Event Database

Basics

For events can be used in a map, the current scene name must be set. Do this in the OnEnter method of the map, and easiest is to use the helper function: *EventDB_SetCurrentScene(...)*.

The best way to manage the events, is to keep the owners and scenes (and any global events) in a global file. And then load all other events per level in the OnEnter method (before setting the current scene).

For the stand facts and triggers to work, EventDatabase/UseStandardTriggers must be set to true in game.cfg.

File format

Events can be added in code, but it is almost always better to use the files to define them. An event database file is in XML and has the extension “event”. The general format is like this:

```
<EventDatabaseData>
  <Owners>
    <Owner [Properties] />
    ...
  </Owners>
  <Scenes>
    <Scene [Properties] />
    ...
  </Scenes>
  <Events>
    <Event[Properties] />
    ...
  </Events>
</EventDatabaseData>
```

For information on the Properties that can be used, see below.

Properties

Command String Formatting

For both Criteria and Actions (see below), a simple syntax must be used when setting up the values.

The basic syntax is:
[varname][operator][value]
Examples:

```
Velocity>32
Velocity==3
MyName=Dennis
Street='Downtown 78b'
```

Two important things to note here:
- You can use both “=” and “==” for comparing values.
- When writing strings you can skip ' or “ if there are no spaces in the string.

Another way writing is:

```
MyVariable
```

This checks if the variable exists and is set to 1. If using it for an actions, it creates the variable and sets it to 1.
this means that the follow statements are equal:

```
IsAtHome
IsAtHome=1
```

Any variables that do not exist get the value 0 at a check, so

```
IsAtHome=0
```

is true even if the variable has never been set.

To have many statements, simple just sepperate them by spaces, like:

```
IsAtHome=0 Fuel>25 LastNote='In the woods'
```

Owner

Owner is where an event belongs. Is basically only there to easily subdivide and sort the events.

ID
The id of the owner, this must be unique and mostly only be set by the tool and never exposed.

Name
Name of the owner.

GroupFlags
This is a bitflag container that can be used to set different owners into groups (and make it possible to send queries to owners of belonging to certain groups only).

Scene

Scene is basically like owner, and yet another way to subdivide events.

ID
The id of the scene, this must be unique and mostly only be set by the tool and never exposed.

Name
Name of the scene.

Event

The event is basically the important payload of the database. The following is a list of all properties of an event and what they do.

OwnerId
The id of the owner connected to this event.

SceneId
The id of the scene connected to this event. If -1 then this is a global event that will be check in every scene.

Trigger
This is the trigger for this event. See below for some standard triggers. Triggers can also be user defined though (for instance as callback whena voice line is over).

Name
The name of the event, only used to easier track-keeping.

Criteria
This outlines the facts that need to be true for the event to happen. See “Command String Formatting” above for general info. Available operators are:

```
== (=), >, >=, <, <=, !=
```

Also, it is possible to check intervals using the syntax:
[varname] ([min] [max])
and
[varname] [[min] [max]]
”()“ includes the min/max into the interval (an open interval), and ”[]“ excludes the min,max (a closed interval). So for instances if the value is 0, then (0 10) is true, but [0 10] i false.
Examples:

```
Prisoners(1 4)
Health[0.1 0.75]
```

Actions
This outlines the actions made to the facts when the event is chosen in a query. See “Command

String Formatting" above for general info. Available operators are:

```
=, +=, -=, /=, *=
```

Output
The output string, depends on the type of output.

OutputType
This type is user defined, but the default types are:
0: A voice subject
1: A description. Output is a category and entry in the lang file.*Syntax: "[category] [entry]"*
2: Script callback function. *Syntax: "[Output](const tString& in asEventName)"*

OutputDelay
How long (in sec) the output of the event happens.

MaxRepetitions
The number of times an event can be repeated. 0 means unlimted number of times. Max number is 255.

Standard Triggers

Standard trigger types that can be checked for in any map. Below the name are the facts that are related to the triggers. Note that these must be enabled (in Player or any other module that triggers them for them to happpen).

PlayerCollide
CollideEntity: Entity that is collided with.
If the player collides with an entity. Note that this one is not checked every update and might miss very brief collisions. Mostly useful to check if player is in an area.

PlayerLeaveCollide
CollideEntity: Entity that player has collided with.
Triggered when player stops colliding with an entity. Mostly useful to check if player leaves an area.

PlayerInteract
InteractEntity: Entity that is interacted with.
If the player interacts with an entity. Note that this is triggered through the basic script.

PlayerLookAt
LookAtEntity: Entity that is look upon.
LookAtDistance: The distance to the entity that is looked upon.
LookAtDuration: The amount of time the player has looked at the entity.
If the player looks at an entity. Note that this is called about every 0.5 secs or so while inside the gaze of the player. This the first time it was triggered it might have been out of range.
Note that if you want to do this check on an area, make sure it has BlocksLineOfSight set to true!

VoiceLineOver
VoiceSubject: The name of the subject, syntax = [Character]_[Scene]_[Subject], eg

“Simon_00_01_Apartment_Greeting”.
VoiceLine: The name of the line, note that if line name is ”“, then this will be the index, eg “0”, “1”, etc.
When a voice line is over.

VoiceSoundOver
VoiceSubject: The name of the subject, syntax = [Character]_[Scene]_[Subject], eg “Simon_00_01_Apartment_Greeting”.
VoiceLine: The name of the line, note that if line name is ”“, then this will be the index, eg “0”, “1”, etc.
VoiceSoundIndex: The index of the sound that was just completed.
When a voice sound file is over.

VoiceIdle
VoiceIdleDuration: The time the voice has been idle.
VoiceCharacterIdleDuration_[charater]: The idle time for a specific character. Called every 3rd seconds when there is no voice playing.

Standard Global Facts

The following are facts that are always accessible. Note that some of this are related to specific triggers, and should only use for these.

CollideEntity
Contains the entity that spawned a *PlayerCollide* trigger. Also used by *PlayerLeaveCollide* and contain what the player used to collide with then. If player is not colliding with anything, it is ”“ (it is reset after the PlayerLeaveCollide trigger).
Additional facts: *CollideEntityTag*, *CollideEntityInstanceTag*. These are the tag values from the entity.

InteractEntity
Contains the latest entity that spawned a *PlayerInteract* (see above).
Additional facts: *InteractEntityTag*, *InteractEntityInstanceTag*. These are the tag values from the entity.

LookAtEntity
Contains the entity that spawned a *PlayerLookAt* trigger. If player is not looking at anything, it is ”“.
Additional facts: *LookAtEntityTag*, *LookAtEntityInstanceTag*, these are the tag values from the entity. *LookAtDistance*, this is the distance to the viewed object. *LookAtDuration*, how long the player has looked at the object.

VoiceSubject
The last sound line that was over. Used by *VoiceLineOver* and *VoiceSoundOver* triggers. syntax = [Character]_[Scene]_[Subject], eg “Simon_00_01_Apartment_Greeting”.

VoiceLine
The last sound line that was over. Used by *VoiceLineOver* and *VoiceSoundOver* triggers.

VoiceSoundIndex
The index of the sound that was over. Used by *VoiceSoundOver* trigger.

VoiceIsPlaying
Is 1 if a voice is currently playing, else it is 0.

VoiceIdleDuration
The amount of time since there was a voice playing (messured in whole seconds only).

VoiceCharacterIdleDuration_[charater]
How long it has been since a certain character has spoken. [character] is simply the name of the character, eg: “VoiceCharacterIdleDuration_Jack”.

EventIsQueued
If an event has been chosen but has not be been activated (meaning outputdelay time is not yet reached) this will be 1, else 0.

Custom Facts and Triggers

Custom triggers can currently be made in two ways:

1. To call the Query or QuertToAll in the EventDatabase. This will let you choose the name any name at all for a trigger.
2. Using the EventCallbackTrigger property for a Voice Line, this will query the event database when the line has finshsed playing.

To set your own facts (apart from setting them in action part of an event), you can call the SetFactXXX functions that exist in either cMap or in the cEventDatabaseHandler. If set in cMap, the fact will be local and only last the current map. Else it will be global and last for the entire game.

The helper file “helper_eventdb.hps” contains helper functions for all of the above.