

Hands on Coding Guide

For the Complete Beginner (Demo)

Making Indie Games

Sync Reloaded Edition

Programmer and Writer: Kevin O'Flaherty

Editor: Ryan Griffin

Table of Contents

4	Introduction	<hr/>
4	How to use this Guide	<hr/>
5	Frist Code	<hr/>
	Getting the Engine	
	Hello World	
6	Using Game Scripts	<hr/>
	Game Scripts	
	Making a Simple Method in a Game Script	
	Executing the Game Script	
8	Variables	<hr/>
	Introduction	
	Variable Types	
	Scope of Variables	
	Testing the Scope of Local Variables	
	Passing Variables to Methods	
	Testing Arguments	
	Returning Variables to Methods	
	Arrays	

13 If Statements

Introduction
Switch Statement

15 Scripting Names

Object IDs
Names

16 Classes

Introduction
Example
Explaining the Code
%this

18 Behaviors

Introduction
Behavior Template
Explaining %this vs. %this.owner
Coding your First Behavior from Sync Reloaded
Testing the Behavior

23 Mixing Classes and Behaviors

Introduction
Classes to Behavior
Getting %this.Alpha at the Class Level

24 Vectors and String Manipulation

Vectors
String Manipulation

Introduction:

The guide contains the prerequisite scripting knowledge for understanding the code in Sync Reloaded Guide. It also creates a good base knowledge for coding in torque script.

Please do not duplicate the guide for it will not allow Making Indie Game to create more educational documentation.

How to use this guide:

Beginners:

I recommend going through each step of this guide, learning scripting as you go. After you have a handle on coding read the Sync Reloaded Guide to see the practical uses of scripting.

Intermediate to Advance:

If you are familiar with scripting I would recommend looking at the last four chapters of this guide. Those chapters contain information that many coders are never introduced to.

First Code

Getting the Engine:

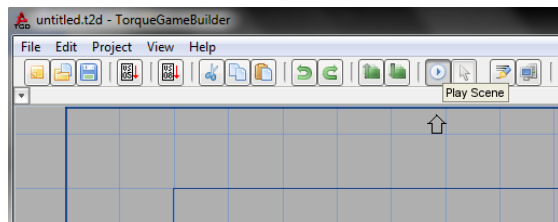
A game engine is a set of tools that a programmer uses to jumpstart the development process. The game engine contains code that controls things like the camera and rendering objects to the screen so the programmer doesn't have to worry about making everything from scratch.

The engine used in this guide is Torque Game Builder by Garage Games. You can find the engine and download a thirty day free trial at torquepowered.com. The reason this engine is used is because it allows a person to create a quality game with little knowledge of scripting. Now that you have the game engine, let's start coding.

Hello World:

In the hello world tradition it is a programmer's first task to display the words "hello world" on the screen. We will be completing this task with the console. The console is a way we can input text into the engine while the game is running.

Create a new project and name it what you wish. After creating the project you should see the TGB interface. In the menu bar, near the top of the screen, there should be a play scene button. Press this to run your game.

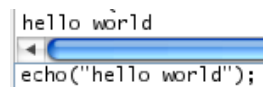


Open the console when the game is running. The console button is the ~ key in the top left of your keyboard and is located next to the number 1. You should now see the console GUI (Graphical User Interface). Enter the following code into it.

Console

```
echo("hello world");
```

Your results should be what you see to the right. The `echo()` method is used to print variables to the console. We will be using this method in later lessons.



Using Game Scripts

Game Scripts:

A game script is a source file of code the engine will run and is made prior to the execution of the game engine. To make a game script you need text editing software such as notepad. Any editor will work as long as it is able to save the file as “.cs”; the extension of script files in TGB.

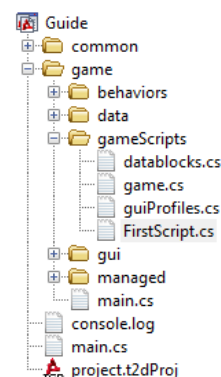
Making a Simple Method in a Game Script:

Open a new text file in your code editor and add the following script.

FirstScript.cs

```
function SayHello()
{
    //Prints to Console
    echo("Hello World");
}
```

Save your code file as “FirstScript.cs” and save it in the gameScripts folder located in the game directory. The default location of the game directory is in the “MyGames” folder under documents. You can see a hierarchy of the project to the right. In the last example we have two main points to discuss.



- **Methods**
 - A method is a collection of code that can be executed by calling it. A method in Torque Script starts with two word functions followed by the methods name and two () at the end. The method contains all the code between the { }.
 - The above method would be called using the code SayHello();
- **Comments**
 - A comment is a programmer’s way of making notes in the code. There are two types of comments: single line and multi-line comments.
 - Single line comments are made by “//” and makes the compiler skip all the text after it until the end of the line.
 - Multi-line comments are opened by “/*” and closed by “*/”. All text between them is left unexecuted.

Executing the Game Script:

When coding with scripts the computer needs to be told to run the scripts. To do this go to your main.cs file located in the game folder. Replace your initializeProject() method with the following.

Main.cs

```
function initializeProject()
{
    // Load up the in game gui.
    exec("~/gui/mainScreen.gui");

    // Exec game scripts.
    exec("./gameScripts/game.cs");
    exec("./gameScripts/FirstScript.cs");

    startGame( expandFilename($Game::DefaultScene) );
}
```

After adding the code and saving the two script files, run your game and open the console. Enter the following code that calls the SayHello() method.

Console

```
SayHello();
```

Your SayHello() method was called executing its code that printed “Hello World” to the screen. The output of the console is to the right.

```
==>SayHello();
Hello World
SayHello();|
```

Variables

Introduction:

Variables are the magic of programming, making programs dynamic. It allows a program to run one way the first time and a different way the second. Variables are simply an address that point to a block of memory.

Variable Types:

The block of memory can be a number or text. Which one it is depends on the variables type. Below is a list of types that are used in Torque Script.

Numbers

- Integer: A single non-decimal number. Example: 5
- Float: A number containing a decimal. Example: 5.05
- Bool: True or false (0 for false, 1 for true). Example: true

Text

- String: A collection of text between “”. Example “Hello”

Variables in Torque Script work differently than other languages in the fact that the engine handles typecasting. In order to understand how this works we will do some experimenting. Run your project and enter the console. Enter the following code into the console. You may need to type in each line one at a time.

Console

```
$variable = 1.00;  
echo($variable);  
$variable = 1;  
echo($variable);  
$variable = true;  
echo($variable);  
$variable = "true";
```

```
==>$variable = 1.00;echo($variable);  
1  
1  
1  
true
```

The code produced the results 1, 1, 1, true. The reason behind the three ones is because the engine typecasts the variables when they are entered into one single format. It then typecasts the numbers again if a Boolean (true value), or float is needed.

Scope of Variables:

There are two different variables that determine scope; how long the variable remains in memory. The first is global variables and the second is local variables.

Global Variables

- Global variables are called by \$.
- Global variables go out of scope at the end of the build.

Local Variables

- Local variables are called by %.
- Local variables go out of scope at the end of the function and are not affected by if statements. This is unlike other languages.

We used global variables in the last coding example and it allowed us to program outside a function. Since the scope of a global variable doesn't end until the program ends it will not be tested further.

Testing the Scope of Local Variables:

Open up the FirstScript.cs script file and add the following code to it.

FirstScript.cs

```
function VariableTest()
{
    %local = 5;
    echo(%local);
    VTest2(); //calls the VTest2() method
}

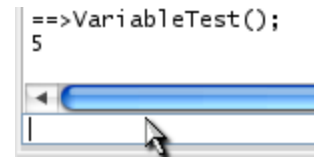
function VTest2()
{
    echo(%local);
}
```

Save the code file and open up TGB. Run the game and enter the console. Once in the console run your VariableTest() method by the following code.

Console

```
VariableTest();
```

The output from the console will look like the picture to the right. The first output, 5, from the first method, where the variable is still in scope. The second output is blank from the second method, where %local became out of scope.



```
==>VariableTest();
5
```

Passing Variables to Methods:

If the second method VTest2() needed the local variable we could pass it using arguments.

Arguments

- Can pass variables to other methods when calling them.
- To pass a variable to a method use the following format, Method(%variable).
- To receive a variable in a method use the following format, function Method(%variable).
- The variable names do not have to match.
- To send multiple variables to a method separate the variables by commas, Method(%variable1, %variable2).

Testing Arguments:

Open up the FirstScript.cs script file and replace the VariableTest() method and VTest2() with the following:

```
FirstScript.cs
function VariableTest()
{
    %local = 5;
    echo(%local);
    VTest2(%local); //calls the VTest2() method and sends the %local variable
}

function VTest2(%local2)
{
    echo(%local2);
}
```

Save the file and open up TGB. Run your project and run the VariableTest() method.

Console

```
VariableTest();
```

The output contains two fives. This means we have successfully passed the local variable to the second method. I would like you to note that the variable in the second method is %local2 instead of %local. This means you can change the names of the argument variables and it will not have any effect.

```
==>VariableTest();
5
5
```

Returning Variables to Methods:

Methods can also return variables to the called method using the return keyword. Using the FirstScript.cs replace the VariableTest() and VTest2() methods with the following code:

FirstScript.cs

```
function VariableTest()
{
    %local = 5;
    echo(%local);
    %local = VTest2();
    echo(%local);
}

function VTest2()
{
    %local = 0;
    return %local;
}
```

Save the file and run your project. Enter the console and call the VariableTest() method, the output can be found at the right. You can imagine the variable being returned replacing the method calling it; in this case VTest2().

```
==>VariableTest();
5
0
```

Arrays:

An array is a special version of a variable that allows unlimited elements to be stored under one name. A table showing the properties of arrays is below.

Arrays

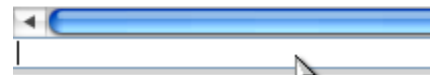
- Can be local or global, call them with \$ or % followed by the name and then [slot]. Example \$array[0].
- The slot starts at element zero.
- Scripting is very flexible and can be any type, (Boolean, Float, String).
- The size of the area also doesn't have to be declared unlike other languages.
- Can be multi-dimensional. Separate the elements by commas. Example \$array[0,2,3].

Arrays will become clearer with an example. Open TGB and run your game. Enter the following into the console.

Console

```
$array[0] = "element 0"; //arrays start at element 0
$array[100] = 100; //can be any variable type in the same array
echo($array[0]);
echo($array[100]);
echo($array[1000]); //displays "".
```

```
==>$array[0] = "element 0";
$array[100] = 100;
echo($array[0]);
echo($array[100]);
echo($array[1000]);
element 0
100
```



Explaining the Code

- Line 1 + 2: Scripting is very flexible allowing an array to hold multiple types of variables. (Booleans, Floats, Strings)
- Line 2: You don't have to specify the size of the array like other languages; it expands it when necessary.
- Line 5: If you call an array slot that has not been defined the array returns "".