The Quest editor

- system architecture –

The Quest architecture

Object types

The Quest engine contains “types” for every single entity. Like item types, monster types, disease types, and so on. Sometimes a type needs a base type, of course these have different properties, and more types can have the same base type.

Type	***Type contains***	***Base type***	***Base type contains***
Picture type	ID, the picture data itself	-	-
Palette type	ID, The palette data itself	-	-
Item type	ID, Palette IDs, properties (weight, ac, …), enchantment, enchant storage, …	Item base type	ID, Picture IDs (inventory and drop), and property hints for its item types
Monster type	ID, Palette IDs, properties (ac, hp, attributes, …), random items, spells/abilities	Monster base type	ID, Picture IDs (frame pictures), property hints for its monster types, sound effect IDs
Map Object type	Picture IDs, type, flags, sound effect IDs	-	-
Ability type	ID, enchant entries	-	-
Alchemy Recipe type	ID, needed effects, target potion properties	-	-
Disease type	ID, enchant entries	-	-
Global type	ID, value	-	-
Interior picture type	ID, Picture frame IDs, Music ID	-	-
Face type	ID, Picture ID	-	-
NPC type	ID, properties (face ID, shop/services, race, …), dialogue	-	-
Quest type	ID, description	-	-
Scene type	ID, music ID, views (picture IDs, text)	-	-
Script type	ID, compiled code	-	-
Sky type	ID, picture ID, cloud, fog color, …	-	-
Sound wav type	ID, wav data itself	-	-
Sound mod type	Header, patterns, …	Sound mod base type	Samples
Spell effect type	ID, picture IDs	-	-
Spell type	ID, move/explode effect ID and sound ID, properties	-	-

There are some more types, f.e. skill, race and attribute types – but these are restricted.

Identifiers in the game engine

Every identifier contains its world's ID. The base game has more "worlds":

World ID	***Contains***	***Filename***
base	Maps, monster types, item types, and more	TheQuestBase
bres	Pictures, palettes, monster base types, item base types, and more	TheQuestRes
bsnd	Sound wav types	TheQuestSound
bmus	Sound mod and mod base types	TheQuestMusic

The "base" world uses some types from the "bres" world, and so on.
An identifier in the base world could be "base_something", this means every single "type" ID's first 4 characters must be the world's ID (then there must be a _ sign).

These IDs are unique **in the actual types**'s array. F.e. there could be a "base_monster1" monster type and base monster type, or global, anything. Just there cannot be two monster types with the same ID.

Map architecture

There are two types of maps in the game engine, "dungeon maps" and "surface maps".

A dungeon map always contains 35x35 positions.
A surface map contains 21x21 positions, and the game loads the surrounding "border" positions from 8 different surface maps – so the game always use the 35x35 as map size.

Every single map contains ID, the default background (floor) and ceiling (for dungeon maps) map object type IDs, properties (flags, fog color, view distance, …), and the following collections:

Collection	***Entry***
Objects	Map objects, some "Map Object types" need existing map object. See below.
Monsters	Each monster specifies the current HP, flags and the monster's monster type
ItemLists	It contains items, and each item specifies the item type, the enchantment and the quality

Npc dialogue system

Every npc in the game can have *greeting topics* and *talk topics*.
Each topic (both *greeting topics* and *talk topics*) has *responses*, and talk topics has a *name*. This name will be displayed at the right side of the screen, so the player can choose any of them (ask about something);.
Each *response* has up to 5 *conditions*, and the engine picks the first response where all the

conditions are true. Condition description: see below.
And of course each response has a *result text, result answers* and *result script.*
Result text: the displayed text. It isn't a script! No message commands, just a huge edit box in the Editor. So you can enter anything, and can use macros (to include something in the text – description: see later).
Result answers: available answers for the response.
Each result answer has a condition (one condition for each answer). It isn't a script, but edit boxes in the editor.The game won't display an answer if its condition is false.
Result script: the script, at least ☺
These will be small scripts, and will be executed after the actual response is displayed. Shouldn't contain other things just add quest, reward, set globals, and so on (the logic of the response).

This system, as you can see, replace the ugly and uncomfortable if – else – if – else architecture. Every script in Legacy could be converted into the new architecture (message -> result text, answer -> defined answers, logic after messages -> result script).

How it starts…

When the player starts a conversation with an NPC, the engine picks a greeting from the greeting list of that NPC. The greeting list has *topics* which can be local ones (defined for this NPC) or shared ones (defined in the "shared dialogues" section, but this NPC use that as well).

The engine always start with the first greeting *topic*, so it must be well-organized by importance and priority. For example: if the player is criminal, then the NPC should mention that instead of the outfit.

The *shared greetings* can have many entries, and of course these won't be used for all NPCs. For example:
You define a shared topic, and set all the NPCs in a city to have that (but no others!). Now the NPCs in that town could tell you about city (or area) – related info.

Conditions in the system

All the responses and answers may (should) have conditions.
A condition looks like this:

Content:

First box	***Second box***	***Description***
Always	-	Always true. You can use this for answers, to make them visible every time. For normal responses, it doesn't have any meaning.
Answer	-	Returns the selected answer's number. 0 if there hasn't been any answers to be chosen, otherwise 1-4. Important: if a response doesn't have Answer as a condition, but the Answer is not 0, the response won't be displayed.
PCCrime	-	Returns the the PC Crime Level (the bounty value). 1. City guards will catch you if this is above 0. 3. Availabe range: 0 – 1000. Only serial killers could have this above 100. 4. Certain NPCs could take care of this "problem", for money of course. 5. The money to be payed when the guards catch the PC: Crime Level * 100
PCHasCrimeGold	-	Whether the player has enough money to pay (when guards catch him). Crime Level*100,
PCFame	-	Fame of the PC, the starting value will be 0 of course. Range: -100 .. +100

		If the player does good things then the fame should increase, otherwise decrease. So this is a kind of “alignment”: -100: pure evil, 0: neutral, +100: saint.
PCGood	-	PCFame>0
PCEvil	-	PCFame<0
PCOutfit	-	Outfit quality. Valid range: 0 – 100. 0: naked, 100: hiq armor. This value is updated frequently (when the player wears/wields/removes something).
PCHPPercent	-	HP/Max HP Percent of the PC (0-100)
PCLevel	-	Level of the PC (1-)
PCRace	Race	If the player’s race is the same as the selected
PCUndead	-	Whether the PC is undead (the Race is “Rasvim”)
PCAc	-	Player character’s armor class.
PCAttribute	Attribute name	PC’s attribute value (strength, dexterity, …)
PCSkill	Skill name	PC’s skill value (block, heavy weapon, …)
PCSameRace	-	The PC’s race is the same race as the NPC’s.
PCHasItem	Item type ID	The number of items the PC has of the specified itemtype
PCDiseased	Disease ID	Whether the PC is diseased (the Disease ID can be empty – means any diseases)
PCKnowsSpell	Spelltype ID	Whether the PC knows the spell
PCHasSavedXp	-	Whether the PC has saved XP (the removed XP can be marked as “can be restored”). It can be restored with a specific script function.
PCHasGold	Gold value	Whether the PC has the specified amount of gold
PCHasAbility	Ability ID	Whether the PC has the specified ability
PCWearWield	Item type ID	Whether the PC wears/wields an item with the specified item type
PCPoisoned	-	Whether the PC is poisoned
PCParalyzed	-	Whether the PC is paralyzed
PCCursed	-	Whether the PC is cursed
PCQuestReceived	Quest ID	Whether the PC has received the quest
PCQuestCompleted	Quest ID	Whether the PC has received the quest, and the quest succeeded.
PCQuestFailed	Quest ID	Whether the PC has received the quest, but the quest failed
Random100	-	Random number between 0 and 99 (inclusive)
ObjState	Map Object ID	State of the specified Map Object
ObjThisState	-	State of the current Map Object
NPCLikesPC	-	Whether the actual NPC likes the PC
WeatherCloud	-	The actual cloud (0: clear sky, 100: cloudy)
WeatherRain	-	The actual rain (0: no rain, 100: heavy rain)
TimeHour	-	Hour of the current day
TimeDay	-	Current day no
Daylight	-	The hour is between 6 and 21 (inclusive)
Night	-	The hour is between 22 and 5 (inclusive)
CurrentMap	Map ID	Whether the current map is the specified
CurrentWorld	World ID	Whether the current world is the specified
Global	Global ID	Value of the specified Global
DiffTimeHour	Global ID	The difference between the current time and the specified global’s value (in hours)
DiffTimeDay	Global ID	The difference between the current time and the specified global’s value (in days)
MonsterDead	Map object ID	Whether the monster is dead (specified by the monster’s object)
MonsterAggressive	Map object ID	Whether the monster is aggressive (specified by the monster’s object)
MonsterPoisoned	Map object ID	Whether the monster is poisoned (specified by the monster’s object)
MonsterDiseased	Map object ID	Whether the monster is diseased (specified by the monster’s object)
MonsterParalyzed	Map object ID	Whether the monster is paralyzed (specified by the monster’s object)
MonsterCursed	Map object ID	Whether the monster is cursed (specified by the monster’s object)
ObjectListResult	-	Whether an ObjectListResult exists
PCMale	-	Whether the PC is male
CurrentMap	Map ID	Whether the current map is the specified
CurrentWorld	World ID	Whether the current world is the specified
MonsterThisAggressive	-	Whether the current monster is aggressive
MonsterThisPoisoned	-	Whether the current monster is poisoned
MonsterThisDiseased	-	Whether the current monster is diseased
MonsterThisParalyzed	-	Whether the current monster is paralyzed
MonsterThisCursed	-	Whether the current monster is cursed
MonsterThisBaseType	Monster Base Type ID	Whether the current monster belongs to the specified monster base type
MonsterThisType	Monster Type ID	Whether the current monster belongs to the specified monster type

Macros in responses

In the **result text**, you can use macros.
Every macro looks like this: %Something%. The system will change that to the meaning. If you want to display a % sign in the text, use %%.

Macro	*Meaning*
%PCName%	PC's name
%PCRace%	PC's race
%PCCrimeGold%	100*PCCrime
%PCGold%	PC's Gold
%Day%	The actual day
%Date%	Something like "Day 2, 6:02 PM"
%Global:global_id%>	Value of the specified global
%PCDisease%	One of the diseases (if any) the player has been infected with. Can be empty string, of course.
%Map%	Current map's name
%AAN%	"a" or "an"
%CAAN%	"A" or "An"
%NPCName%	The actual NPC's name

Item architecture

Like everything else in The Quest, item types have IDs. Every single items belongs to an "Item type", and every item type belongs to a "Base item type".

Item classes/subclasses

The item classes and subclasses in the game are the following:
- **Weapon**: Hand (fixed type), Short sword, Long sword, Mace, Axe, Hammer, Club, Magic staff, Throwing, Short bow, Long bow, Quiver, Crossbow (not implemented), Bol quiver (not implemented)
- **Armor**: Shield, Armored pants, Armor, Helm, Gauntlets, Boots, Cloak, Belt
- **Light armor**: Shield, Armored pants, Armor, Helm, Gauntlets, Boots, Cloak, Belt
- **Accessory**: Amulet, Ring
- **Book**: Book, Letter, Map
- **Alchemy**: Mortar/pestle
- **Alchemy ingredient**: Ingredient
- **Potion**: Potion
- **Magic**: Scroll, Spellbook, Blank scroll, Wand
- **Money**: Money
- **Key**: Key, Lockpick
- **Repair**: Hammer
- **Misc**: Misc
- **Comestible**: Food, Water
- **Gem**: Gem
- **Card**: Card

Base item type

The base item types contain the bitmaps, and the default base data for its item types.
When you make a new item type, you select a base item type as a parent.
Every item base type specified its "item class" and "item subclass".

Item type

The item types are the actual entities you work with.
Let's say there's a Blade, with the ID of "base_weapon_blade".
If you want to make a blade which needs to have +1 minimal and +1 maximal damage, you make a new item type, name it as f.e. "base_weapon_bladespecial", and you modify the damage values. So now the Blade base item type has 2 children, "base_weapon_blade" and "base_weapon_bladespecial".
The item type has exactly the same "item class" and "item subclass" as its parent.

Item

Every item belongs to an item type, that's all. You can't modify an item, just an item type.

Checking the inventory

You can check whether the inventory (of the player, of a shelf, of a monster, …) contains the specified "item type".

If(player.hasitem("base_weapon_bladespecial",1))

… means … does the player have at least one item with the item type of "base_weapon_bladespecial" ?

Auto-items

Every item type has a setting: "Auto-style". This can be either "none", "cheap", "normal", "better", "best", "superior".
If you add a "cheap random" item to a shelf (monster, …), then the engine replaces it with a random item of the item types which has "cheap" as auto-style.
Quest items should be signed as "none". It wouldn't be good if the player finds the grail in a barrel. On the streets of a small village. Other attributes of the item can be signed--f.e., if you don't want the item to be sold.

Item quality

The item classes/subclasses may need or don't need the "quality" setting.
The quality can be between 0 and 65535. An armor with 65535 as quality may never need repair at all – however its state modifies the final values (damage, ac, …). A knife with 1000 as quality may need regular repair.

Class/subclass	***Meaning***
Weapon/all subclasses	Every weapon loses quality as you fight with them – so the quality defines how often you need to repair them. An ancient weapon may not need a repair at all, because of extraordinary high quality.
Heavy armor/Light armor, all subclasses	Every armor loses quality as it decreases the damage the monsters try to apply on the player. Also, an ancient armor may not need repair.
Accessory/all subclasses	Don't need the quality – the amulets and rings never lose quality.
Book/all subclasses	Don't need the quality
Alchemy/all subclasses	A mortar/pestle loses quality as the player uses it to make new potions. The quality loss depends on the player's skill.
Ingredient	Don't need the quality
Potion	Don't need the quality
Magic	Don't need the quality
Key/lockpick	A lockpick loses quality as the player uses it to pick locks. The quality loss depends on the player's skill.
Repair/hammer	A repair hammer loses quality as the player uses it to repair items. The quality loss depends on the player's skill.
Misc	Don't need the quality

Comestible	The quality defines the overall state of the food/water. If it's too low, the food could be poisonous.
Gem	Don't need the quality
Card	Don't need the quality

Outfit Quality

Every weapon, armor, accessory has an outfit quality data (0-10). The final outfit value will be calculated by the system. The most important component is the armor (weighted), I mean its outfit quality.

Required alignment

Every item can specify the required alignment – which means the fame. If the item specifies "evil" as required alignment, then the player with positive fame can't equip it.

Monster architecture

Like everything else in The Quest, monster types have IDs. Every single monster belongs to a Monster type, and every Monster Type belongs to a Base Monster Type.

Monster families

Monsters don't have such "race" info as the player does. Both Base Monster Types and Monster Types have a "family" info (of course the data in the Base Monster Types is just a hint – its children, the Monster Types could set another). The family can be one of the following: ***humanoid, creature, undead, citynpc***.
This is important – since certain weapons can be enchanted to do extra damage against one of these families.F.e. "The executioner" sword which does +50% damage against humanoids.

Base Monster Types

The Base Monster Types have the bitmap info, and hints for its children, the Monster Types. The hints cover the damage, ac, attributes, and so on. When you make a new Monster Type, its data is filled from the parent Base Monster Type. Then you can change anything.
Important: the data in the Base Monster Types means *per level*, f.e. a Dragon usually has 10 hp/level (just an example). So the base hp for your Dragon Monster Type (level 10) will be 100, but you can modify this easily.

Monster Types

The Monster Types mean everything in the game about monsters. Their properties, items, spells, resistance, hp, damage, and so on. Every Monster Type contains the *Level* information – so if you need a Level 10 Dragon and a Level 11 one, those are separate Monster Types!
Every Monster Type have exactly 1 parent, a Base Monster Type.

Monsters

Every Monster has exactly 1 parent, a Monster Type.

Monsters have limited own data – that's why there could be hundreds of them around the world, still takes very low amount of memory. They have Hp information (while the Max Hp comes from the Monster Type), and the engine keeps a database about Monster diseases and other modifiers globally – since (normally) just a few of them have any diseases.
They have a have flags, btw:
- Aggressive: the Monster Type can have the Aggressive flag set, while the actual monster can be peaceful (and you can change it from script).
- Visible: They're all visible initially, you can also change it from script.

Placing monsters on the map

You need to select a Monster Type, then put it on the map. That's all.

Scripting monsters

Monsters can have more scripts…
1. One defined for their "Object", like in Legacy, like for Map Objects.
2. One defined for their "Monster Type".

The order is the following:

1. If the monster has an Object and it has a script, then the engine will run that if an event occurs.
2. If the monster doesn't have an Object, ***or*** the Object doesn't have script ***or*** the Object script calls the ***Default()*** function, then the engine runs the script defined for the Monster Type.

Creating, re-creating, removing monsters/monster types

There are certain script commands to create, re-create and remove monsters:

1. **RemoveMonsterTypes *(monstertypeid)***
   This will remove the monsters (which belong to the specified Monster Type) from the *current map.*
2. **CreateMonsterObj *(monstertypeid,mapobjid)***
   The will create a monster (with a Monster Type specified by *monstertypeid*), and place it at the map object (specified by *mapobjid*).
3. **CreateMonsterPos (*monstertypeid,x,y*)**
   This will create a monster (with a Monster Type specified by *monstertypeid*), and place it at x,y position of the *current map*.
4. **CreateMonsterDir (*monstertypeid,dir,distance*)**
   This will create a monster (with a Monster Type specified by *monstertypeid*), and place it at *dir* of the player, at *distance*. F.e. CreateMonsterDir("base_mymonster","north",5) – this will create the monster 5 squares away from the player, to north.

Monster -> NPC

The Monster Type needs to specify whether it's an NPC or not.
So when you create a monster, then you may create an NPC as well – it depends on the NPC setting of the Monster Type.

Monster's resistance/spell knowledge/other modifiers

Every monster type can have unlimited number of modifiers (called enchantment):

Enchant type	***Enchant effect***	***Data range***

Constant	Fortify magic	-100%..+100%
	Fortify melee	-100%..+100%
	Fortify spell	A spell, -100%..+100% (fortify a specific spell – if the monster casts it)
	Resist diseases	-100%..+100%
	Resist magic	-100%..+100%
	Resist normal weapons	-100%..+100% (normally undeads should resist normal weapons – this means a weapon w/o an enchantment)
	Resist paralysis	-100%..+100%
	Resist poison	-100%..+100%
	Resist spell	A spell, -100%..+100% (resist a specific spell)
Spell knowledge	A spell	10%..100% (When the monster would like to cast a spell, the system picks one from the list). The sum of the % values doesn't have to be 100%.
When attack	Curse	10..100% (during melee attack, how often will the attack result in cursing)
	Drain hp	10..100%
	Drain sp	10..100%
	Infect disease	A disease, 10..100%
	Level drops	10..100%
	Paralyze	10..100%
	Poison	10..100%

Script architecture

Certain objects can have scripts in the system:

Place	***description***
Map object	Similar to the object script in Legacy. The difference is, one script can be used more times, at different places. Available events: *stepon, stepoff, use, hit, spell* (yes, now map objects can check what weapon the player has used to hit them, or what spell has been cast).
NPC	The NPCs can have scripts. Also can be used by more NPCs. Such script could calculate complex values, set a global, the NPC's dialog conditions can check the global. Available events: *use (as a talk)*, *die*.
Monster	Similar to an NPC's script. Could calculate complex values. Available events: *use (as a talk), hit, spell*, *die*.
Dialogue/response	Now each response can have own script – it cannot be used more times. These are usually small scripts, the logic of the response (add/complete quest, set player's values, and so on). *NO events!* The right script is executed when its parent response is being displayed.
Item	The item types can have scripts. So something could happen if the player wears/wields/removes an item. Available events: *wear, wield, remove*, *hit*.
Spell	It could be used to create complex spells. Available events: *spell*.

Now the script system uses compression (compilation), so these scripts are much smaller then the ones in Legacy.

IMPORTANT
Every script in the system first needs to check the actual event !!!
F.e.

```
if(IsEventUse())
{
```

```
  Message("You can't use this");
}
```

Script commands

There are several script command families, explained below.

Global commands

These commands modify the global system or requests its properties.

Command	***Meaning***	***Example***
GetGlobal (*globalid*)	Read the specified global's value (can be either string or number)	Var=GetGlobal("base_myglobal");
SetGlobal (*globalid,value*)	Sets the specified global's value (can be either string or number)	SetGlobal("base_myglobal", "myvalue");
IsDayLight ()	Returns whether the time is between 6 and 21 (inclusive)	If(IsDayLight()) Message("daylight");
IsNight ()	Returns whether the time is between 22 and 5 (inclusive)	If(IsNight()) Message("night");
GetTime ()	Returns the current time (the elapsed game minutes since the game start)	Time=GetTime();
GetDay ()	Returns the current day no	Day=GetDay();
GetHour ()	Returns the current hour of the day	Hour=GetHour();
GetMinute ()	Returns the current minute of the hour	Minute=GetMinute();
GetDayDiff (*time*)	Returns the difference between the current time and the specified time (in days)	Daydiff=GetDayDiff(GetGlobal("base_savedtime"));
GetHourDiff (*time*)	Returns the difference between the current time and the specified time (in hours)	Hourdiff=GetHourDiff(GetGlobal(base_savedtime"));
GetMinuteDiff (*time*)	Returns the difference between the current time and the specified time (in minutes)	Minutediff=GetMinuteDiff(GetGlobal("base_savedtime"));
GetDamage ()	Returns the actual damage (can be used in "hit" or in "spell" events)	If(GetDamage()>10) SetDamage(10);
SetDamage (*damage*)	Sets the actual damage (can be used in "hit" or in "spell" events)	If(GetDamage()>10) SetDamage(10);
IsSpell (*spelltypeid*)	Returns whether the actual spell is the specified one (can be used in "spell" events)	If(IsSpell("base_spburninghand")) SetDamage(0);
IsWeapon (*itemtypeid*)	Returns whether the actual weapon is the specified one (can be used in "hit" events)	If(IsWeapon("base_mysword")) SetDamage(GetDamage()*2);
ItemValue (*itemtypeid*)	Returns the value of the specified item type	If(ItemValue("base_mysword")>10000) Message("It's really expensive!");
IsCurrentMap (*mapid*)	Whether the actual map is the specified one	If(IsCurrentMap("base_mymap1")) Message("Welcome to my map!");
IsCurrentWorld (*worldid*)	Whether the actual world is the specified one	If(IsCurrentWorld ("xxxx")) Message("I shouldn't be here…");
CreateObjectList (*title*)	Creates a selection list	CreateObjectList ("Which item do you want?");
AddObjectToList (*id,title*)	Add an object to the list	AddObjectToList ("base_mysword", "The sword");
ShowObjectList ()	Finalized the object selection list	ShowObjectList();

GetObjectListResult ()	Returns the selected item entry's ID. Empty if cancelled.	Item=GetObjectListResult();
		If(item!="")
		Player.ReceiveItem(item);
		else
		Message("Goodbye, then.");

Message (*message*)	Displays a message	Message("hello!");
Random (*max*)	Returns a random value between 0 and max-1 (inclusive)	If(Random(50)>40) Message("Hey!");
TalkNPC (*npcid*)	Initiate a conversion with an NPC	TalkNPC("base_mynpc");

SetMapFog (*mapid, viewdistance, fogcolor*);	Sets a dungeon map's fog and relative view distance. Viewdistance: 0-100 (inclusive). Fogcolor: f.e. "FF0000", hex rgb color (first two digits – red, then green and blue).	SetMapFog("base_mymap",70,"FFFF00");
ResetMapFog (*mapid*)	Resets the specified map's relative view distance and fog color.	ResetMapFog("base_mymap");
GetWeatherCloud ()	Returns the actual cloud (0..100, inclusive)	Cloud=GetWeatherCloud();
SetWeatherCloud (*skyid,cloud*)	Sets the actual cloud. The sky id can be empty for normal skies and a valid skytype id for spec ones.	SetWeatherCloud("base_mysky",0);
GetWeatherRain ()	Returns the actual rain.	Rain=GetWeatherRain();
SetWeatherRain (*rain*)	Sets the actual rain.	SetWeatherRain(100);
RemoveMonsterTypes (*monstertypeid*)	Remove the monsters of the specified monster types from the actual map.	RemoveMonsterTypes("base_mymonster");
CreateMonsterObj (*monstertypeid, mapobjid*)	Creates a monster of the specified monster type and places it at the specified map object.	CreateMonsterObj ("base_mymonster", "base_monspos");
CreateMonsterPos (*monstertypeid, x, y*);	Creates a monster of the specified monster type and places at at x,y of the current map. The x and y is between 0-34 for dungeon maps and 0-20 for surface maps.	CreateMonsterPos ("base_mymonster", 10,10);
CreateMonsterDir (*monstertypeid, dir, distance*);	Creates a monster of the specified monster type and places it at [dir] of the player, at [distance] distance. The dir can be "north", "sound","west" or "east".	CreateMonsterPos ("base_mymonster", "north",2);
PlaySound (*soundid*)	Plays a sound	PlaySound ("base_hitmiss");
PlayMusic (*musicid*)	Plays a music	PlayMusic ("base_surfaceday");
AttractMonsters (*distance*)	Attracts nearby monsters. Monsters with objects cannot be attracted by this command.	AttractMonsters (3);
CalmMonsters (*distance*)	Calms nearby monsters down. Monsters with objects cannot be calmed down by this command.	CalmMonsters (3);
PlayScene (*sceneid*)	Plays a scene (intro…)	PlayScene ("base_intro");
PlayCard (*cardskill*)	Initiates a card play. The opponent's card skill can be defined between 1 and 100 (inclusive). Not supported.	PlayCard (100);

EvtStepOn ()	Whether the actual event is "stepon"	If(EvtStepOn())
		{
		…
		}

EvtStepOff ()	Whether the actual event is "stepoff"	If(EvtStepOff())
		{
		…
		}

EvtItemChange ()	Whether the actual event is "itemchange"	If(EvtItemChange())
		{
		…
		}

EvtHit ()	Whether the actual event is "hit"	If(EvtHit ())
		{
		…
		}

EvtSpell ()	Whether the actual event is "spell"	If(EvtSpell ())
		{
		…
		}

EvtUse ()	Whether the actual event is "use"	If(EvtUse ())
		{
		…
		}

EvtDie ()	Whether the actual event is "die"	If(EvtDie ())

		{
		…
		}

EvtWear ()	Whether the actual event is “wear”. Not supported.	If(EvtWear())
		{
		…
		}

EvtWield ()	Whether the actual event is “wield”. Not supported.	If(EvtWield())
		{
		…
		}

EvtRemove ()	Whether the actual event is “remove”. Not supported.	If(EvtRemove())
		{
		…
		}

IsEvent (*eventname*)	Whether the actual event is the specified one.	If(IsEvent(“use”))
		{
		…
		}

Default ()	Runs the specified implementation of the current object.	Default();
QuestionYesNo (*question*)	Displays a question, with Yes and No buttons.	QuestionYesNo(“Do you want to touch it?”);
QuestionOkCancel (*question*)	Displays a question with Ok and Cancel buttons.	QuestionOkCancel(“You enter the tomb.”);
Answer ()	Returns the answer for the Question. 1: Yes/Ok, 0: No/Cancel	If(Answer()==1) Message(“Then let it be.”);
Goodbye ()	Terminates the dialogue conversation (has no effect in normal scripts!)	Goodbye();

Player commands

Command	***Meaning***	***Example***
AddQuest (*questid*)	Adds a quest to the player	Player.AddQuest(“base_myquest”);
SolveQuest (*questid*)	Sets the quest as solved.	Player.SolveQuest(“base_myquest”);
FailQuest (*questid*)	Sets the quest as failed.	Player.FailQuest(“base_myquest”);
HasQuest (*questid*)	Whether the player has received the quest.	If(Player.HasQuest(“base_myquest”)) Message(“You already have your quest.”);
IsQuestFailed (*questid*)	Whether the quest has failed.	If(Player.IsQuestFailed(“base_myquest”)) Message(“That quest has failed.”);
IsQuestSolved (*questid*)	Whether the quest has been solved.	If(Player.IsQuestSolved(“base_myquest”)) Message(“That quest has been solved.”);
RemoveQuest (*questid*)	Removes the quest from the player.	Player.RemoveQuest(“base_myquest”);
FindItem (*itemtypeid,amount*)	The player finds an item (or more items).	Player.FindItem(“base_mysword”,2);
ReceiveItem (*itemtypeid,amount*)	The player received an item (or more items).	Player.ReceiveItem(“base_mysword”,2);
RemoveItem (*itemtypeid,amount*)	The player loses an item (or more items).	Player.RemoveItem(“base_mysword”,2);
HasItem (*itemtypeid*)	Whether the player has the specified amount of items of the specified item type	If(Player.HasItem(“base_mysword”,2)) Message(“Yeah you have two of them.”);
AddXp (*value*)	The player receives xp (the value cannot be negative).	Player.AddXp(100);
RemoveXp (*value,canberestored*)	The player loses xp (the value cannot be negative). The “canberestored” parameter indicates whether the lost xp can be restored later.	Player.RemoveXp(100,1);
RestoreSavedXp ()	Restored the saved xp.	Player.RestoreSavedXp();
HasSavedXp ()	Whether the player has saved xp.	If(Player.HasSavedXp()) Message(“You could restore the lost xp…”);
FindGold (*amount*)	The player finds gold.	Player.FindGold (10);
ReceiveGold (*amount*)	The player receives gold.	Player.ReceiveGold (10);
RemoveGold (*amount*)	The player loses some gold.	Player.RemoveGold (10);

GetGold ()	Returns the player's gold.	Message("You have " + Player.GetGold() + " gold.");
GetSkillValue (*skillid*)	Returns the specified skill's value.	Skill=Player.GetSkillValue("Perception");
AddSkillValue (*skillid,value*)	Adds the value to the specified skill's value of the player.	Player.AddSkillValue("Perception",1);
AddSpell (*spelltypeid*)	The player learns a spell.	Player.AddSpell("base_myspell");
RemoveSpell (*spelltypeid*)	The player forgets a spell.	Player.RemoveSpell("base_myspell");
KnowsSpell (*spelltypeid*)	Returns whether the player knows the specified spell.	If(Player.KnowsSpell("base_myspell")) Message("You know what spell already.");
AddAbility (*abilitytypeid*)	The player receives an ability.	Player.AddAbility("base_myability");
RemoveAbility (*abilitytypeid*)	The player loses an ability.	Player.RemoveAbility("base_myability");
HasAbility (*abilitytypeid*)	Whether the player has the specified ability.	If(Player.HasAbility("base_myability")) Message("You already have that ability.");
GetAttributeValue (*attributename*)	Returns the specified attribute's value of the player.	Str=Player.GetAttributeValue("strength");
AddAttributeValue (*attributename*)	Adds the value to the specified attribute's value of the player.	Player.AddAttributeValue("strength",1);
Move (*mapobjid*)	Moves the player to the specified object's position.	Player.Move("base_targetobj");
MoveMode (*mapobjid,movemode*)	Moves the player to the specified object's position, using the specified method. The method can be one of the following: "" (empty string), "teleport", "fall", "ladderdown", "ladderup", "stairdown", "stairup"	Player,MoveMode("base_targetobj","teleport");
MoveQuestion (*mapobjid,question*)	Asks the player whether it wants to move the specified object's position.	Player.MoveQuestion("base_targetobj", "Do you want to go there?");
Turn (*direction*)	Turns the player to the specified direction.	Player.Turn("north");
MovePos (*x,y*)	Moves the player to the specified x,y position.	Player.MovePos(5,5);
GetPosX ()	Returns the player's X coordinate.	X=Player.GetPosX();
GetPosY ()	Returns the player's Y coordinate.	Y=Player.GetPosY();
InfectDisease (*diseaseid*)	Infects the player with the specified disease.	Player.InfectDisease("base_mydisease");
CureDisease (*diseaseid*)	Cures the player of the specified disease.	Player.CureDisease("base_mydisease");
IsDiseased (*diseaseid*)	Returns whether the player has been infected by the specified disease (can be empty – for checking any diseases).	If(Player.IsDiseased("base_mydisease")) Message("You are sick!");
GetRace ()	Returns the player's race.	Race=Player.GetRace();
GetHp ()	Returns the player's hp.	Hp=Player.GetHp();
SetHp (*value*)	Sets the player's hp to the specified value.	Player.SetHp(20);
AddHp (*value*)	Adds the value to the player's hp.	Player.AddHp(20);
SetFullHp ()	Restores the full health of the player.	Player.SetFullHp();
GetMaxHp ()	Returns the max hp of the player.	Maxhp=Player.GetMaxHp();
GetSp ()	Returns the player's sp.	Sp=Player.GetSp();
SetSp (*value*)	Sets the player's sp to the specified value.	Player.SetSp(20);
AddSp (*value*)	Adds the value to the player's sp.	Player.AddSp(20);
SetFullSp ()	Restores the full mana of the player.	Player.SetFullSp();
GetMaxSp ()	Returns the max sp of the player.	Maxsp=Player.GetMaxSp();
IsGood ()	Returns whether the player's fame is above 0.	If(Player.IsGood()) Message("You're a good person.");
IsEvil ()	Returns whether the player's fame is below 0.	If(Player.IsEvil()) Message("You're a terrible person.");
IsUndead ()	Returns whether the player's race is Rasvim.	If(Player.IsUndead()) Message("Get away, beast!");
ExecuteSpell (*spelltypeid,skill*)	Executes a spell at the player.	Player.ExecuteSpell("base_myspell");

GetFame ()	Returns the player's fame.	Fame=Player.GetFame();
AddFame (*value*)	Adds the specified value to the player's fame (can be negative).	Player.AddFame(1);
GetCrime ()	Returns the player's crime.	Crime=Player.GetCrime();
ClearCrime ()	Clears the player's crime.	Player.ClearCrime();
AddCrime (value)	Adds the value to the player's crime (can be negative)	Player.AddCrime (2);
GetOutfit ()	Returns the player's outfit.	Outfit=Player.GetOutfit();
GetLevel ()	Returns the player's level.	Level=Player.GetLevel();
GetAc ()	Returns the player's ac.	Ac=Player.GetAc();
MoveToJail ()	Moves the player to the nearest jail. RESERVED function – shouldn't be used except by city guards.	Player.MoveToJail();
MoveBack ()	Moves the player back to the previous position.	Player.MoveBack();
ClearJail ()	Clears the player's jail state. RESERVED function – shouldn't be used except by city guards.	Player.ClearJail();
IsInJail ()	Returns the player's jail state. RESERVED function – shouldn't be used except by city guards.	Isinjail=Player.IsInJail();
IsParalyzed ()	Returns whether the player is paralyzed.	If(Player.IsParalyzed()) Message("You're paralyzed.");
IsPosioned ()	Returns whether the player is poisoned.	If(Player.IsPoisoned()) Message("You're poisoned.");
IsCursed ()	Returns whether the player is cursed.	If(Player.IsCursed()) Message("You're cursed.");
IsMale()	Returns whether the player is male.	If(Player.IsMale()) Message("Hey, pal, wazzup?");

Map object commands

Command	***Meaning***	***Example***
IsOpened ()	Returns whether the dungeon door is opened.	If(base_mydoor.IsOpened())
Open ()	Opens the dungeon door.	Base_mydoor.Open();
Close ()	Closes the dungeon door.	Base_mydoor.Close();
GetLockedState ()	Returns the locked state of the object (dungeon door/city door/city window). Can be 0-101. 101 means magical (cannot be opened by lockpick/spell).	If(Base_mydoor.GetLockedState()==0)
SetLockedState (*value*)	Sets the locked state of the object (dungeon door/city door/city window). Can be 0-101. 101 means magical (cannot be opened by lockpick/spell).	Base_mydoor.SetLockedState(10);
GetTrappedState ()	Returns the trapped state of the object (dungeon door/ container/ lever/ shelf/ secret switch/). Can be 0-101. 101 means magical (cannot be disarmed by skill/spell).	If(Base_mydoor.GetTrappedState()==0)
SetTrappedState (*value*)	Sets the trapped state of the object ((dungeon door/ container/ lever/ shelf/ secret switch/). Can be 0-101. 101 means magical (cannot be disarmed by skill/spell).	Base_mydoor.SetTrappedState(10);
IsVisible ()	Returns whether the object is visible	If(Base_mydoor.IsVisible())
SetVisible ()	Sets the object's visibility.	Base_mydoor.SetVisible();
SetHidden ()	Sets the object's visibility.	Base_mydoor.SetHidden();
IsOn ()	Returns whether the object's on/	If(Base_mytorch.IsOn())

	off state is “on” (could mean various things – check the object’s property page in the editor)	
SetOn ()	Sets the object’s on/off state.	Base_mytorch.SetOn();
SetOff ()	Sets the object’s on/off state.	Base_mytorch.SetOff();
GetState ()	Returns the object’s state (0..255).	State=Base_myObj.GetState();
SetState (*state*)	Sets the object’s state (0..255).	Base_myObj.SetState(1);
ReceiveItem (*itemtypeid,amount*)	The shelf receives an item (or more items).	Base_myShelf.ReceiveItem(“base_mysword”, 1);
RemoveItem (*itemtypeid,amount*)	The shelf loses an item (or more items).	Base_myShelf.RemoveItem(“base_mysword”, 1);
HasItem (*itemtypeid,amount*)	Returns whether the shelf has the specified amount of the specified item.	If(Base_myShelf.HasItem(“base_mysword”,2))

Monster command

Command	***Meaning***	***Example***
IsVisible ()	Returns whether the monster is visible	Visible=IsVisible();
SetVisible ()	Sets the monster’s visibility	SetVisible();
SetHidden ()	Sets the monster’s visibility	SetHidden();
IsDead ()	Returns whether the monster is dead	Dead=IsDead();
FindItem (*itemtypeid,amount*)	The monster finds an item (or more items).	FindItem(“base_mysword”,1);
ReceiveItem (*itemtypeid,amount*)	The monster receives an item (or more items).	ReceiveItem(“base_mysword”,1);
RemoveItem (*itemtypeid,amount*)	The monster loses an item (or more items);	RemoveItem(“base_mysword”,1);
HasItem (*itemtypeid,amount*)	Returns whether the monster has the specified item (specified amount of)	If(HasItem(“base_mysword”,1))
StealGold ()	The monster steals gold from the player. Not supported.	StealGold();
KnowsSpell (*spelltypeid*)	Returns whether the monster knows the specified spell.	If(KnowsSpell(“base_myspell”))
Move (*mapobjid*)	Moves the monster to the specified object’s position.	Move(“base_pos”);
Turn (*direction*)	Turns the monster to the specified direction (works only for streetwalkers).	Turn(“north”);
MovePos (*x,y*)	Moves the player to the specified x,y position of the current map.	MovePos(5,5);
GetPosX ()	Returns the X coordinate of the monster (current map).	X=GetPosX();
GetPosY ()	Returns the Y coordinate of the monster (current map).	Y=GetPosY();
InfectDisease (*diseaseid*)	Infects the monster by the specified disease.	InfectDisease(“base_disease”);
CureDisease (*diseaseid*)	Cures the monster of the specified disease.	CureDisease(“base_disease”);
IsDiseased (*diseaseid*)	Returns whether the monster is infected by the specified disease (can be empty for checking any diseases)	If(IsDiseased(“base_disease”))
GetHp ()	Returns the monster’s hp.	Hp=GetHp();
SetHp (*value*)	Sets the monster’s hp to the specified value.	SetHp(20);
AddHp (*value*)	Adds the value to the monster’s hp.	AddHp(20);
SetFullHp ()	Restores the full health of the monster.	SetFullHp();
GetMaxHp ()	Returns the max hp of the	Maxhp=GetMaxHp();

	monster.	
IsUndead ()	Returns whether the monster's race is "Rasvim"	Undead=IsUndead();
ExecuteSpell (*spelltypeid,skill*)	Execute the specified spell at the monster (with a specified skill value)	ExecuteSpell ("base_myspell",20);
GetLevel ()	Returns the monster's level.	Level=GetLevel();
GetAc ()	Returns the monster's ac.	Ac=GetAc();
IsAggressive ()	Returns whether the monster is aggressive.	Aggr=IsAggressive();
SetAggressive ()	Sets the monster as aggressive.	SetAggressive();
SetPeaceful ()	Sets the monster as peaceful.	SetPeaceful();
IsParalyzed ()	Returns whether the monster is paralyzed.	Para=IsParalyzed();
IsPoisoned ()	Returns whether the monster is poisoned.	Poisoned=IsPoisoned();
IsCursed ()	Returns whether the monster is cursed.	Cursed=IsCursed();

Item commands

There are no specific Item commands – so in item scripts, you could use only the other family commands. Note: item scripts are not supported.