# Devil Whiskey Scripting Engine Interface

## Creating Custom Events

The scripting system for Devil Whiskey is an embedded Python interpreter. In order to create your own events, you need to follow the instructions laid out in this document. The process is easy but the results can be quite powerful.

Step 0: Familiarize yourself with the Python language.

Head over to www.python.org for information, tutorials, sample code, and references.

Step 1: Create an Event Module and Function

With your favorite text editor, open up a blank document and save it as "YourEventName.py". There is only 1 thing that every script absolutely, positively must have, and that is a function at the module level named "fireEvent" which takes a single parameter. This parameter is the address of the internal C++ scripting interface handler, but you don't need to bother yourself with those details. All you need to know is that this parameter is IMPORTANT! All of eventAPI module's functions require this parameter to be passed back to it.

Here's a simple event script:

```
def fireEvent(e):
   return 1
```

That's it! Of course, it doesn't do anything, but that's the beginning of the event framework.

Step 2: Make the Event Do Something

In order to call into the eventAPI module, you need to import its functions. The easiest way to do this is to insert the following line at the very top of your script:

```
from eventAPI import *
```

There are more efficient ways to get this done, but this is the most basic way to open up the Event API to your scripts.

There are many functions currently available in the eventAPI module, and more will be added as the game grows (see Appendix A for details on the currently available functions).

Now, you need to actually call into the Event API. We will make the game interface change to ask a simple question in the text area in the upper right. Change your

fireEvent(e) function to look more like this:

```
def fireEvent(e):
  stringInput(e, ["Is this a decent tutorial so far?"])
  return 1
```

Step 3: Add Your Event to the Map Events File

Please see the Map Editing tutorial for information about this step

Step 4: Try It Out!

Now, fire up the game and go to the place on the map where you added the event. When you enter the square, the interface should change to ask you the question we coded above.

## Event Script Details

There are a few things you should be aware of when creating your own events that are particular to the Devil Whiskey Event API.

First and foremost is the significance of the value returned from fireEvent(e). When this value is returned as a 1, the move that the player performed to bring the player's party into the event's map square is completed upon return. When a 0 is returned, then the player's party is moved to the location that was indicated as the exit square and facing.

For example, the Devil Whiskey events such as equipment drops will return 1, because the party needs to end up in the square where the event occurred, maintaining their current facing. The building-based events (such as when talking to Yeodaugh) return 0, because the party needs to be moved to the event's preset exit location and facing.

## Appendix A: The **eventAPI** Module

Here is a rundown of the currently implemented functions available to scripters, and their general operation. In each function, eventPtr is the parameter that was passed into fireEvent(e)… in other words, the "e".

showPicture(eventPtr, picture)
-Looks up the picture key given as the string parameter "picture" and displays it in the main viewport of the game.
-Returns 1 if the picture was found, or 0 if there was an error.

clearPicture(eventPtr)
-Resets the main viewport to the default 3D view.
-Returns None

playSpecAudio(eventPtr, audioTag)
-Looks up the audio track given as the string parameter “audioTag” and plays it on the Spec channel.
-Returns 1 if the sound was found and played successfully, or 0 if there was an error.

stopSpecAudio(eventPtr)
-Halts the audio being played on the Spec channel
-Returns None

optionInput(eventPtr, text, options, optkeys)
-Displays a list of strings and a list of options in the text portion of the game interface, and prompts the user for one of a list of possible actions. The “text” parameter is a list of strings which will be displayed. The “options” parameter is a list of strings which will be displayed below the “text” strings. The “optkeys” parameter is a list of SDLKey integers which indicate which keys are valid for the options.
-This function will not return until the user presses one of the keys listed in “optkeys”.
-Returns the SDLKey code for the key that was pressed

stringInput(eventPtr, text)
-Displays a list of strings and a prompt for (nearly) free-form user input in the text portion of the game interface.
-This function will not return until the user presses the “Enter” key.
-Returns the text that the user entered before pressing “Enter”.

getGameTime(eventPtr)
-Returns the internal game tick number. Each tick represents 15 seconds of elapsed game time. The game starts at tick 316946160.

setTag(eventPtr, tag, val)
-Sets the game-engine-internal global variable given by the string parameter “tag” to the 32-bit integer value given in the “val” parameter. If the variable did not previously exist, it is created and initialized to the given value.
-Returns 0 if successful, or –2 if the variable indicated was actually read-only
(Mention something about the read-only variables as well as the special vars?)

getTag(eventPtr, tag)
-Returns the 32-bit integer value stored in the game-engine-internal global variable given by the string parameter “tag”. There are several read-only variables that are internal to the game engine and are set based on game events. These include:
‘Combat’, which is set to 0 initially, changed to 1 if the party won the most recent combat, 2 if the party successfully ran away, or 3 if the party perished.
‘Chest’, which is set to 0 initially, changed to 1 if the chest was opened, or 2 if the party left the chest unopened.

'lFlagU' (that's a lower-case L, not a one), is a special "local" variable which is unique to each *instance* of an event.
'lFlagS' (that's a lower-case L, not a one), is a special "local" variable which contains game engine status for the current square. The bits of this value are defined as follows:

| **Bit** | **Description** |
|---|---|
| 0 | Set if the cell has been walked on. This will always be true – the event will not be called until the cell is walked on. |
| 1 | Set if the cell traps should be tripped. This allows traps to know whether or not they should go off. |
| 2 | Set if the cell spinner traps should be tripped. As above, note spinners are separate from normal traps. |
| 3-7 | Reserved |
| 8 | Set on first entry to a cell. The event may be called a number of times while the party is in a given cell – this bit will only be set the first time. |
| 9 | Set on first rotation in a cell. As above – this bit is set when the party turns in the cell, then cleared once all events in the cell have been executed. |
| 10-31 | Reserved |

tickGame(eventPtr, ticks)
-Cause the game to tick forward the number of ticks indicated in the integer parameter "ticks".
-Returns 0

getPartyCount(eventPtr)
-Returns the number of members in the party: living, dead, and monster.

isPC(eventPtr, partyIdx)
-Returns whether the party member in the slot indicated by the (zero-based) integer parameter "partyIdx" is a PC or not. A return of 1 means that it is a PC, a return of 0 means it is not a PC.

hasItem(eventPtr, partyIdx, item)
-Returns whether or not the party member in the slot indicated by the (zero-based) integer parameter "partyIdx" has one or more of the item named in the string parameter "item". A return of 1 means that the party member has one or more of the item, a 0 means s/he/it does not.

modifyXP(eventPtr, partyIdx, amount)
-Modifies the party member's experience points (in the slot indicated by the (zero-based) integer parameter "partyIdx") by the integer amount given in "amount". Reductions in XP will not result in loss of level at this time.
-Returns whether or not the modification was successful, as a 0/1 truth value.

modifyHP(eventPtr, partyIdx, amount)

-Modifies the party member's hit points (in the slot indicated by the (zero-based) integer parameter "partyIdx") by the integer amount given in "amount".
-Returns whether or not the modification was successful, as a 0/1 truth value.

modifySP(eventPtr, partyIdx, amount, toZero)
-Modifies the party member's spell points (in the slot indicated by the (zero-based) integer parameter "partyIdx") by the integer amount given in "amount". If the amount is less than zero, and the toZero parameter indicates integral truth (is non-zero), then the party member's spell points will be clamped to 0.
-Returns whether or not the modification was successful, as a 0/1 truth value.

modifyBP(eventPtr, partyIdx, amount)
-Modifies the party member's bard points (in the slot indicated by the (zero-based) integer parameter "partyIdx") by the integer amount given in "amount". If the amount is less than zero, and the toZero parameter indicates integral truth (is non-zero), then the party member's bard points will be clamped to 0.
-Returns whether or not the modification was successful, as a 0/1 truth value.

modifyGold(eventPtr, partyIdx, amount)
-Modifies the party member's gold (in the slot indicated by the (zero-based) integer parameter "partyIdx") by the integer amount given in "amount". If the amount is less than zero, and the toZero parameter indicates integral truth (is non-zero), then the party member's gold will be clamped to 0.
-Returns whether or not the modification was successful, as a 0/1 truth value.

callEvent(eventPtr, evtNum, argv)
-Calls an internally defined core game event. This will be documented in a future Appendix.
-Returns the integer value returned by the internal event call.

setExit(eventPtr, mapName, x, y, rot)
-Sets the current event's exit location and facing to the integer values indicated in "x", "y" and "rot". "Rot" should be 0, 90, 180 or 270, and the x, y coordinates should be appropriate for the map given in the string parameter "mapName". If necessary, the appropriate map will be loaded upon the end of the event.
-Returns None.

savingThrow(eventPtr, saveType, partyIdx, modifier)
-Executes a saving throw for the type of save given in the string parameter "saveType" against the party member whose slot is indicated by the integer parameter "partyIdx" with the integral modifier given in "modifier". Valid saveType values are: fire, water, earth, air, dex, con, paralyzation, poison, death, stoning, polymorph, breath, spells, psionics.
-Returns a 0/1 truth value indicating whether or not the save was successful.

removeItem(eventPtr, partyIdx, item)

-Removes one instance of the item named by the string parameter “item” from the party member whose slot is indicated by the integer parameter “partyIdx”.
-Returns a 0/1 truth value indicating whether or not the item removal was successful.

combat(eventPtr, enemies)
-Begins a combat session against the enemies listed in the special “enemies” parameter. “Enemies” must be formatted as a list of tuples, each tuple having 4 elements in this order: a string indicating the monster type, the base number of enemies in this group, a maximum number of enemies to add to the group (randomly chosen at combat start), and a range for the group to begin at. For example:

combat(e, [ ( “Orc”, 10, 5, 30 ) ])

will cause 10-15 Orcs to appear at 30’ range. Range must be divisible by 10. Multiple groups of the same type at the same range will be automatically combined.
-Returns an integer indicating the outcome of the combat: 1 indicates a party win, 2 indicates that the party ran, and 3 indicates that the entire party perished.

poison(eventPtr, partyIdx, dice, dieType, extra, tickStride, saveCure, duration)
-Causes the party member in the slot given by the integer parameter “partyIdx” to become poisoned. The poison’s strength is given by the “dice”, “dieType” and “extra” integer parameters, and is calculated as (dice)d(dieType)+extra. The poison takes its toll every “tickStride” ticks (must be integral) until “duration” ticks (must be integral) have passed. “saveCure” is a 0/1 truth value dictating whether or not a successful saving throw will cure the poison. This saving throw is re-rolled before each tick worth of damage is delivered. If this value is set to false, no saving throws are given.
For example:

poison(e, 0, 1, 6, 2, 4, 0, 20)

will cause 1d6+1 hit points worth of damage every 4 ticks (one minute) for 20 ticks (5 minutes), with no chance for natural curing through saving throws.
-Returns None

giveItems(eventPtr, itemList)
-Gives the items named in the list of strings in parameter “itemList” to the party. This shows the “treasure” image in the main viewport and gives the player the item distribution interface.
-Returns None

chest(eventPtr, chestType, difficulty, gold, itemList, randItemCount, text)
-Runs an internal chest event. The chestType integer parameter indicates the style of chest that will appear (a -1 indicates that there should not be a chest, just a pile of gold and items). The difficulty integer parameter indicates how difficult the chest is to open/untrap. The chest will contain an amount of gold as indicated by the integer “gold” parameter. It will also contain the items named in the list of strings parameter “itemList”,

as well as a number of random items as indicated by the integer parameter "randItemCount". The text given by the string parameter "text" is displayed during the event as the chest description.
-Returns an integer indicating the action that was performed by the party: -1 (indicating an internal error), 1 (indicating the party opened the chest), 2 (indicating the party left the chest untouched), or (-100 – N), where N is an index into itemList indicating an invalid item name.