Ustoria Script

# Introduction

This is a scripting language designed to be used as part of the Ustoria editor and player. These are high-level tools to allow non-technical people to create stories which can be developed into textual presentations, graphic novels, interactive movies and video games. Ustoria Script will be a way for the author of the stories to specify how their characters respond to events that happen to them at different times. The goal of the scripting language is to be declarative whenever possible and only resort to procedural techniques when declarative techniques do meet the requirements.

# Ustoria Script Language

## Scope

Ustoria Script provides 3 scope levels:

1. Global Scope is where you define things which can be accessed throughout the story.
2. Scene Scope is where you define things which can be accessed within a scene of the story for the lifetime of that scene. Normally, a scene exists from the time when it is created until the time when it is destroyed.
3. Advisor Scope is where you specify one or more actions associated with a concern of a character. You can define objects which are local to this scope and can only be accessed from within this scope.

Objects in any scope can access objects in the same scope or a higher scope.

Objects are usually named and these names must follow the following rules:

* names must start with an upper or lower case letter or "\_".
* the remaining characters in a name can be numbers, letters, or underscores.
* Any variable can have its scope specified by using a fully qualified name where the name of the object is prefixed by the name of the scope and the components of the name are separated by "." characters. For example, to access a variable named "x" in the global scope can be accessed as global.x. You can specify the value in any scope using the notation <scopeName>.objectName. You can also use the keyword "super" to specify the scope immediately above you.
* In general, object names should be unique. Within a single scope, object names must be unique. A lower-level scope can re-define an object with the same name in a higher scope. When this is done, the author must be notified of this to ensure that this was done on purpose and not accidentally. The object with the same name in a higher scope can be disambiguated by qualifying the name with the name of the higher scope.
* An object cannot be the same as one of the reserved words in the scripting language.

## Variables

Variables store the value of something. Variables are named and can be constant so that their values cannot be modified after they are initially assigned.

The following variable types are supported:

* number: an integer or floating point number
* string: a character string that has the capacity for 4 GB of data.
* boolean: true or false
* ValueSet: A set of values as defined by the author.

Variables can be declared to be of a specific type or the type can be declared contextually. To declare a variable contextually, you simply assign it a value and the type of the value assigned determines the type of the variable. Once the type of variable has been assigned, it cannot be changed for the lifetime of the variable. Variables are created in one of the three scopes and their lifetime and visibility is determined by the scope in which they reside.

### Value Sets

A value set is a user-defined variable type. It consists of a set of variable values, each of which has a name associated with it. The values can be distinct so that their value is different from every other value in the set or they can be fuzzy so that some of their values overlap some of the values in an adjacent set. To better understand this, let's look at some examples.

All value sets draw their potential values from a subset of the numbers. Consider the case where we want to represent the animation currently being played by a character. The character could be idle, walking, jumping, or running. We could represent this as:

idle

walking

running

jumping

0

1

2

3

In this case, each of the values has been associated with a number and each of the values is distinct. This works well in the situation where we can only be playing one animation at a time. To create a value set like this you need to specify:

* the names of the values that can be assigned (idle, walking, running, jumping)
* the fact that these values should be distinct

In another case, you might want to specify the decorations displayed on a character in your game. You might show a halo around the character when there is a power-up in action, an optional health bar over the character's head, and a circle on the ground around the character if it has been selected by the player. It is possible that any combination of these decorations could be displayed on a character so the variable has to store any combination of the different decorations, including none at all. This represents a set of values that are stored in the variable and any combination of the individual values can be stored. To create a value set like this you need to specify:

* the names of the values which can be stored (halo, health\_bar, selected)
* the fact that this is a set of values

Another type of value set represents a series of values which is not exact. Consider the health of your character. It could be very healthy, healthy, slightly sick, sick or very sick. None of these values is exact and, in fact, the values overlap one another. The transition from sick to very sick does not happen instantly, but gradually. We can represent this graphically, as shown below.

very\_sick

sick

slightly\_sick

healthy

very\_healthy

Each of these values actually encompasses a range of the number line, not just a single value. The transition from very sick to sick is gradual, not abrupt. There is an overlap between the two, when you are more one than the other. This lets you use inexact terms and have the story act in an appropriate manner.