade wall
- 6 - quickfire

**short is\_object\_on\_space(short x,short y,short object\_type)** - Returns 1 if an item of type object\_type is sitting on space {x,y}, 0 otherwise. The object types are:

- 0 - web
- 1 - barrel
- 2 - crate
- 3 - fire barrier
- 4 - force barrier
- 5 - ne/sw mirror
- 6 - nw/se mirror

**short is\_stain\_on\_space(short x,short y,short stain\_type)** - Returns 1 if a stain of type stain\_type is on space {x,y}, 0 otherwise. Stain types are as follows:

- 0 - small blood
- 1 - medium blood
- 2 - large blood
- 3 - small slime
- 4 - large slime
- 5 - dried blood
- 6 - bones
- 7 - rocks

**void put\_field\_on\_space(short x,short y,short field\_type)** - Places a field of type field\_type on space {x,y}. Field types are listed above. If field\_type is -1, removes all fields from the space.

**void put\_object\_on\_space(short x,short y,short object\_type)** - Places an object of type object\_type on space {x,y}, 0 otherwise. Object types are listed above. If object\_type is -1, removes all objects from the space.

**void put\_stain\_on\_space(short x,short y,short stain\_type)** - Places a stain of type stain\_type on space {x,y}. Stain types are listed above. If stain\_type is -1, removes all stains from the space.

**void set\_floor(short x,short y,short floor\_type)** - Sets the floor type at {x,y} to floor\_type. This call can be used outdoors.

**void set\_height(short x,short y,short height)** - Sets the height type at {x,y} to new\_height. This call can be used outdoors.

**void set\_terrain(short x,short y,short ter\_type)** - Sets the terrain type at {x,y} to ter\_type. This call can be used outdoors.

**void swap\_floor(short x,short y,short floor\_1,short floor\_2)** - If the terrain at spot {x,y} is floor\_1, turns it to floor\_2. otherwise, if it is terrain\_2, turns it to floor\_1. This function will not work properly outdoors.

**void swap\_terrain(short x,short y,short terrain\_1,short terrain\_2)** - If the terrain at spot {x,y} is terrain\_1, changes it to terrain\_2. otherwise, if it is terrain\_2, turns it to terrain\_1. This function will not work properly outdoors.

### Dialogue Calls

**void add\_string(short which\_string)** - This call only works when called inside the code portion of a dialogue node. Sets the string which\_string (which is from 1 to 8) to be displayed. This only has an effect if the string was already hidden by the remove\_string or clear\_strings calls.

**void begin\_shop\_mode(char shop\_name,char shop\_desc,short which\_shop,short price\_adjust,short sell\_adjustment)** - Immediately puts the party in shopping mode, buying from which\_shop, which has name shop\_name and description shop\_desc. price\_adjust is a number from 0 to 6 determining how expensive the shop items are (0 - cheaper, 6 - more expensive). sell\_adjustment is either -1 (which means the shop doesn't buy items) or 0 to 6, a number which determines how much the shop pays for items (0 means the store pays more, 6 means pays less).

**void begin\_talk\_mode(short start\_node)** - Begins talking mode, starting at node start\_node. This should not be called while outdoors, or while entering or leaving town.

(Advanced detail: If a conversation is started from a creature or terrain script, there is a Done button in the Dialogue area (so the player can leave the conversation at any time). Otherwise, there will be no Done button, and the conversation can only end when the player reaches the end of a talking thread.)

**short character\_talking\_to()** - This call only functions when called while in talk mode. Returns the number of the character the party is currently talking to.

**void clear\_strings()** - This call only works when called inside the code portion of a dialogue node. Hides all strings, so none of them are displayed. After this call is used, you should use the add\_string call at least once, so the player doesn't see an empty screen.

**short current\_personality()** - This call only functions when called while in talk mode. Returns the number of the personality the player is talking to.

**void remove\_string(short which\_string)** - Hides string which\_string (which is from 1 to 8), so it is not shown to the party when the dialogue node is displayed.

**void set\_char\_dialogue\_pic(short which\_char\_or\_group,short which\_sheet,short which\_icon)** - Gives the Creature/Group a dialogue picture, which will appear in the talk corner when they

are talked to. This should probably be called when the town is entered. Dialogue pictures are placed in a sheet by themselves with no frame, and it is recommended that they are no larger than 64 x 64. The number of the graphic resource is which\_sheet. which\_icon is not used and should be left at 0. For more information, read Chapter 4.5.

### **Basic Dialog Box Calls**

All of these calls are used to set up and display dialog boxes, where the player is given a host of options to select from. The dialog box you will build can have up to 6 chunks of text and 3 choices (each with text you provide).

To display a dialog box more complicated than just a chunk of text and an OK button (for this, use the message\_dialog call), follow these steps:

- i. Call reset\_dialog. This cleans up all the text and choices, and sets the first of the three choices to OK.
- ii. Fill in the text you want to appear. Use the call add\_dialog\_str once for each line of text in the dialog.
- iii. Set the text for up to three choices, using the call add\_dialog\_choice.
- iv. Use the call run\_dialog to make your dialog box appear and wait for the player's choice. The call will return which of the three choices the player made (1 for 1st choice, 2 for 2nd, and 3 for 3rd). Call run\_dialog(1) if you want there to be a Record button, and run\_dialog(0) otherwise.

**reset\_dialog()** - Erases all previous dialog box settings. Clears all text and all buttons, and then sets first button to say "OK".

**void add\_dialog\_choice(short which\_option,char choice\_text)** - Adds a choice to the bottom of the dialog. which\_option is the choice to add (0 to 2) and choice\_text is the text for the choice (which can be up to 255 characters).

**reset\_dialog\_preset\_options(short dialog\_options)** - Resets dialog box, and then sets buttons and dialog text to one of a list of presets. The meanings of dialog\_options are:

- 0 - empty text, 1 is ok button
- 1 - empty text, button 1 - leave it, button 2 - take it
- 2 - text for a lever, button 1 - leave, button 2 - pull
- 3 - text for a portal, button 1 - leave, button 2 - enter
- 4 - text for a button, button 1 - leave it, button 2 - push
- 5 - text for stairs, button 1 - leave, button 2 - climb
- 6 - text for scrolls, button 1 - leave, button 2 - read
- 7 - text for a book, button 1 - leave, button 2 - read
- 8 - text for a wheel, button 1 - leave, button 2 - turn it
- 9 - text for a trapdoor, button 1 - leave, button 2 - climb
- 10 - text for a trap, button 1 - leave, button 2 - disarm

**void add\_dialog\_str(short which\_str,char new\_text,short indent)** - Sets a line of text for the dialog you are about to display. which\_str is the line of text to set (from 0 to 5), new\_text is the text string (which can be up to 255 characters), and indent is the number of pixels to indent the line (usually 0).

**void message\_dialog(char text1,char text2)** - This call does two different things depending on whether it is used in or out of dialogue mode.

If not called from dialogue mode, simply displays a dialog box with the text text1 and text2, and the option to select OK. This dialog box will have a Record button.

If called from a dialogue node (or from a town or scenario script called by that dialogue node), this call forces the next text the character says to be text1, followed by text2. Each of these pieces of text can be at most 255 characters long. (This option is provided mainly to provide backward compatibility with Blades of Exile. This sort of call was how custom dialogue text was set in Blades of Exile, but Blades of Avernum has far superior ways to do it.)

**short run\_dialog(short record\_button)** - Actually displays the dialog box you have created using the previous functions. Expects the player to choose an option and then returns what was chosen (which will be 1, 2, or 3). If record\_button is 1, there will be a button in the dialog box which records the displayed text in the journal. If record\_button is 0, the text can't be recorded.

## Calls For Terrain and Creature Scripts

### Terrain Script and Creature Script Calls

These calls can only be made from a creature or terrain script. Note that none of these functions should ever be called outdoors.

**short can\_see\_loc(short loc\_x, short loc\_y)** - Returns 1 if there is a clear line of sight from the creature/terrain spot to space {loc\_x,loc\_y}, and 0 otherwise.

**void damage\_nearby(short damage\_amount,short radius,short damage\_type,short good\_evil\_or\_all)**  
- Does damage\_amount damage of type damage\_type to all characters visible to and within radius spaces of creature/terrain spot. If good\_evil\_or\_all is 0, only party-friendly creatures are damaged. If good\_evil\_or\_all is 1, only hostile characters are damaged. If good\_evil\_or\_all is 2, everyone nearby is damaged.  
Damage types - 0 - weapon, 1 - fire, 2 - poison, 3 - general magic, 4 - unblockable, 5 - cold, 6 - acid

**short dist\_to\_char(short which\_char)** - Returns the distance of the creature/terrain spot from character which\_char.

**short dist\_to\_loc(short x,short y)** - Returns the distance from the creature/terrain spot to location {x,y}.

**short dist\_to\_party()** - Returns the distance of the creature/terrain spot from the nearest party member.

**short floor\_in\_this\_spot()** - Returns the floor type the creature or terrain script is located on. This does not work outdoors.

**void force\_status\_nearby(short status\_amount,short radius,short what\_status,short good\_evil\_or\_all)** - For all characters visible to and within radius spaces of creature/terrain spot, changes what\_status by an amount equal to status\_amount. This change cannot be resisted. If good\_evil\_or\_all is 0, only party-friendly creatures are affected. If good\_evil\_or\_all is 1, only hostile characters are affected. If good\_evil\_or\_all is 2, everyone nearby is affected.

**short get\_memory\_cell(short which\_cell)** - It returns the value of that creature or terrain script's memory cell which\_cell.

**short get\_nearest\_char(short radius)** - Returns the number of the character nearest to the creature/terrain spot which the creature/terrain spot has a clear sight line to. Returns -1 if no such character.

**short get\_nearest\_evil\_char(short radius)** - Returns the number of the character nearest to the creature/terrain spot which the creature/terrain spot has a clear sight line to and which is hostile to the party. Returns -1 if no such character.

**short get\_nearest\_good\_char(short radius)** - Returns the number of the character nearest to the creature/terrain spot which the creature/terrain spot has a clear sight line to and which is friendly to the party. Returns -1 if no such character.

**void heal\_nearby(short heal\_amount,short radius,short good\_evil\_or\_all)** - Heals heal\_amount damage to all characters visible to and within radius spaces of creature/terrain spot. If good\_evil\_or\_all is 0, only party-friendly creatures are healed. If good\_evil\_or\_all is 1, only hostile characters are healed. If good\_evil\_or\_all is 2, everyone nearby is healed.

**short my\_loc\_x()** - Returns the x coordinate of the creature or terrain spot's location. This does not work outdoors.

**short my\_loc\_y()** - Returns the y coordinate of the creature or terrain spot's location. This does not work outdoors.

**void restore\_energy\_nearby(short energy\_amount,short range,short good\_evil\_or\_all)** - Restores energy\_amount spell energy to all characters visible to and within radius spaces of creature/terrain spot. If good\_evil\_or\_all is 0, only party-friendly creatures are energized.

If `good_evil_or_all` is 1, only hostile characters are energized. If `good_evil_or_all` is 2, everyone nearby is energized.

**`void set_memory_cell(short which_cell,short new_value)`** - It sets the value of that creature or terrain script's memory cell `which_cell` to `new_value`.

**`short set_script_mode(short what_setting)`** - For the currently running script (terrain or creature), sets how often it is run. Values for `what_setting` are

- 0 - call every 8 moves, but not if party is far away
- 1 - call every 8 moves, even if party is far away
- 2 - call every move, but not if party is far away
- 3 - call every move, even if party is far away

**`void status_nearby(short status_amount,short radius,short what_status,short good_evil_or_all,short penalty)`** - For all characters visible to and within radius spaces of creature/terrain spot, changes `what_status` by an amount equal to `status_amount`. For hostile effects, resistances apply. The higher `penalty` is, the harder the effect is to resist. If `good_evil_or_all` is 0, only party-friendly creatures are affected. If `good_evil_or_all` is 1, only hostile characters are affected. If `good_evil_or_all` is 2, everyone nearby is affected.

**`short terrain_in_this_spot()`** - Returns the terrain type the creature or terrain script is located on. This does not work outdoors.

### Terrain Script Calls

These calls can only be made from a terrain script. Note that none of these functions should ever be called outdoors.

**`short char_on_me()`** - Returns the number of the character standing on the terrain spot, or -1 if there is none.

**`short char_who_activated_script()`** - terrain scripts are called when a character steps on or searches them. This returns the number of the character that activated the script.

**`short get_mechanism_difficulty()`** - Gets the current complexity of the terrain script's mechanism. See the chapter on terrain scripts for more information.

**`short get_physical_strength()`** - Gets the current physical strength of the terrain script. See the chapter on terrain scripts for more information.

**`void set_mechanism_difficulty(short new_difficulty)`** - Sets the current complexity of the terrain script's mechanism to `new_difficulty`. See the chapter on terrain scripts for more information.

**`void set_physical_strength(short new_strength)`** - Sets the current physical strength of the terrain script to `new_strength`. See the chapter on terrain scripts for more information.

### Movement, AI Functions

These calls are used to move around and otherwise manipulate creatures. Some of them can only ever be used inside a creature script. In general, you will probably only ever use any of these in a creature script.

**`short am_i_doing_action()`** - This function can only be called in a character script. Returns 1 if the character currently has a movement path set or a hostile target, 0 otherwise.

**`short approach_char(short which_char_or_group,short char_to_approach,short within_dist)`** - Has the character move closer to character `char_to_approach`. If the character is within `within_dist`, does not move any closer. Returns 1 if the character doesn't start moving (because the target character is close, dead, or can't be reached, or because the character is already moving somewhere), 0 otherwise. If this function is called while the character is already moving, nothing happens and 0 is returned.

**`void approach_ter_script(short which_char,short which_ter_script, short distance)`** - Has the character move closer to the terrain spot with script `which_ter_script`. If the character is

within `within_dist`, does not move any closer. Returns 1 if the character doesn't start moving (because the target script doesn't exist or can't be reached, or because the character is already moving somewhere), 0 otherwise.

**short approach\_waypoint(short which\_char\_or\_group,short waypoint\_to\_approach,short within\_dist)** - Has the Creature/Group start to approach `waypoint_to_approach` until they are within `within_dist` spaces of it. Returns 1 if the character doesn't move (because it is close already or can't find a path there), 0 otherwise.

**void fidget(short which\_char\_or\_group,short fidget\_chance)** - If character is moving, nothing happens. If not, there is a percentage chance equal to `fidget_chance` that the character will select a visible nearby spot randomly and move to it.

**short flee\_char(short which\_char,short target\_char,short new\_dist)** - Has the Creature/Group move away from `target_char` until it is at least `new_dist` spaces away. Returns 1 if the character doesn't move (because it is the right distance already or can't find a good point), 0 otherwise.

**short maintain\_dist\_to\_char(short which\_char\_or\_group,short target\_char,short new\_dist)** - Has the Creature/Group move to a random point a distance close to `new_dist` from character `target_char`. Returns 1 if the character doesn't move (because it is the right distance already or can't find a good point), 0 otherwise.

**short move\_to\_loc\_x\_y(short which\_char\_or\_group,short x,short y)** - Has the character move as close as possible to location {x,y}. If the character is already moving, the old destination is forgotten. Returns 1 if the character doesn't move (because it is close already or can't find a path there), 0 otherwise.

*This call was added for Scenario Format Version 2, only use it with Mac version 1.1 or later or Windows version 1.0 or later.*

**short return\_to\_start(short which\_char\_or\_group,short what\_dist)** - Has the character return to the point it started at. Returns 0 if the character is not there and can move closer, returns 1 if it is there or if the path there is completely blocked.

If the character is currently moving, this call will leave the current destination unchanged and the call returns 0. If you want the character to return to start no matter what, use the `stop_moving` call first.

**void stop\_moving(short which\_char\_or\_group)** - Immediately makes the Creature/Group stop moving and forget the path it was traveling along.

### Combat and Targeting Calls

These calls are used to control creatures. Some of them can only ever be used inside a creature script. In general, you will probably only ever use any of these in a creature script.

**void do\_attack()** - This call can only be used inside a creature script, and only has an effect if the creature in question has a legitimate hostile target. In this case, the creature will select an attack and use it against the enemy. The attack might be a spell, a hostile ability, or a melee attack. If the target is far away and a melee attack is selected, the creature will move closer to its target. Action points for the creature's actions are deducted automatically. This call can be used even when not in combat.

**void do\_attack\_tactic(short which\_tactic)** - This call is exactly like `do_attack`, except that you have the option to force the character to strongly prefer a certain sort of attack. The options for `which_tactic` are:

- 0 - act normally (as if `do_attack` was used)
- 1 - strongly prefer firing missiles or using innate special ability
- 2 - strongly prefer to cast a spell
- 3 - strongly prefer to use an item in its inventory

**short get\_aggression(short which\_char)** - Returns the current aggression of `which_char`.

**short get\_courage(short which\_char)** - Returns the current courage of `which_char`.

**short get\_strategy(short which\_char, short dummy)** - Returns the current strategy of which\_char. See set\_strategy for the meanings of the different strategy values. The value of dummy is ignored.

**short get\_target()** - This call can only be used inside a creature script. Returns the current hostile target of the creature, or -1 if there is no target.

**short select\_target(short which\_char\_or\_group, short within\_range, short other\_criterion)** - Has the Creature/Group search for and set its hostile target to the best enemy within within\_range spaces. This only sets the hostile target. It does not actually make the character do anything. If an appropriate target is found, the call returns 1, 0 otherwise. If this character has been alerted (by seeing a foe or use of the alert\_char function), the character is hostile to the party, and no foes are visible within within\_range spaces, the character picks a random member of the party to be its target. It will then hunt the party down (possibly from a long way away).

The field other\_criterion allows you to give special criterion for when selecting the target:

0 - takes any target available.

1 - the target must be wounded

2 - the target must be heavily wounded (i.e. lost over half its health)

Advanced: The character's aggression (which always defaults to 100) affects whether the character can get a target. When the call tries to get a target, the aggression is the percentage chance that the character will be able to get a target (so if aggression is 75, has a 75% chance of getting a target and within\_range is reduced by a quarter). Also, the range is reduced to the aggression percentage (so if aggression is 75 and within\_range is 8, it is changed to 6).

**void set\_aggression(short which\_char\_or\_group, short new\_value)** - Sets the aggression value of the Creature/Group to new\_value, which should be from 0 to 100.

A note about aggression. If a character tries to get a target and its aggression level prevents it, the creature's turn ends immediately.

**void set\_courage(short which\_char\_or\_group, short new\_value)** - Sets the courage value of the Creature/Group to new\_value, which should be from 0 to 100. A character whose courage is 0 always flees.

**void set\_strategy(short which\_char\_or\_group, short new\_value)** - Sets the strategy value of the Creature/Group to new\_value, which has meanings as follows:

0 - Regular creature.

1 - Archer/caster. Always seeks to maintain a safe distance from its target.

10 - Regular creature, game never changes its target. (normally, the game will switch a creature's target to someone much closer)

11 - Archer/caster, game never changes its target.

**void set\_target(short which\_char\_or\_group, short what\_target\_or\_group)** - Manually sets the hostile target for the Creature/Group to what\_target. If the value what\_target\_or\_group is the number of a character (i.e. less than 120), sets the target to that character. If the value is for a group (i.e. 1000 + x, for group x), sets the target to be a random member of that group. If the target given is not legitimate (because the given creature is dead, doesn't exist, or is not hostile), the hostile target is set to -1.

**short target\_ok()** - This call can only be used inside a creature script. Returns 1 if the creature has a hostile target and it is alive, 0 otherwise.

**short who\_hit\_me()** - This call can only be used inside a creature script. It is used by a creature script to determine if the creature has been attacked. The call returns -1 if the character has been attacked, and otherwise returns the number of the creature who last attacked it. This value is remembered by the creature but it is set back to -1 at the end of every turn/combat round (so your script gets one chance to check this, and then it is set back to -1).

## Advanced Topics

### Advanced Dialog Box Calls

**void check\_text\_response\_match(char match\_text)** - Checks to see if the last text entered by the player (when you called get\_text\_response) matches the text match\_text. If so, the next time you call got\_text\_match, it will return 1. Otherwise, it will return 0. Note that the checking is not case sensitive, so "FRED", "Fred", and "fred" are considered to all match each other.

**short get\_selected\_pc()** - Returns the number of the last character selected by run\_select\_a\_pc, or -1 if no character has been selected.

Note that, whenever a script is called and starts to run, the last character selected is reset to -1. So, for example, if the party stepped on a space and triggered a town script and the very first call was get\_selected\_pc, the call would return -1, no matter what scripts had done earlier.

**void get\_text\_response(char top\_text)** - Brings up a dialog box asking the player to enter a chunk of text. The text top\_text will appear on the dialog box above the text entry field. To then check to see if the player entered a certain word or phrase, use the calls check\_text\_response\_match and got\_text\_match.

**short got\_text\_match()** - Returns 1 if the text entered by the player when you last called get\_text\_response matches the text checked for last time you called check\_text\_response\_match, 0 otherwise.

**void large\_draw\_pic\_dialog(short what\_pic, char label\_text)** - Brings up a dialog box that displays a large graphic (up to 400x400 pixels) you provide. The graphic will be labeled by the text label\_text, which will be at the top. This text can be, at most, 100 characters long. The graphic that is displayed is sheet what\_pic.

**short run\_select\_a\_pc(short select\_mode)** - Displays to the player a dialog box showing all of his or her party members. The player can either select a party member or the cancel button. The call returns 1 if the party selected a character, and 0 if the player hit Cancel. The character the player selected can be recovered using the call get\_selected\_pc.

The select\_mode field determines what characters are made available for selection:

0 - only living, present party members

1 - any party member, even a dead one

If select\_mode is 2, no dialog box is displayed. Instead, the choice the player made in any previous calls to run\_select\_a\_pc is set to -1 (no character selected).

**void small\_draw\_pic\_dialog(short what\_pic, char label\_text)** - Brings up a dialog box that displays a small graphic (up to 200x200 pixels) you provide. The graphic will be labeled by the text label\_text, which will be at the top. This text can be, at most, 50 characters long. The graphic that is displayed is sheet what\_pic.

### Grouping Calls

These calls are all used to manage groups: to make them, destroy them, and see who is in them. Remember that group 0 is reserved for the party, and that the other groups are numbered 1 to 7.

To use a call like set\_name or set\_level on a group, add 1000 to the number of the group and give that for the value of which\_char\_or\_group. (So to change the level of everyone in group 3 to 45, use the call "set\_level(1003,45)".)

**void add\_char\_to\_group(short which\_char, short which\_group)** - Places character which\_char in group which\_group.

**void add\_range\_to\_group(short first\_char, short last\_char, short which\_group)** - Places the range of characters from first\_char to last\_char into group which\_group.

**short char\_in\_group(short which\_char, short which\_group)** - Returns 1 if character which\_char is in group which\_group, 0 otherwise.

**void clear\_all\_groups()** - Removes all characters from all groups (except group 0).

**short first\_group\_member(short which\_group)** - Returns the first character in the group (the first character added to it), or -1 if the group is empty.

**short num\_chars\_in\_group(short which\_group)** - Returns the number of characters in group which\_group.

**short random\_group\_member(short which\_group)** - Returns a randomly selected member of group which\_group, or -1 if the group is empty. Remember that group 0 is the party.

**void remove\_char\_from\_group(short which\_char,short which\_group)** - Takes character which\_char out of group which\_group.

**short what\_group\_in(short which\_char)** - Returns the group the character is in, or -1 if the character is in no group. If the character is in multiple groups, returns the first one.

If this call is used in a creature's script, which\_char(ME) returns the group the creature is in.

### Messaging Calls

These calls are all used to pass messages to characters and terrain scripts, and for those scripts to check for and handle the messages. Remember that, right after a character's or terrain's script is run, its message is set back to -1, so if you don't check for a message once, the creature/terrain will miss it.

These calls have no effect outdoors.

**void broadcast\_char\_message(short message\_range,short what\_message,short must\_match\_attitude)**  
- This call only has an effect when called from a creature or terrain script. This call broadcasts to every visible creature within message\_range spaces of the creature/terrain spot the message what\_message. If the message is being broadcast from a creature script and must\_match\_attitude is 1, the message is only given to creatures with the same attitude as the one sending the message. If the message is being broadcast from a terrain script, must\_match\_attitude is ignored.

**void broadcast\_message\_from\_x\_y(short x,short y,short what\_message,short message\_range)** - This call gives message what\_message to every creature and terrain script that is within message\_range spaces of {x,y}, and can be seen from {x,y}.

**void broadcast\_terrain\_message(short message\_range,short what\_message)** - This call only has an effect when called from a creature or terrain script. This call broadcasts to every visible terrain script within message\_range spaces of the creature/terrain spot the message what\_message.

**void give\_char\_message(short which\_char,short what\_message)** - This gives message what\_message to character which\_char.

**void give\_ter\_script\_message(short which\_ter\_script,short what\_message)** - This gives message what\_message to terrain script which\_ter\_script. You can find out the number of a terrain script by selecting it in the editor.

**short my\_current\_message()** - This call only has an effect when called from a creature or terrain script. Returns the character/terrain spot's current message, or -1 if there is no message.

### Advanced Item Management Calls

These calls have to do with getting information on items in the party's inventory. You can search through a character's items, see what he or she has, and destroy or take them away. Please be very careful with these calls! People work very hard to assemble their gear, and it should not be taken away arbitrarily.

Each character has 40 item slots, numbered 0 to 39. The first 13 (numbered 0 .. 12) are equipped items. The rest are items in the pack.

The equipped item slots are numbered as follows:

0 - armor

- 1 - weapon
- 2 - missile
- 3 - boots
- 4 - helmet
- 5 - shield
- 6 - necklace
- 7 - bracelet
- 8 - ring
- 9 - gloves
- 10 - cloak
- 11 - pants
- 12 - ammunition

Important note: You can not use ME for which\_char for these calls. You must give the actual number for the creature. Inside a creature script, this can be gotten with the my\_number() call.

**void destroy\_char\_item(short which\_char,short which\_slot)** - Destroys the item held by character which\_char in slot which\_slot. Be very, very careful when using this call.

If you destroy an item in a pack (i.e. not equipped), all items in higher slots shift down.

**short get\_item\_variety(short item\_type)** - Returns the variety of item type item\_type (which is from 1 to 499). For the list of what the different item varieties are, consult the section on scripting item types.

**short item\_type\_in\_slot(short which\_char,short which\_slot)** - Returns the type of item held by character which\_char in slot which\_slot. The item type will be from 1 to 499, or -1 if there is no item in that slot.

**void take\_item\_char\_item(short which\_char,short which\_slot,short loc\_x,short loc\_y)** - Takes the item held by character which\_char in slot which\_slot and moves it to space {loc\_x,loc\_y} in the current town. Do not use this call outdoors.

### NPCs Joining Party Calls

These calls are used to assign characters to travel with the party. When not in combat, these characters will just follow the party around.

A character who joins the party must have its own script. When in combat mode, they will act according to its script. When do\_attack is called, the character will look for targets and attack them. It will not, however, move too far from the party when hunting targets.

**short add\_char\_to\_party(short which\_char)** - Attempts to take character which\_char and add it to the party. Character slots 4 and 5 are reserved for characters traveling with the party. If both of these slots are full, the call fails. You can't use this command when outdoors or in combat. It functions best when used during a conversation.

Returns 1 if the character is added to the party, 0 otherwise.

When a character is added to the party, its character ID remains unchanged. The calls character\_in\_party and remove\_char\_from\_party will refer to this character ID.

**short character\_in\_party(short char\_flag)** - If a character with character ID char\_flag is in the party, returns their character number (which will be 4 or 5). Otherwise, returns -1.

**short remove\_char\_from\_party(short char\_flag)** - Checks if a creature with character ID char\_flag is in the party. If so, erases that character. It disappears, and no items are dropped. Returns 1 if a creature is erased, 0 otherwise.

**short this\_char\_is\_in\_party()** - This call can only be used inside a creature's script. Returns 1 if the creature is part of the party, 0 otherwise.

### Splitting Up the Party Calls

You can split up the party, so the player only gets to play with one of his or her characters for a while.

You can only split the party indoors. You must make sure that the split character can't leave town without reuniting the party.

The call `try_to_split_party` splits up the party and moves the lone character somewhere else in the town. The `reunite_party` call restores the party at the location where they split up.

**void reunite\_party()** - If the party is not split up, this call does nothing. Otherwise, it immediately reunites the party at the point where they were split up.

**void split\_off\_one\_char(short dest\_x,short dest\_y,short telep\_noise,short which\_char)** - Splits up the party (so character `which_char` is active and the rest are absent). The lone active character is placed at `{dest_x,dest_y}`. If `telep_noise` is 0, the split is silent. If it is 1, a teleport noise plays.

**short try\_to\_split\_party(short dest\_x,short dest\_y,short telep\_noise)** - Splits up the party (so only one character is active and the rest are absent). Asks the player to choose a character, with the option of canceling. If the player cancels, the party is not split and the call returns 0. Otherwise, the lone active character is placed at `{dest_x,dest_y}` and the call returns 1. If `telep_noise` is 0, the split is silent. If it is 1, a teleport noise plays.

### Custom Item Ability Calls

These calls are only used for coding the effects of items with custom abilities. When such an item is used, it calls a state in the scenario script. These calls can only be made in that state.

**short who\_is\_custom\_item\_target()** - If the item is used on enemies (like wands), returns the number of the character it was targeted on.

**short who\_used\_custom\_item()** - Returns the number of the party member who used the item.

### Custom Character Ability Calls

These calls are only used for coding the custom special abilities. When such an ability is used, it calls a state in the scenario script. However, these calls can be used in any sort of script. There are 4 special ability slots, numbered 0 to 3.

**void change\_custom\_abil\_uses(short which\_char,short which\_abil,short what\_change)** - For character `which_char`, changes the number of uses of ability `which_abil` by `what_change` (which can be negative).

**short get\_custom\_abil\_uses(short which\_char,short which\_abil)** - For character `which_char`, returns the number of uses of ability `which_abil` the character currently has.

**void init\_special\_abil(short which\_abil,string abil\_name,short what\_state\_to\_call)** - This call should probably be placed in the state `LOAD_SCEN_STATE` in the scenario script. For custom ability `which_abil`, sets its name to `abil_name` and the state in the scenario script that is called when the ability is used to `what_state_to_call`.

**short who\_used\_custom\_abil()** - This call returns the number of the character in the party who used the custom ability.

### Cutscene Calls

These calls are used to manually create cutscenes, dramatic scenes where the player watches what goes on. These calls can position characters, move them around, shift the screen view, and pause the action. None of these calls can be used outdoors.

Note that calls that move characters or shift the view do not actually redraw the screen. You need to use the `force_instant_terrain_redraw` call for that.

When you have a cutscene and want to leave the party where you moved them, be sure to use a `block_entry` call.

**void erase\_text\_bubbles()** - Erases all text bubbles on all characters and terrain spots.

**void force\_instant\_terrain\_redraw()** - Immediately redraws all of the terrain. You need to use this call whenever you want the changes made by the Cutscenes calls to be displayed to the player.

**void force\_view\_center(short x,short y)** - Shifts the terrain view so its center is at {x,y}. Be sure that {x,y} is somewhere in the town visible to the party.

**void march\_party(short x,short y)** - Has your party walk a space in the animation. Moves the lead character to spot {x,y}, and has each following character move to the space vacated by the previous character.

**void pause(short tenths\_of\_second)** - Stops the game for tenths\_of\_second tenths of a second.

**void relocate\_character(short which\_char,short x,short y)** - Moves character which\_char to {x,y}. This call doesn't check whether the character can legally be there, so be careful not to move a character into solid terrain or onto another character. If the character is dead or doesn't exist, nothing happens.

**void set\_character\_facing(short which\_char\_or\_group,short direction)** - Changes a character which\_char's facing. Values for direction: 0 - north, 1 - nw, 2 - west, 3 - sw, 4 - south, 5 - se, 6 - east, 7 - ne.

**void set\_character\_pose(short which\_char\_or\_group,short pose)** - Sets which pose (version of the graphic) to use for the character. The change will not be visible until you use the force\_instant\_terrain\_redraw call. Note that not all poses are available for the smaller graphic template. When the cutscene is done, you should restore all poses to 0. The pose values are: 0 - standing. 1 - attacking. 2 - combat ready. 5 - sitting in chair facing north. 6 - sitting in chair facing west. 7 - sitting in chair facing south. 8 - sitting in chair facing east. 11-14 - death poses 1-4.

**void set\_total\_visibility(short mode)** - Changes how the game determines whether spaces out of view of the party are visible or not. If mode is 0, the player can not see spots of out view (the normal setting). If mode is 1, the player will be able to see every spot on the screen, whether visible to characters or not. This is used to show areas in cutscenes that are out of view. If you set every space as visible, be sure to set them as not visible before ending the cutscene.

### String Manipulation Calls

These calls are designed to give the scenario designer a lot more control over the text displayed to the player. They're a little complicated, but much more powerful.

Blades of Avernum maintains a string buffer. It is a 255 character long chunk of text in memory. You can clear this buffer, append chunks of text to it, and then dump the buffer into a string variable.

For example, here is a bit of code.

```
string sample_string;
```

```
clear_buffer(); // empties out the string buffer.
append_string("Hello, "); // puts this text into the buffer
append_string("friend."); // adds this text to the end of the buffer
get_buffer_text(sample_string); // the buffer is copied into string variable sample_string
print_str(sample_string); // prints "Hello, friend." to the text area
```

*These calls were added for Scenario Format Version 2, only use them with Mac version 1.1 or later or Windows version 1.0 or later.*

**void append\_char\_name(short which\_char)** - Appends the name of the character which\_char to the end of the string buffer.

**void append\_number(short num)** - Appends the number num to the end of the string buffer.

**void append\_string(string source\_string)** - Appends the text source\_string to the end of the string buffer. If the result would be longer than 255 characters, the text is truncated.

**void clear\_buffer()** - Empties the text buffer. Sets it to a blank string.

**void get\_buffer\_text(string dest\_string)** - Dest\_string must be a string variable. Copies the text in the buffer into dest\_string.

### Debugging Procedures

These are all debugging functions, used to provide information while you write a scenario. They probably should be removed before you actually release your scenario to the public.

**void print\_num(short num)** - Prints the number num in the text area.

**void print\_nums(short a,short b,short c)** - Prints the numbers a, b, and c in the text area.

**turn\_on\_debug\_mode()** - Enables debug mode for your scenario. To turn on debugging mode, type "}", followed by "=" (so a right bracket, followed by an equals sign). To turn off debugging mode, type an equals sign. Once on, you can use the following commands:

**Shift-'k'**: Kills all hostile monsters in the whole town.

**Shift-'m'**: Magic map. All creatures in the area will show up on the map. Type Shift-'m' again to turn this off.

**Shift-'g'**: Enter ghost mode. You will be able to walk through walls, and even through blackness. Type Shift-'g' again to turn this off.

**Shift-'h'**: Heals and restores spell energy for everyone in your group.

**Shift-'l'**: Add light. Gives the party 100 more turns of light.

**Shift-'i'**: Advances the time in the scenario one day.

**Shift-'s'**: Set special items. Dialog boxes will come up asking you for two numbers. The first number you give is the number of a special item. The second number is the number of that special item the party would be set to have (so if you enter 12 and 4, the party will have 4 of special item 12).

**Shift-'f'**: Set Stuff Done Flags. You will be asked for 3 numbers. The first two numbers are the coordinates of a stuff done flag. If the third number is -1, you will be told the value of that flag in the text area. Otherwise, that SDF will be set to the third number. (So if you enter 5, 23, and -1, then the text area will tell you the value of SDF(5,23). If you enter 5, 23, and 8, then SDF(5,23) will be set to 8.

This call should definitely be removed before you release your scenario.

Debug mode is automatically turned off whenever you load a saved game or enter a new scenario.