

Torque Game Builder – TorqueScript Overview

(Adapted from Edward Maurina's book, "The Game Programmer's Guide to Torque" [GPGT]. Grab the entire book to take your Torque knowledge to the next level.)

Introduction

This document offers an introduction to TorqueScript. You will not be a TorqueScript master after simply reading this - that takes time and practice. But you will walk away with a good feel for how TorqueScript is structured and how it works. With this knowledge in hand, you will be ready to take off and start learning how to best use the tools TorqueScript gives you to create your game.

What is Scripting?

Scripting languages are programming languages designed to enable scripting. Scripting is the act of using preexisting (engine) components to accomplish new tasks. In other words, we use a scripting language to access features in the engine and then use those features to provide a game experience.

Generally, scripting languages are interpreted, not compiled (like C++ and other languages). This makes scripting tasks somewhat slower than compiled tasks, but we make this trade in order to gain flexibility and visibility, as well as ease of use. Because scripting languages also allow you to modify your program without having to recompile it, we are able to rapidly prototype and repair code. This speeds development significantly.

Often, scripting languages allow you to write code without worrying about nitty-gritty programming details like data types or memory management. This is both a boon and a bane. It is a boon in that it simplifies many programming tasks, but it is also a bane because it allows us to make mistakes that a strict compiled language and its compiler would find. TGB will catch and notify you of syntax errors, but it does not check for type mismatches or variable naming mistakes.

TorqueScript

TorqueScript is a very powerful, flexible language with syntax similar to C++. The following sections will describe all of its features and provide examples to demonstrate its usage.

Torque Game Builder – TorqueScript Overview

Language Features

- **Type-insensitive** - In TorqueScript, variable types are converted as necessary and can be used interchangeably. TorqueScript provides several basic literal types, which are described later in this chapter.
- **Case-insensitive** - TorqueScript ignores case when interpreting variable and function names.
- **Statement termination** - Statements are terminated with a semi-colon (;) as in many modern programming languages (C++, Java, Javascript, etc).
- **Full complement of operators** - The complete list of TorqueScript operators is listed in the TorqueScript Quick Reference. TorqueScript provides all the basic operators common to most programming languages, along with a few more advanced operators.
- **Full complement of control structures** - As with any robust language, TorqueScript provides the standard programming constructs: if-then-else, for, while, and switch.
- **Functions**- TorqueScript provides the ability to create functions with the optional ability to return values. Parameters are passed by value and by reference.
- **Inheritance and Polymorphism** - TorqueScript allows you to inherit from engine objects and to subsequently extend or override object methods.
- **Namespaces** - Like C++, TorqueScript supports the concept of namespaces. Namespaces are used to localize names and identifiers to avoid collisions. Huh? This means, for example, that you can have two different functions named dolt() that exist in two separate namespaces, but which are used in the same code.
- **Compiles and executes PCODE** - As a bit of icing on the cake, the TorqueScript engine compiles scripts prior to executing them, giving a speed increase as well as providing a point at which errors in scripts can be reasonably found and diagnosed.

Torque Game Builder – TorqueScript Overview

Variables

Variables come in two flavors in TorqueScript: local and global. Local variables are transient, meaning they are destroyed automatically when they go out of scope. And what is scope? "Scope" is a term used to refer to the block of code a variable is defined in. For example, if we have a function, and we declare a local variable inside of that function, the local variable will be destroyed as soon as the function is done processing. When this happens, we say the variable has "gone out of scope". So, local variables only exist in their local scope- the function or code-block they are defined in. A piece of code inside a different function would not be able to see the local variable. Global variables, on the other hand, are permanent and exist throughout the entire program they are defined in.

TorqueScript specifically marks local and global variables with special characters, so that they are easy to tell apart. The syntax is as follows:

```
%local_var = value;  
$global_var = value2;
```

In TorqueScript, variables do not need to be created before you use them. If a piece of code attempts to evaluate the value of a variable that was not previously created, TorqueScript will create the variable automatically.

Variable names may contain any number of alpha-numeric (a..z, A..Z, 0..9) characters, as well as the underscore (`_`) character. However, the first character in a variable's name cannot be a number. You may end variable names with a number, but if you do, you must be especially careful with array names. For further explanation, see the section on arrays.

Data Types

TorqueScript implicitly supports several variable data-types: numbers, strings, booleans, and arrays. Each type is detailed below.

Numbers:

```
123      (Integer)  
1.234    (floating point)  
1234e-3  (scientific notation)  
0xc001   (hexadecimal)
```

Torque Game Builder – TorqueScript Overview

Nothing mysterious here. TorqueScript handles your standard numeric types.

Strings:

```
"abcd"      (string)
'abcd'      (tagged string)
```

Standard strings, in double-quotes, behave as you would expect. This is what is used in most cases. Strings which appear in single quotes, 'abcd', are treated specially by TorqueScript. These strings are called "tagged strings", and they are special in that they contain string data, but also have a special numeric tag associated with them. Tagged strings are used for sending string data across a network. The value of a tagged string is only sent once, regardless of how many times you actually do the sending. On subsequent sends, only the tag value is sent. Tagged values must be de-tagged when printing.

Booleans:

Like most programming languages, TorqueScript also supports Booleans. Boolean numbers have only two values - true or false.

```
true       (1)
false      (0)
```

The constant "true" evaluates to the number 1 in TorqueScript, and the constant "false" evaluates to the number 0.

Arrays:

```
$MyArray[n]      (Single-dimension)
$MyMultiArray[m,n] (Multi-dimension)
$MyMultiArray[m_n] (Multi-dimension)
```

TorqueScript is extremely flexible with arrays. Many scripting languages are more flexible with arrays than compiled languages like C++, but TorqueScript is even more flexible than most scripting languages. All this flexibility can get confusing, and deciphering how TorqueScript handles arrays sometimes trips people up. Part of this confusion comes from the various ways in which array variables can be accessed. For instance, \$MyArray[1] and \$MyArray1 are identical variable names. Also, \$MyMultiArray[0,1], \$MyMultiArray[0_1] and \$MyMultiArray0_1 all reference the same variable.

Operators

A complete listing of TorqueScript's operators can be found in the TorqueScript Reference.

Torque Game Builder – TorqueScript Overview

Control Statements

TorqueScript supports all the common control statements.

- **if-else**

```
if (%val < 0)
{
    %abs = -%val;
}
else
{
    %abs = %val;
}
```

The above example will execute the statement inside the if block if the number stored by %val is less than 0. Otherwise, the statement inside of the else block will be executed.

- **switch**

```
switch(%val)
{
case 1:
    echo("The value is 1");

case 2:
    echo("The value is 2");

default:
    echo("The value is something else");
}
```

- **switch\$**

```
switch$(%val)
{
case "Hello":
    echo("The string is Hello");

case "World":
    echo("The string is World");

default:
    echo("The string is something else");
}
```

Torque Game Builder – TorqueScript Overview

The first switch statement (switch) works with numbers and the second (switch\$) works with strings. Other than that their functionality is identical. A single case statement will be executed depending on the value inside the parentheses. For example, if %val is “Hello”, the statements after 'case “Hello”:' will be executed.

- for

```
for(%i = 0; %i < 10; %i++)
{
    echo(%i @ "/" @ 10);
}
```

- while

```
%i = 0;
while (%i < 10)
{
    echo(%i @ "/" @ 10);
    %i++;
}
```

The statements inside of a loop are executed repeatedly until an exit condition is met. In the case of a for loop, the exit condition is the second statement in the declaration. So, in the example, the loop will continue to execute until the variable %i is greater than or equal to 10. The first statement is the initialization statement and is executed once before the loop is run for the first time. The last statement is executed after each iteration of the loop prior to the exit condition test.

The exit condition of the while loop is between the parentheses after the *while* keyword. The for loop and while loop above both do the exact same thing, though in most situations one of the constructs will work better than the other.

Functions

Basic functions in TorqueScript are defined as follows:

```
function myFunction(%arg1, %arg2)
{
    %result = %arg1 + %arg2;
    return %result;
}
```

The above example is a simple adding function. It receives two numbers as parameters, adds them together, and returns the result. Functions can have any number of parameters and may or may not return a value.

Torque Game Builder – TorqueScript Overview

Objects

In Torque, every item in the game world is an object, and all game world objects can be accessed via script. For example, `t2dSceneObject`, `t2dSceneGraph`, and `t2dStaticSprite` can all be accessed via script, though they are defined in C++. Objects are created with the 'new' keyword.

```
%object = new t2dSceneObject(MyObject)
{
    Position = "10 10";
    Size = "5 15";
    Rotation = 45;
    OtherField = 10;
};
```

In this example, we are creating a `t2dSceneObject`, naming it "MyObject", and storing a reference to the object in the variable `%object`. This variable can then be used to call various methods defined for the `t2dSceneObject` class. For example, the following will set the newly created object's position to the point (10, 20):

```
%object.setPosition(10, 20);
```

The object's name can also be used to reference the object. Assigning a name to an object is optional.

The first three statements inside the object definition assign values to fields that the engine has exposed for the object. These are the object's position, size, and rotation. So, the object will be created with position (10, 10) width 5, height 15, and rotation 45. There are several other fields exposed as well. To see them all check out the TGB Reference document. The last field is a dynamic field. Dynamic fields are defined only in script and are used to store values for a specific instance of an object.

Methods

In addition to supporting the creation of functions, TorqueScript allows you to define methods on object classes and namespaces.

```
function t2dSceneObject::move(%this, %amount)
{
    %position = %this.getPosition();
    %newPosition = t2dVectorAdd(%position, %amount);
    %this.setPosition(%newPosition);
    return %newPosition;
}
```

Torque Game Builder – TorqueScript Overview

```
}
```

Methods are very similar to functions. The difference is they are associated with a specific class type. The above example creates a method for `t2dSceneObjects`. Assuming `%object` is a `t2dSceneObject`, this method could be called like this:

```
%object.move("10 5");
```

That would move `%object` 10 units in the x direction and 5 units in the y direction. Inside the method definition, the `%this` variable is actually a reference to the object that called the method. So in this case, it is the same as `%object`. `%this` is always the first parameter in a method definition (though it doesn't necessarily have to be named `%this`), and is automatically passed to the method by the engine.

Namespaces

Namespaces are a slightly advanced topic and can be confusing, but they are extremely useful when fully understood. First, every object belongs to a namespace. By default, the namespace is the name of the object's class. So, a `t2dSceneObject` is in the namespace 't2dSceneObject'. Also, namespaces are inherited. A `t2dStaticSprite` object is in the namespace 't2dStaticSprite', and also in the namespace 't2dSceneObject' since `t2dStaticSprite` is derived from `t2dSceneObject`. In this way, methods can be defined for a base class and used by object's of a derived class. For instance, the `setPosition` method is defined for `t2dSceneObject`, but a `t2dStaticSprite` can use it as well.

Conclusion

This was an extremely quick overview of the various features of TorqueScript. If you are new to programming, don't expect to understand everything that was covered right away. Programming is a confusing topic that takes time and experience to grasp fully. Hopefully you were able to learn some of the concepts involved with scripting and what it can do for you. The next step is to start following tutorials and practicing on your own. Good luck!